A Study of Leaping in Prosimian Primates

September 1992

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor of Philosophy by William Irvin Sellers

Contents

A study of leaping in prosimian primates

By William Sellers

Abstract

This study investigates the biomechanics of leaping in a group of six prosimian primates: *Microcebus murinus*, *Lemur catta*, *Cheirogaleus major*, *Mirza coquereli*, *Galago garnettii* and *Galago moholi*. They cover a 40 fold mass range and include animals from the three commonly recognized distinct prosimian leaping categories. The animals were filmed leaping under controlled captive conditions to obtain kinematic and kinetic data. This data was used to formulate a predictive model for leaping to enable the analysis of internal forces.

As an integral part of this project, a large amount of technical development work was done to produce a complete, computerized video gait analysis system. This uses digital image storage and on-screen calibration and measurement to enable flicker free analysis at a 0.02s interval. The system is capable of 3D reconstruction from twin camera systems at arbitrary positions and real-time 3D animation of a solid rendered model.

Counter to expectations, except for *Galago moholi*, the animals were found not to use the 45° takeoff angle predicted by a maximum efficiency hypothesis except for very long leaps. Even for *G. moholi*, there was considerable variation in leaping angle, though this variation has only a minor energetic cost. A constant force model was found to be a reasonable predictor for the observed leap parameters. Hind-limb extension was seen to show strong positive allometry with increasing body mass. The predictive model indicated that leaping was mainly hip driven in all the animals with appreciable negative work being performed at the knee joint in all the animals except *Galago moholi*. It also showed the importance, in this model, of torque about the takeoff point.

Acknowledgements

I would like to thank the following people for their help and guidance during the course of this project: Robin Crompton, Michael Günther, Russell Savage, Professor Wood and the other members of the **Primate Evolution and Morphology Group** at Liverpool University; McNeil Alexander at Leeds University; Nick Ellerton and his staff at Chester Zoo; Elwyn Simons, Lousie Martin and the animal technicians at Duke University Primate Centre; Jean-Jacques Petter, Emanuelle Goix and the keepers at Paris Zoo. I would also like to specially mention to my stars: Bitters, Rapunzel, Seritra, Tuff and Viburnam.

There are also a number of other people whom I would like to thank for helping me retain, and at times lose, my sanity (they will understand this cryptic reference).

This work was funded by grants awarded to Robin Crompton from the Liverpool Research Development Fund, the Hasselblad Foundation and the Science and Engineering Research Council.

Si je vous raconté ces détails sur l'astéroïd B 612 et si je vous ai confié son numéro, c'est à cause des grandes personnes. Les grandes personnes aiment les chiffres. Quand vous leur parlez d'un nouvel ami, elles ne vous questionent jamais sur l'essentiel. Elles ne vous disent jamais: «Quel est le son de sa voix? Quels sont les jeux qu'il préfèr? Est-ce qu'il collectionne les papillions?» Elles vous demandent: «Quel âge a-t-il? Combien a-t-il de frères? Combien pèse-t-il? Combien gagne son père?» Alors seulement elles croient le connaître. Si vous dites aux grandes personnes: «J'ai vu une belle maison en brique roses, avec des géraniums aux fenêtres et des colombes sur le toit...» elles ne parviennent pas à s'imaginer cette maison. Il faut leur dire: «J'ai vu une maison de cent mille francs.» Alors elles s'écrient: «Comme c'est joli!»

Ainsi, si vous leur dites: «La preuve que le petit prince a existé c'est qu'il était ravaissant, qu'il riait, et qu'il voulait un mouton. Quand on veut un mouton, c'est la preuve qu'on existe», elles hausseront les épaules et vous traiteront d'enfant! Mais si vous leur dites: «La planète d'ou il venait est l'astéroîde B 612», alors elles seront convaincues, et elles vous laisseront tranquille avec leurs questions. Elles sont comme ça. Il ne faut pas leur en vouloir. Les enfants doivent être très indulgents envers les grandes personnes. (Saint-Exupéry 1946)

A Study of Leaping in Prosimian Primates

Introduction

The object of this project was to study the design of a locomotor system in the context of its ecology and morphology[1]. Leaping was chosen for several reasons: there are some clear morphological correlates between leaping proclivity and, for example, intermembral index[2] (Walker 1974); it as a form of locomotion that leads to very high forces on the skeleton compared to other forms of locomotion (Calow and Alexander 1973) and so is more likely to elicit structural adaptations[3]; in addition, leaping is a relatively simple form of locomotion, obeying well understood ballistic[4] principles (eg. Gibbs 1990, Norton 1987).

Locomotor Function

Primates are popular animals in which to study locomotion: for an order with a small number of species, they show a very wide range of locomotor behaviours, with only flight and burrowing being entirely missing, and

---

[1]The concept of a relationship between form and function stems from the early attempts to understand the world in a rational way postulated by Plato (428-348 B.C.) and continued by Aristotle (384-322 B.C.). To Plato, the form of a structure, biological or otherwise, could be understood from its function, since it was the function that dictated the form. However, the philosophies of the Greek thinkers considered that ideal generalizations were true and unchangeable and variation was illusion: an imperfect reflection of reality. Whereas, the more modern concept is that the generalization is merely a convenient reflection of the pluralities of reality (Strickberger 1990).

[2]The intermembral index is the ratio of the ulna and humerus length to the tibia and femur length. Its value is generally lower in primarily leaping species.

[3]Prost, when attempting to establish a methodology for summarizing the locomotor behaviour of a species, contrasts "critical locomotor habits" with "frequent locomotor habits", stressing that the frequency of a behaviour may not be a valid indicator of its adaptive importance to the animal (Prost 1967). Leaping may be relatively infrequent, but of extreme importance both anatomically and ecologically.

[4]Ballistics is the 'science of projectiles', and has been studied with a great deal of enthusiasm ever since the first *ballista* was used to destroy the walls of a castle.

swimming only represented in a very minor way (Napier and Napier 1985). Among the primates, the prosimians show the greatest degree of specialization for leaping behaviour. It includes slow quadrupedal animals that are never seen to leap such as *Perodicticus potto* (Charles-Dominique 1977) and expert leapers such as *Galago senegalensis* which is reckoned by some authors (eg. Schmidt-Nielsen 1983) to hold the record for a standing vertical jump of 2.25m under well controlled conditions (Hall-Craggs 1965).

Notes and observations on the locomotor behaviour of prosimian primates have been made sporadically since the late 19th century (eg. Shaw 1879, Thomas 1896). However, until, the general acceptance of the locomotor classification proposed by Napier and Napier (Napier and Napier 1967), quantitative descriptions of locomotion were hampered by ambiguities in descriptive terms. Indeed, even subsequently, there have been relatively few quantitative studies of general locomotion which has led workers such as Oxnard, Crompton and Lieberman (Oxnard et al. 1990), requiring comparative activity frequency data, to devise a semi-quantitative scoring system based on the descriptive terms found in the literature.

The term **vertical clinging and leaping** was coined in 1967 (Napier and Walker 1967) to describe the general form of leaping in primates. The locomotion of *Tarsius bancanus* is perhaps the best example of this form of locomotion with the animal leaping between vertical supports for 61% of the time (Crompton and Andau 1986). However, this term is somewhat too restrictive, and has been expanded on morphological and behavioural grounds to include 3 sub-divisions: indriid-type, epitomized by *Indri indri*; galagine-type, for example *Galago senegalensis*; cheirogaleine-type as in *Microcebus murinus* (Oxnard et al. 1981a, b).

Locomotion was initially studied as an isolated phenomenon (eg. Marey 1874, Muybridge 1899) in laboratory environments. The gross correspondence between overall body proportions and locomotor behaviour was soon recognized by anatomists (eg. Mollinson 1911, Schultz 1930). Ripley's work in 1967 (Ripley 1967) treated locomotion as just another form of behaviour that could not be divorced from other behaviours such as play, feeding and social interaction. In addition, workers such as Napier and Napier (Napier and Napier 1967) started stressing the importance of the structural aspects of the environment to primate locomotion: in particular its continuity or discontinuity. Subsequent fieldworkers recorded locomotor data in the context of support use and overall behaviour (eg. Bearder and Doyle 1974, Fleagle 1976, Charles-Dominique 1977).

A parallel approach to these classical comparative studies is the study of biomechanics[5]. Here, two types of studies stand out (Emerson 1985): investigations into the effects of scaling in animals, specifically jumping in this case; analyses of muscle forces, bone stress and energy storage.

The effects of changes in body mass were first postulated by the ancient Greek philosophers, and re-iterated by renaissance scholars, but perhaps the classic paper on scaling as it might affect the locomotor system was based on a "Friday Evening Discourse" given by Hill at the Royal Institution in the autumn of 1949 (Hill 1950). He showed that geometrically similar animals would be expected to leap the same distance regardless of size. This has been shown to be largely true in a

[5]Biomechanics is simply the application of engineering mechanical principles to bilogical systems. The first real "biomechanicians" were probably Leonardo DaVinci, Galileo, Lagrange, Bernoulli and Euler (Winter 1990). Borelli was also an extremely important early pioneer. For a more generalized definition of the term, see the article by Hatze in the Journal of Biomechanics (Hatze 1974).

very broad sense: Schmidt-Nielsen compared the maximum jumps of animals varying in mass by a factor of 10^8 (Schmidt-Nielsen 1983). On a more specific level however, for example in frog species of differing sizes (Emerson 1978), it can be shown that this prediction does not always hold up. In this case, absolute jump distance increased with body size. This implies either that the basic assumption of Hill's model, geometric similarity, may be wrong, or that some other aspect of the locomotor physiology may not scale appropriately. Frog body shape does appear to be conserved for different masses (one of the reasons why frogs were chosen for this study), but later work (Sperry 1981) has shown that, unlike the situation in mammals, the number of muscle fibres per unit area of muscle increases with increasing body mass in frogs which may account for the discrepancies.

In addition, the effects of scale mean that the power required per kilogram body mass for a given jump distance increases as body size decreases (Bennet-Clark 1977), so that at some point, as animals reduce in size, it is no longer possible to leap a given distance by relying solely on direct muscle action. For small animals, long leaps require some form of elastic storage mechanism, for example the catapult action of a flea (Rothschild et al. 1972) using the elastic energy storage properties of resilin (Weiss-Fogh 1960).

Analyses of muscle forces, bone stress and energy storage have been performed in several ways: in the first instance, simple observation of the movements of the segments of the hind-limb by high-speed cine photography (eg. Hall-Craggs 1964) extended by using cine-radiography to observe the actual positions of the bones rather than the limb outlines (Jouffroy and Gasc 1974). Incorporation of muscle physiological and anatomical data, and calculation of forces from observed accelerations, is

the next step in this analysis (Hall-Craggs 1977). An external force plate was used to greatly increase the accuracy of internal force measurements to investigate the jumps of frogs (Calow and Alexander 1973), a dog (Alexander 1974) and the kangaroo (Alexander and Vernon 1975). To date, perhaps the most complete investigation, using high speed cinematography, a force plate, and telemetered electro-myography[6] all combined, is the work by Günther on *Galago moholi* (Günther 1989).

These latter approaches allow the calculation of the total forces around joints and the overall stresses on the skeleton. However, in most cases in mammals, the actions of muscles around joints are extremely complicated. Individual muscles combine in groups for any given action, with other muscles firing antagonistically to improve joint stability, so that, even with EMG, it is impossible to identify the precise contribution of each muscle at a given stage in a movement. The number of muscles present, (especially if one considers that each muscle may have a number of functionally independent bundles of fibres) is considerably greater than the number required to control the mechanism (Alexander 1983), so that it is impossible to calculate the rôle of an individual muscle without making further assumptions. Internal forces can be measured by appropriate invasive surgery. Strain gauges can be fixed internally to muscle tendons (Riemersa et al. 1988) so that the actual force generated can be measured at all stages of the movement. In a similar fashion, strain gauges can be attached directly to bones (eg. Lanyon and Bourne 1979). However, this approach is not often practicable, and the surgery itself

---

[6]Electro-myography, or EMG, is the recording of the electrical activity of muscles using implanted or surface mounted electrodes. Telemetry is simply measuring remotely, that is using a radio transmitter mounted on the animal to send the signals from the EMG electrodes to a receiver some distance away, to avoid the problem of trailing wires interfering with the animal's movement.

may cause atypical movement. It is clearly unethical to use invasive experimental techniques on endangered species.

Turning the problem around: deciding on the goal of locomotion and then postulating mechanisms that might achieve it, is one way out of this problem. The first attempts to do this were by building machines that could mimic natural movements. For example, the first legged vehicle that could walk by itself under computer control was the "Phoney Pony" built by Frank and McGhee in 1966 (McGhee 1966). The difficulties in building such a machine can help highlight the actual problems in the natural system that it is attempting to copy. In addition, with a sufficiently realistic model, measurements can be taken from the model, and these will have some bearing on the values to be expected in the real world. With the recent improvements in computing technology, it is no longer necessary to actually build working models for systems. They can be simulated in an entirely abstract form. Early work was done with specifically written software with highly simplified models of limb movement (eg. Townsend and Seireg 1972), but now, general purpose predictive dynamic software packages are available that can be used to model any mechanical system (eg. DADS 1989, ADAMS 1990). ADAMS in particular, has been specially adapted to allow modelling of human movement with a special pre-processor called ANDROID. Currently, much of the predictive work is being done in the fields of robotics (Paul 1981) and in computer aided animation (Armstrong and Green 1985). Simulations of simple biological movements have been reasonably successful, but because of the severe constraints required for a true forward solution model, more complicated forms have not yet been validated (Winter 1990).

Locomotor Ecology

In a paper written in 1963, Tinbergen suggests that the ethologist should attempt to answer the questions of the causation, development, survival value and evolution of a behaviour pattern (Tinbergen 1963). These questions have come to be known as the "Tinbergen Why's": the first two involve looking for the *proximate* causes and the second two, the *ultimate* causes of an observed behaviour. Biomechanics specifically answers the causation question: how does an animal perform a particular action? It can also have some bearing on development, since it is known that bones are remodelled depending on the forces they experience during life (Curry 1984). However, questions about survival value and evolution must be viewed in the context of the other behaviours of the animal and its environment. This is the purpose of locomotor ecology.

Evolution is, to a great extent, driven by natural selection, as described by Charles Darwin (Darwin 1859). Natural selection tends to maximize an individual's inclusive fitness[7]; that is, not only its own reproductive success, but also that of its kin (Hamilton 1964). In other words, its total genetic contribution to subsequent generations. In practice however, measuring inclusive fitness is extremely difficult. Simple reproductive success is usually a good measure (Grafen 1982), but even this requires long term studies and because of the complexity of the natural environment it is almost impossible to measure the change in fitness that derives from a specific behaviour pattern.

[7]Fitness, as in Darwin's "survival of the fitest", was not rigorously defined by Darwin himself. However, it is now widely recognized by biologists to be a measure of the capacity to produce offspring (McFarland 1985).

One approach to this problem is to evaluate the costs and benefits associated with a particular behaviour: most behaviours (and indeed most aspects of an animal's biology) are a compromise between their merits and their disadvantages. Animals are faced with alternative options and they need to make a trade-off between the costs and benefits associated with each. From an evolutionary standpoint, it would be expected that a well adapted animal would make optimal choices to maximize its inclusive fitness, so that measuring the costs and benefits of a particular behaviour can allow some estimate of the real goal of measuring fitness.

Analyzing costs and benefits requires the separation out of the increments and decrements to inclusive fitness ascribed to each aspect of the animal's internal state and behaviour. The total specification of an animal in these terms is its *cost function* which can be defined as the "specification of the instantaneous level of risk incurred by (and reproductive benefit available to) an animal in a particular internal state, engaged in a particular activity in a particular environment" (McFarland 1977, 1985).

It is unrealistic to try to measure the complete cost function for an animal, but one can try to identify aspects that are important. Considering the rôle of locomotion, a number of hypotheses can be postulated as to how its design may lead to the least cost, or greatest benefit:

(1) An animal can be designed to minimize the energy cost of its locomotion. The less energy the animal has to use to move around, the more energy it will have available for other activities, such as reproduction, and the less food it will need to survive.

(2) An animal can be designed to minimize the time it spends "locomoting". In other words, it maximizes its locomotor performance. This may be because time is the major limiting factor for its other activities: it has plenty of energy, but insufficient time to do all that it would ideally want to do. Alternatively, improving performance may enable the animal to do things that it otherwise could not: such as a cheetah being able to catch its prey; or a rabbit being able to outrun its pursuers.

(3) Or, an animal may design its locomotion to maximize its safety. This may be accomplished by reducing the risk of physical injury or by reducing the chance of predation.

Obviously, in a real world situation, the design has to take all these considerations into account to some degree. The optimal design will be a compromise, but certain factors will generally be considerably more important than others. Experiments on bumblebees (Heinrich 1979) indicate that when the animals have to travel some distance to find productive flowers time is the limiting factor: it is worthwhile to expend energy in order to save time. When foraging on relatively unproductive flowers, or when foraging at low temperatures, energetic efficiency becomes more important, and the bee slows down in order to save energy. Observations on squirrels eating chocolate chip cookies in a park (Lima et al. 1985) show that the animal is not primarily concerned with either time or energy: the squirrel grabs the cookie and retreats to a tree to eat it. This behaviour has been interpreted as minimizing predation risks.

The various benefits and costs associated with leaping locomotion can be summarized as follows:

Leaping, as a form of horizontal locomotion, is moderately expensive (Walton and Anderson 1988). Walking, in an appropriately designed animal on a level surface, is extremely cheap, with energy recovery rates per stride as high as 70% (Cavagna et al. 1977), and galloping and hopping allow elastic energy recovery between cycles which reduces their energetic cost (Heglund 1985). In a three dimensional environment, leaping may provide a relatively cheap way of gaining height. It is certainly much quicker than climbing.

The speed of leaping depends on the takeoff angle. At its most efficient angle of 45°, it is relatively slow compared with galloping (Günther et al. 1991). However, at shallower angles, it compares more favourably. In fact, galloping can be described as a series of connected, shallow leaps. The difficulty of leaping any appreciable distance at a shallow angle is probably one of precision. At small takeoff angles, a small change in angle has a very large influence on the distance leapt, and an error that led to too small a distance being covered could have very serious consequences.

The main advantage of leaping is that it enables the animal to cope with discontinuities in the substrate. Thus, *Galago moholi* can leap from tree to tree, instead of having to climb down one tree, move across the ground to the other, and climbing back up the second tree. This latter approach would almost certainly be both more expensive energetically and slower (Crompton 1984).

As far as physical injury goes, leaping is extremely risky. There is only very limited scope for correcting a bad leap in mid-flight since no external forces can be brought into play, and the animal risks missing the destination substrate completely, which, in an arboreal environment, could lead to a possibly fatal fall (Cartmill 1985). Compared to this, the

risks of missing ones footing when galloping along a branch would seem to be lower. However leaping is a good way of avoiding predators. Arboreal predators, including snakes, may not be able to leap so that the primate can escape by leaping from one branch to another, or to another tree. For aerial predators, such as raptors, leaping from tree to tree does expose the animal. However, moving along the ground between trees leads to exposure to terrestrial predators as well as aerial ones, and would generally be much slower (Crompton 1980).

Leaping is an extremely good means of predator avoidance. It combines a rapid start with poor direction predictability (Emerson 1985). A non-leaping animal on a branch is restricted to escaping along that branch. Its escape direction is easily predicted so that the chances of capture are higher. A leap can be off in any direction and is therefore very much harder to predict. The only disadvantage is that during the flight phase of the leap, the animal is totally committed to its choice. It has no option of changing mid-way.

In addition, leaping has also been suggested as a means of prey capture. Insects can be captured in mid-air, or a sudden pounce can be used to catch animals on the ground (Crompton 1984).

Mechanical Considerations

Leaping is a ballistic form of locomotion. During the takeoff phase, the animal applies a force to the substrate to accelerate it to the required takeoff velocity. During the flight phase, it acts as a projectile, moving through a parabolic path dependent entirely on the initial takeoff angle and velocity. The effect of air resistance depends on the animal's size, but is minimal for the prosimian range (Bennet-Clark 1977). There is some evidence that gliding may be used to extend the flight path a small amount (Günther 1991). Drag and relative rotation of parts of the body are used to adjust orientation during flight (Dunbar 1988).

For efficient leaping, an animal should have very light limbs to minimize the energy lost in internal kinetic energy (Alexander 1983). The total mass of the animal should also be kept as low as possible. It is also important to have sufficient precision to propel the centre of mass in a straight line throughout the takeoff phase. This should also be chosen to be 45° (Emerson 1985). Any deviation requires extra energy expenditure to correct. The choice of substrate will also affect the energy cost. Ideally, a rigid start support should be used so that no energy is wasted bending the branch, and a flexible landing support should be used to dissipate the kinetic energy of the jump externally without the animal needing to do any negative work.

For maximum performance, the animal needs long, strong hind-limbs relative to its body mass. Takeoff velocity depends both on the force applied and the length of time for which the force is applied. Longer limbs prolong the contact phase and lead to a higher takeoff velocity and hence a longer leap (Emerson 1985). The ideal takeoff angle depends upon whether the animal is trying to maximize speed or jump length. 45°

leads to the maximum possible jump distance, but for maximum speed, the shallowest angle that allows the animal to leaps the required distance is needed. By choosing a suitably "springy" start substrate, the animal may be able to store up energy from a previous jump and use the branch like a springboard to extend the maximum range of a leap (Günther 1991). Landing on a rigid substrate is also generally quicker since the animal is able to recover its poise faster.

For safety, an animal needs stronger, shorter limbs to reduce the chance of breakage. Obviously, higher performance jumps are intrinsically more risky, and a flexible, cushioning destination is preferable. Falling damage is a real risk for arboreal primates (Alexander 1989).

As mentioned before, the overall design must be a compromise among these features. For example: light hind-limbs will be efficient; long and powerfully muscled ones will produce the longest jumps; extremely robust limbs will be safest. There is a trade-off between the requirements for the different design goals.

Also, leaping adaptations can affect other related locomotor and non-locomotor activities that the animal might wish to perform. For example, efficient walking depends on the stride frequency being close to the natural pendulum frequency of the limbs (Hildebrand 1985). Leapers tend to have elongated hind-limbs for the reasons mentioned above. This makes them less efficient walkers since the natural pendulum frequencies of their fore and hind limbs differ.

Aims

The aims of this study were five fold:

(1) The development of a computerized system for the kinematic and kinetic analysis of locomotion. This system should be as flexible as possible, allowing two and three dimensional measurements of arbitrary models. The main source of raw data would be video tape, so that facilities to make the measurement of still video images as easy, reliable and rapid as possible were also required. Data output from the package was to be smooth running, three dimensional animations; on-screen or printed graphical displays; numerical data in suitable formats for import into spreadsheets and statistical programs.

(2) The design and use of an experimental protocol to investigate the mechanics of leaping. This had to cope with the restrictions imposed by working with endangered species: very limited manipulation of the animals, and only minimal disturbance to their cage environments.

(3) To test the hypothesis that the animals would generally choose to leap in such a fashion as to minimize their energy expenditure by looking at the takeoff angles used by the animals.

(4) To formulate and test the predictions of a number of simple leaping and scaling models by looking at the effects of varying body mass and leap distance on a number of measured kinematic and kinetic leap parameters.

(5) To develop and test a predictive model of leaping that could in future be used as an investigative tool for non-observed behaviours such as leaping in lorises and in fossil and sub-fossil prosimian species.

Methods

The basic technique I have chosen to use to look at the mechanics of leaping is kinematic analysis. This is the study of motion by looking at the positions of all the components of a system with respect to time. This technique was pioneered by Muybridge in the 1880s (Muybridge 1889) who photographed a large number of animals using a series of still cameras triggered electrically. He had been asked to solve the vexing question as to whether all four limbs of a horse left the ground during a trot. He was able to get the first sufficiently good photograph on a freshly made wet collodion plate to freeze the motion in mid-stride, and to reveal that all four legs did indeed leave the ground at once. The early pioneers of high-speed photography followed on in a similar vein qualitatively describing things that happen too quickly for the unaided eye to follow.

The natural extension of this is to obtain quantitative data. This involves calibrating the optical system being used and measuring the positions of interest on the resultant frames. This is a relatively simple, if rather time-consuming exercise since there can be many thousands of frames to measure. Much more recently, with the advent of cheap and powerful computing facilities, attempts have been made to automate the measurement process. Various systems are available that can recognize markers, but these are still extremely expensive and can be a little unreliable. This will doubtless all change in the not too distant future.

Presently the options for filming are between video and photographic film. Each has merits and disadvantages Video is cheap, the camera gives instant feedback of success or failure, timing is very precise, and they can often function in relatively low light levels. However, normal video has a low resolution, and the framing rate is also low even though the shutter

speed can be very high.[8] Photographic film is more expensive and one has to wait for the film to be developed before the quality of any particular sequence can be assessed. It can operate at much higher framing rates, though the shutter speed is often a direct function of this rate so that a camera capable of very high rates may be required simply to freeze the motion. The resolution of the eventual picture is very much higher, though the light level required is much higher too. In addition, the framing rate usually has to be measured by incorporating timing marks. The choice between the two is thus complicated and it is not possible to generalize on the suitability of either. Non-standard video formats are overcoming some of the disadvantages of video. Both high speed and high definition cameras exist, though not, unfortunately, high speed and definition in the same unit.[9] In this study, since some of the animals were nocturnal zoo specimens, it was a requirement to cause as little disturbance to the animals as possible. This effectively ruled out using movie film because it would have required more too much light. This also gave scope to try out some innovative analysis techniques which will be discussed later.

An attempt was also made to try 3D kinematic analysis. This is basically an extension of still stereo-photogrammetric techniques. However, it is not without its difficulties. Firstly, still photogrammetric cameras are extremely precisely built, often using glass photographic plates to insure a completely flat focal plane. They are carefully calibrated so that their

---

[8]The framing rate is the number of discrete frames exposed each second. The shutter speed is the amount of time any particular frame is exposed to the light. Thus a high shutter speed is required to freeze movement, (even a still camera can freeze movement) but a high framing rate is required to slow it down.

[9]For an excellent review of filming techniques, and of other methods of obtaining kinematic data, see Winter 1990.

optical parameters can be described mathematically. This level of precision is just not possible with moving pictures with out very great expense. Secondly, if two or more cameras are being used to produce the images, they need to be exactly synchronized so that they show the moving subject frozen at exactly the same instant. The first problem has been overcome by a variety of mathematical approximations that allow calibration of the cameras in-situ, for example, the direct linear transform (DLT) equations (Shapiro 1978). The second cannot yet be easily countered using film cameras, but with video, a technique known as genlocking is used to electronically synchronize all the cameras. This is a standard broadcast technique because video signals need to be synchronized to allow mixing.

Historically, there are a number of other techniques for analyzing motion. Their use currently is restricted to specialized applications. Strobe lights can be used to illuminate a subject moving in front of a still camera with its shutter held open. This produces a set of overlaid images of the moving target; one for each flash of the strobe. The images produced tend to be fairly poor quality, and the flashing strobe can be off-putting for the subject. However, it is very cheap, and the effective framing rate that can be achieved is comparatively high. Alternatively, various mechano-receptors can be attached to the subject's body. These can relay telemetered signals indicating position, bending, velocity and acceleration. The main advantage of these is that the measurements can be read directly by a computer. Also, the measurements are continuous rather than discrete which greatly helps mathematical analysis later on. Indeed, one of the biggest problems, that of getting accelerations from position data, is completely alleviated by measuring the acceleration directly. The "down-side" is that there is a great deal of interference with the subject wiring up all the sensors, and they are limited to

measuring exactly the positions where they are attached. Their accuracy depends on their mounting method. Direct bone mounting on experimental animals is extremely precise, but skin attachment to measure a human hip joint is much less so.[10]

X-ray photogrammetry is an obvious extension for skeletal motion analysis. It allows the joints to bee seen directly without having to infer their positions from external appearances. It has been done for a very limited number of primate species, and only once for a leaping prosimian (Jouffroy and Gasc 1974) but is currently limited by the expense and the limitations of the equipment. There are also problems with the radiation dosage[11] required and the limited size of the field of view.

The Animals

Six prosimian species were used in the study. They have a 40 fold variation in body mass and there is at least one representative from each of the three main prosimian leaping classifications. They are described here in order of leaping proclivity with the most frequent leaper first (Oxnard et al. 1981).

Galago moholi, the lesser bushbaby, (a galagine-type leaper)[12] was filmed at Duke Primate Centre. The individual used in the experiment, *Viburnam*, was an adult male in good health, weighing 0.21kg. The lesser

---

[10]Again see Winter 1990.

[11]Both to the subject animal and to the human experimentor. Jouffroy describes her subject as suffering from "a slight radiodermatitis" after filming. Whilst this sort of exposure will not lead to an unduely large dose to an extremity, if given to the whole body, it is likely to lead to bone marrow depression which is clearly unacceptible when working with endangered animals.

[12]Oxnard and his co-workers sub-divided leaping behaviour in prosimians into three sub-groups depending upon a multivariate analysis of a number of anatomical and behavioural factors. These groups are entitled indriid-type, galagine-type and cheirogaleine-type. (Oxnard et al. 1981, Oxnard et al. 1990)

bushbaby is widespread in sub-saharan Africa in a wide range of habitats from sea level to 1,500m (Bearder and Doyle 1974). It is about 16cm long and the mean wild caught weight is about 0.25kg (Rasmussen and Izard 1988). It is mainly a gum feeder, though it also eats small animals (Oxnard et al. 1990).

Microcebus murinus, the grey mouse lemur, (a cheirogalein-type leaper) was filmed at Duke. The subject was *Bitter*, a 0.066kg male. This species is primarily insectivorous, though with a reasonable quantity of fruit in its diet. It inhabits dense tangles of foliage in the forest fringe habitat. These tangles can, however have a markedly different vertical position, being at ground level in secondary forest, but 30m up in the canopy in primary rainforest (Oxnard et al. 1990). An average weight is approximately 0.08kg (Harvey et al. 1987).

Mirza coquereli, Coquerel's mouse lemur, (a cheirogalein-type leaper) was also filmed at Duke. The individual studied was called *Seritra*, was female, and weighed 0.35kg. It lives in the lower 6m of the forest, moving mainly on horizontal supports. Its diet consists mainly of animal items and fruit (Tattersall 1982). Interestingly, in the dry season, it survives on the secretions of homopteran larvae. An average weight is about 0.3kg.[13]

Galago garnettii, the greater bushbaby, (a galagine-type leaper) was also filmed at Duke Primate Centre. *Tuff*, the animal studied, was male and weighed 1.13kg. It has a more restricted range than *Galago senegalensis*, being restricted to dense evergreen forest and riparian bush in south-central Africa where there is a plentiful supply of fruit (Bearder and Doyle

---

[13]*Mirza coquereli* and *Galago crassicaudatus* have the same recorded leaping frequency (Oxnard et al. 1990). I have put them in this order because in my experiments, *Mirza coquereli* was the more enthusiastic leaper.

1974). *Galago garnettii* has an average weight of about 0.9kg (Nash and Harcourt 1986). Its diet consists of about equal quantities of fruit and animal items (Oxnard et al. 1990).

Lemur catta, the ring-tailed lemur, (an indriid-type leaper) is rather larger than the other animals in this study, and is the only diurnal one. A group of these animals were filmed at Chester Zoo where they had been encouraged to leap across a 2.2m horizontal gap to a feeding site. It was not possible to identify individuals in the film, so a nominal figure of 2.7kg was used for the mass (Harvey at al. 1987). *Lemur catta* has been widely studied, but its behaviour appears to be very flexible. It eats mostly fruit, leaves and flowers (Oxnard et al. 1990). It is fairly widespread in Madagascar in moist and dry forests, but not in open country (Napier and Napier 1985).

Cheirogaleus major, the greater fat-tailed dwarf lemur, (a cheirogalein-type leaper) filmed at Duke Primate Centre. *Rapunzel* was female and weighed 0.34kg. Very little is known about these animals in the wild. However, it lives in tropical rainforest in Eastern Madagascar, where it utilizes mostly large diameter, horizontal branches and is largely frugivorous. Its mean mass is 0.35kg (Napier and Napier 1985).

Filming

Most of the filming was performed at Duke Primate Centre in North Carolina using the experimental setup described below. In addition, some film for *Lemur catta* was taken at Chester Zoo.[14]

The subject was housed in a relatively large, empty cage with only two supports that could be conveniently used by the animal. The camera was placed so that the field of view covered the area on the first support from which, it was hoped, the animal would leap. During the course of the experiment, the second support was positioned at varying distances from the first support, and the animal was filmed whenever it leapt between the two supports. In this way, horizontal leaps with no height change over a known distance could be reliably measured. Restricting the number of available supports, and positioning these supports well off the ground improves the chances of the animal leaping the whole distance. The problem with floor mounted force plate type experiments with untrained animals is that the animal tends not to jump the desired distance, but hops along the floor instead (M. Günther personal communication). Food rewards were placed on the second support to encourage the animal to leap, and the animal's nest box was placed near the first support to encourage it to go there. Animals were moved from their normal cages into the experimental cage, and leaps were recorded over the next several weeks, as the animal explored its new environment. In general, there was little activity for the first few days, but eventually,

---

[14]There was no experimental setup in this case. The animals habitually leapt across a 2.2m gap in a wooden causeway to get from an island to a feeding site. This provided an ideal position to film this animal performing a relatively large leap except for the fact that the camera operator had to stand in 5ft of cold water in the lake to get an orthogonal view of the jump!

the animals would leap between the supports as required, though not particularly frequently.

The camera used was a standard portable CCD[15] video camera with a 1ms second exposure. As the animals were nocturnal, the lighting in the enclosure was reverse cycled, with bright, fluorescent tube lighting from 9 pm to 9 am, and rather dimmer, incandescent, red filtered lighting from 9 am to 9 pm. The light level was rather lower than ideal for the camera, but minimized the stress on the animals and caused them to be more active.

The camera was calibrated for each series of measurements by filming a reference object of known dimensions at the start of the session. This consisted of a rigid rod with a series of markers fixed at known intervals along it. The field of view was carefully chosen to be as small as possible, to maximize the resolution, and yet include the whole of the takeoff phase of the leap. The camera was positioned outside the cage, three metres from the takeoff position so that the effects of parallax could be ignored.[16] The lens quality was judged to be sufficiently good, given the level of other measurement uncertainties, to allow us ignore optical distortion. Timing accuracy of video cameras is extremely good (considerably less than 1% error), and the framing rate was therefore assumed to be exactly 50 fields per second.[17]

---

[15]A CCD, or charge coupled device, is a solid state array of optical sensors positioned at the focal plane of the lens. All the receprors act in parallel to give a very fast snapshot of the image. Older cameras use evacuated glass "camera" tubes that have a fluorescent screen that is read by scanning with an electron beam. These can have very high light sensitivities but the scanned reading method means that the image produced is not instantaneous.

[16]The parallax error is the ratio between the camera distance from the subject and the depth of the subject. In this case, it is about 1%.

[17]When filming, it is customary to talk of the number of images recorded per second as frames per second.. However, a PAL video signal overcomes problems with limited

In addition, attempts were made to record the jump in 3D. Here, two cameras were genlocked together and the signals were fed through a video mixer and recorded as a split screen image on a single video recorder. The start position of the animal was arranged to be covered by both their fields of view. Calibration was performed using a reference object constructed out of a small box approximately 4cm long and wide and 2cm deep made of cast aluminium. 5 of its 6 faces had telescopic aerials attached to produce a structure similar to the diagram:

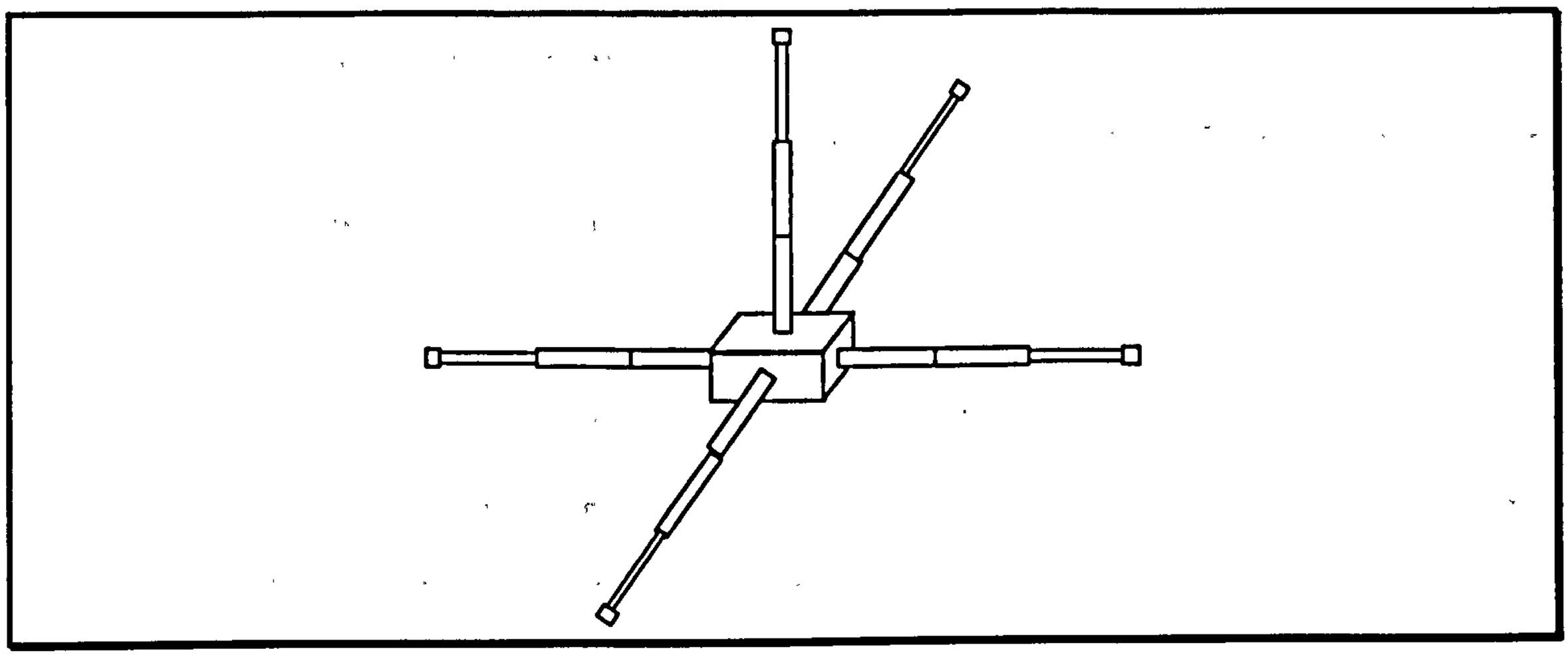


Diagram of the reference object used for the 3D measurements.

The base of the box was attached to a photographic tripod so that the whole object could be maneuvered into a position where it was seen by both cameras. Markers were fixed at measured intervals to the telescopic aerials which could be extended to fill as much of the common field of view as possible. The minimum number of known points required for DLT reconstruction is six. Tests have shown that the more known points

---

bandwidth, resolution and flicker, by incorporating two *interlaced* fields in each frame. The framing rate is 25 frames per second, but with CCD cameras switched to their high shutter speed modes, each field is a separate entity recorded at 50 fields per second, and sent to the video recorder in pairs as the complete interlaced frame.

measured, the better, and that these should extend throughout the volume of interest (Wood and Marshal 1986). Unfortunately, the results from this part of the study were extremely disappointing. When the resultant film was viewed, it was discovered that the common field of view was such a small portion of the video frame that it was impossible to pick out the limb positions of the animals.

The major experimental difficulty was the delay between activating the video recorder, and actually starting to record the action. Due to the duration of the experiment, it was not possible to keep the video recorder running continuously, and this meant that I had to become very proficient at predicting what the animals were about to do (this was not an easy task).

The video signal was initially recorded onto a portable VHS recorder, and back at the laboratory, this was copied onto a Hi-Band SP U-matic tape for editing. Kinematic analysis requires sequential access to each frame of the film. Various methods were tried for this. In the first instance, the single frame advance of a VHS video recorder was used and measurements were made directly from the monitor screen. This proved unsatisfactory since, though the still frame quality was reasonably good, the time resolution was reduced to 25 fps since the recorder stepped by frames and could not resolve the individual fields. The VHS recording system records each frame as a single rotation of its recording head so that it is very easy to step between frames automatically. On still frame mode, one field of the image is removed from the signal so that the image on the screen is completely still, and not a mixture of two fields 1/50th of a second out of step. It is generally not possible to get access to the other field. To resolve individual fields, I moved to using the jog control on the U-matic editing equipment. On a U-matic system, the separate

fields are accessible sequentially, but in this case, the recording system is such that stepping a single field is difficult to do automatically, and the manual jog control means that it is difficult to get the rotating head to line up accurately enough on the tape to get an interference free picture. There was also considerable vertical displacement between individual frames and a problem with the machine's safety cutout coming into operation if a still frame was viewed for too long.

Eventually, both these problems were solved by converting the video signal on the tape to a series of digital pictures on a PC. The computer was fitted with a Matrox PIP-1024 card that was capable of grabbing up to four complete video frames in real-time from a video source. These images could then be split up into the individual fields and displayed and measured on the computer monitor at leisure. Also, since the frames were grabbed from the moving video, the quality was excellent. The only difficulty was that the leaping sequences were generally about 25 frames long. This was overcome by designing what was basically a simplified modem circuit that could convert signals transmitted from the computer over its RS-232 port to audio tones that could then be recorded onto the audio track of the video tape. A program running on the PC counted the synchronization pulses from the video signal on the tape, and this count was dubbed onto the audio track. To digitize a sequence of frames from the tape, it was played through once and the pulse counts were displayed on the computer screen and the start and stop counts noted. The computer was then instructed to digitize in sequence, all the frames between these two counts, and the computer would then prompt the

operator to replay the section of video tape of interest a number of times, grabbing and saving its four frames on each pass.[18]

The kinematic analysis requires that the positions of the joint centres are measured for each frame in a leap sequence. This measurement was performed using the specially written **Gait Analysis Program** software (GAP).[19] This program runs on a Hewlett-Packard graphics workstation, and the image data acquired by the PC is passed over to it via a local area network link. Each image has a resolution of 512 by 256 pixels and has 256 grey levels. GAP allows the user to display the image on the computer screen, optionally expanded up to 1024 by 768 to restore the correct aspect ratio, and points are measured using the mouse driven pointer.

To assist the measurements, and to automate some of the kinematic calculations, the computer program requires some additional information from the user. This is provided in the form of a data file for each set of experiments. This specifies the structure of the model being used. It defines the names of the joints, the names of the segments, and how the joints are linked together by the segments. The same file also contains data on the masses of the segments, the relative positions of the centres of mass of the segments and the moments of inertia about the centres of mass of the segments. The names given are used as user prompts for the measurements required. This file can be created in a word processing package and saved to disk. Although it is quite tricky to set up in the first place, it generally requires very little alteration between experiments.

---

[18]Details of the circuit and the computer software are given in the technical section.

[19]Again, a full description of GAP is included in the technical section, along with details of data file formats, and operating instructions.

As mentioned before, the system is calibrated by filming a reference object of known dimensions. This section of film is digitized to produce a single reference frame. The only requirement is that it should define the origin of the coordinate system[20], and have one other known point. The program asks the user to select the origin from the picture, type in the real-world coordinates of the known point, and select this point from the picture. If necessary, a fiducial[21] point can also be defined. This point is used as a registration to line up individual frames, and should be visible on all frames in a sequence, as well as the reference frame. It is not used for straightforward video work since there is virtually no jitter between frames when they are digitized from the moving film. However, it is useful for video copies of high-speed movie film where there can be noticeable jitter.

The coordinate systems are converted by calculating the offset of the origin, a scale factor, and a rotation matrix:

(1) $$\mathbf{O} = \mathbf{S}_{org}$$

(2) $$S = \frac{|\mathbf{W}_{ref}|}{|\mathbf{S}_{ref}|}$$

(3) $$\theta = \mathbf{S}_{ref} \angle \mathbf{W}_{ref}$$

(4) $$\mathbf{M} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

---

[20]The picture that is actually measured on the computer screen in arbitrary pixel diameter units can be thought of as a mapping of the real-world coodinate system chosen by the experimenter. This coordinate system needs a zero position and a scale of measurement.

[21]The fiducial point is simply a point that is known not to move between individual frames on the film.

Where:

O	is the offset vector
S_{org}	is the screen coordinate vector of the reference origin
S	is the scale factor
W_{ref}	is the world coordinate vector for the reference point
S_{ref}	is the screen coordinate vector reference point
θ	is the angle between the reference vectors
M	is the rotation matrix

Measurements

Having obtained a calibrated film sequence, a set of points to be measured must be defined. This choice depends on a number of factors: the points must be clearly visible on the film; they must relate to underlying skeletal structures; they must make some sort of mechanical sense. The first factor is obvious. If the desired position cannot be seen on the film, then it is not possible to measure it, no matter how desirable this may be. The second and third factors are tied together. Vertebrate body mechanics are extremely complicated indeed and it is essential that a much simpler model of the real system is used so that any meaningful information can be gleaned. To this end, a **link segment model** is constructed (Bresler and Frankel 1950). This will be discussed in more detail in the technical section, but it basically involves dividing up the body into a series of rigid links connected by joints. Each link can be considered in isolation using the information known about its movement to calculate the external forces acting upon it. These external forces must come from its contact with the surroundings, or, through its joints, from other links. In this way, the forces (both internal and external) necessary to produce the observed movement in the whole structure can be calculated. This procedure is know as **inverse dynamic analysis**.

The links chosen, and the points on the film chosen to specify them are as follows:

Link:	**From:**	**To:**
Fore-foot	Toe tip	Mid-tarsal joint complex
Hind-foot	Mid-tarsal joint complex	Ankle joint
Calf	Ankle joint	Knee joint
Thigh	Knee joint	Base of tail[22]
Lower arm	Wrist complex	Elbow joint
Upper arm	Elbow joint	Neck[23]
Head	Nose tip	Neck
Torso	Neck	Base of tail
Tail	Base of tail	Tip of tail

These choices necessarily involve approximations. The base of the tail, as can be seen in lateral X-ray photographs, is reasonably close to the hip joint. The shoulder joint approximation is rather less good, but this is permissible since the rôle of the upper limb in leaping is far less important. In both cases, allowing the upper limb to share the joint between the head and the torso, and allowing the lower limb to share the joint between the torso and the tail considerably simplifies the resultant link segment model. Other authors have split the torso into a head/thorax and an abdomen (or even abdomen/tail) segment (Wells and DeMenthon 1987, Smith 1987). In my film, I was unable to see a reliable position on which to base this division, though mechanically, a split there

---

[22]Nominal position of the hip joint.

[23]Nominal position of shoulder joint.

rather than at the neck might well have been desirable. Also, since none of these authors were attempting inverse dynamic analysis, they had less incentive to simplify the link segment model.

To perform inverse dynamic analysis, the following kinematic measures are required for each link at each time interval:

Linear	Position of centre of mass
	Velocity of centre of mass
	Acceleration of centre of mass
Rotational	Angle
	Angular velocity
	Angular acceleration

The linear measures refer to the displacement in space of the centre off mass of the segment. The centre of mass is the position where all the mass of the whole segment can be considered to act as a point mass. It is defined mathematically later. The rotational measures refer to the internal circular movement of the segment around its centre of mass.

Kinematic Analysis

Signal Processing

The raw data obtained by measuring the joint positions contains a reasonable degree of random sampling error due mainly to the difficulty in accurately estimating the positions of the joint centres. This shows up as high frequency noise superimposed on the position signal. This is not too much of a problem when considering the position data alone, but the process of differentiation needed to calculate the other kinematic parameters of velocity and acceleration tends to amplify this high frequency signal to such an extent that the data becomes useless (Winter 1990).

To get any form of useful secondary data, the original signal needs to be smoothed. There are two common approaches. Firstly, a suitable mathematical function can be fitted to the data: various orders of splined polynomials are often used. Secondly, the data can be filtered using a digital low-pass filter so that the troublesome high-frequencies are attenuated. The former approach produces perfectly smooth curves, and the fitted function can then be differentiated analytically to produce more smooth curves of velocities and differentiation. Unfortunately, the shapes of these curves depend more on the function chosen to fit the data than on the underlying mechanics of the filmed performance. Thus, the results obtained are only of interest if there is some mechanical justification for the function used. The latter approach does not smooth up the data as aesthetically, and certainly, there is still an appreciable noise level when the signal is differentiated. However, the process is much more robust, and the results more closely reflect the mechanics. The only problem can be if the framing rate is not high enough, and then

the degree of filtration required to get a useable signal can flatten out any rapid peaks in the signal. Experimental comparison of these techniques shows that smoothing rather than function fitting is the best approach (Pezzack et al. 1977).

There are a variety of alternative digital filters (Radar and Gold 1967). Initially, a Butterworth second order low pass filter with a 10Hz cutoff was tried. This was applied twice, the second time in the reverse direction, to produce a fourth order zero phase shift filter (Winter et al. 1974). However, although this technique worked well on the central parts of the sequence, it required approximately five frames at each end to stabilize which was unsatisfactory since there were generally only three or four frames after takeoff where the subject was still fully in the field of view. So, instead, a simple unweighted moving average was used. This is symmetrical, so only requires a single pass, and it only loses one frame at each end of the sequence. Its main disadvantage is that, unlike the Butterworth filter, it does not have well defined physical properties. It simply smooths the data in a non-specific way.

The formula used was:

(1) $$s_n = \frac{x_{n-1} + x_n + x_{n+1}}{3}$$

Where:

s_n is the n^{th} smoothed value

x_n is the n^{th} raw value

Differentiation

A simple linear integration scheme was used (this is effectively fitting a straight line between two data points and measuring its gradient). The equation for the differentiation algorithm used is as follows:

(2) $$v_n = \frac{x_{n+1} - x_{n-1}}{t_{n+1} - t_{n-1}}$$

(3) $$a_n = \frac{4(x_{n+1} - 2x_n + x_{n-1})}{(t_{n+1} - t_{n-1})^2}$$

Where:

x_n	position at the n^{th} sample
v_n	velocity at the n^{th} sample
a_n	acceleration at the n^{th} sample
t_n	time at the n^{th} sample

Angular Properties

As well as the positions, linear velocities and linear accelerations of the joints, the angles, angular velocities and angular accelerations of the limbs also need to be calculated. These can be obtained by considering each segment of the animal as a vector. For instance, the thigh can be represented as follows:

(4) $$\begin{pmatrix} x_{hip} - x_{knee} \\ y_{hip} - y_{knee} \end{pmatrix}$$

Where:

x_{joint}	is the x coordinate of a joint
y_{joint}	is the y coordinate of a joint

Then the angle can be calculated as[24]:

$$\theta_{thigh} = \tan^{-1}\begin{pmatrix} x_{hip} - x_{knee} \\ y_{hip} - y_{knee} \end{pmatrix} \quad (5)$$

Once θ has been calculated, then angular velocities and accelerations can be calculated in exactly the same way as for their linear counterparts:

$$\omega_n = \frac{\theta_{n+1} - \theta_{n-1}}{t_{n+1} - t_{n-1}} \quad (6)$$

$$\alpha_n = \frac{4(\theta_{n+1} - 2\theta_n + \theta_{n-1})}{(t_{n+1} - t_{n-1})^2} \quad (7)$$

Where:

θ_n	angle at the n^{th} sample
ω_n	angular velocity at the n^{th} sample
α_n	angular acceleration at the n^{th} sample
t_n	time at the n^{th} sample

Centre of Mass

Each individual segment of the model has a centre of mass. In addition, the overall model has a centre of mass that can be calculated from the positions of the individual centres of mass. The segment centres of mass can be measured or calculated geometrically as described later. The position of the centre of mass of a segment, in a general form, is expressed as it relative distance from one of the joints. Thus, a value of 0.5 means that the centre of mass is halfway along the segment. The

[24]Care must be taken to get the correct angle in this calculation since $\tan^{-1}$ will only give answers between -90° and 90°. Checks need to be made on the direction of the vector by looking at the signs of its x and y component and working out from this which quadrant it is in. Alternatively, the 'C' language function **atan2(y,x)** will produce the correct result.

actual position of the centre of mass in a segment can be obtained from the following vector equation:

(7) $$P_{cm} = P_{joint} + RV_{segment}$$

Where:

P_{cm}	is the position vector of the centre of mass
P_{joint}	is the position vector of the joint
R	is the relative position of the centre of mass
$V_{segment}$	is the segment vector

Knowing the centre of mass position in all the segments in the model allows the overall centre of mass to be calculated (this is the position where the total mass can be considered to be a point mass as far as translational motion is concerned).

(8) $$P_{cm} = \frac{1}{m_t}\sum_{i=1}^{n} P_i m_i$$

Where:

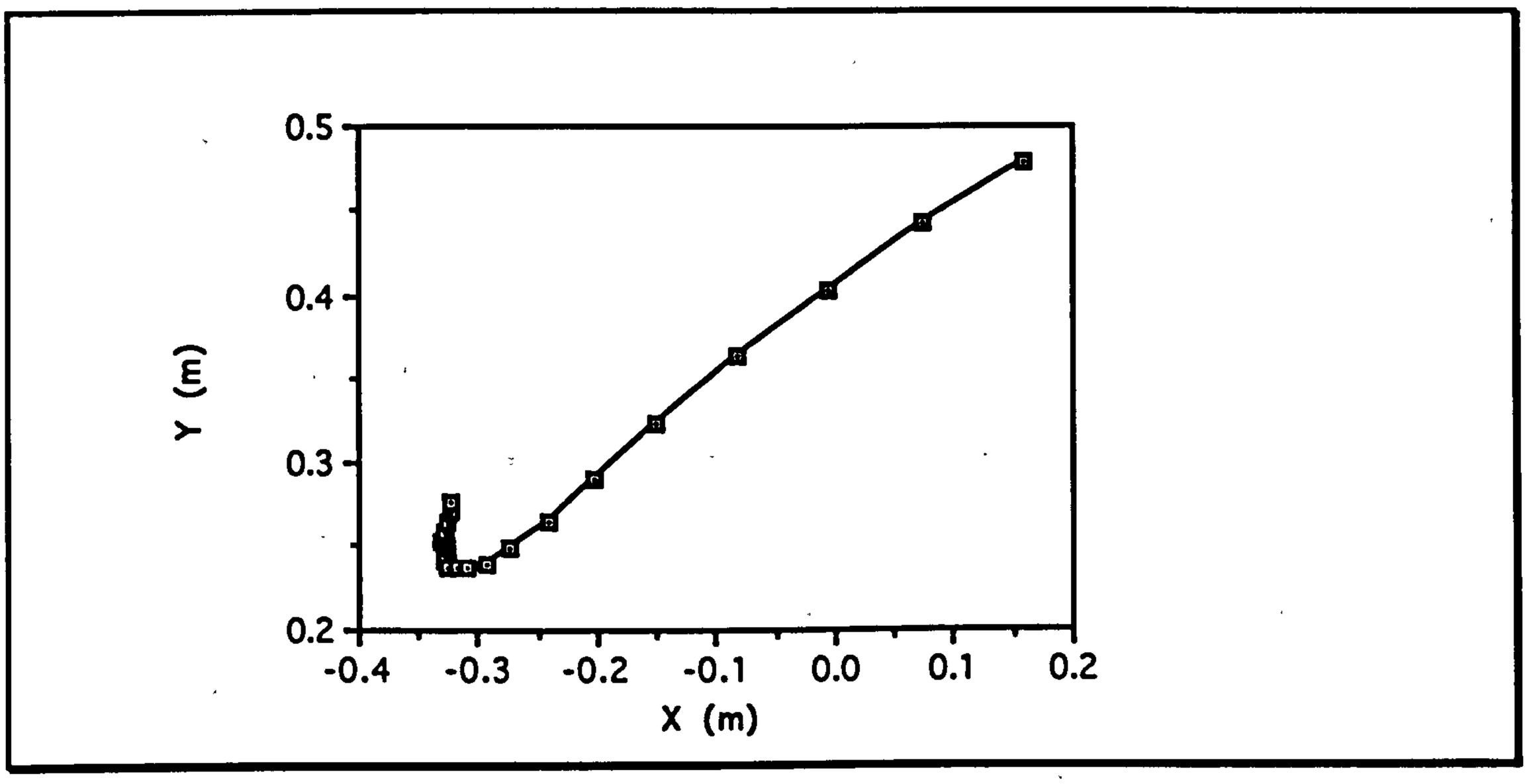
P_{cm}	is the overall position of the centre of mass
m_t	is the total mass of the animal
P_i	is the position of the i^{th} segment centre of mass
m_i	is the mass of the i^{th} segment

Trajectory

Once the overall centre of mass has been calculated for each frame of the film, then the trajectory of the animal (the path followed by the centre of mass) can be produced. This can be plotted out and the initial gradient

calculated, giving the takeoff angle for the animal. This is an important parameter, being one which the animal can alter when it wants to move more rapidly at the expense of energy efficiency. In practice, the takeoff angle was calculated by fitting a straight line to the centre of mass position between the start of the leap (position of maximum flexion) to toe off and calculating the gradient of this line. In addition, all trajectories were plotted to check that the straight line assumption was reasonable.

The following graph shows an example trajectory:



Graph of the trajectory of *Lemur catta* leaping 2.22 m. The origin position is arbitrary, and the last two points are after toe off.

Segment Mass Properties

In order to perform an inverse dynamic analysis of a link segment model, the mass properties of the segments are needed. The actual parameters needed are the length and mass of the segment; the relative position of the centre of mass along the segment; and the moment of inertia about the centre of mass. The moment of inertia is a measure of the distribution of the mass away from the centre of mass: it is the rotational analogue of mass.

These values can be measured directly but it is a time consuming process, and moreover, involves killing of the measured animal. It has also been shown that these values vary very substantially from individual to individual, especially for moments of inertia, which can differ by a factor of two or more even when body mass differences have been taken into consideration (Smith 1987). In humans, these values can be obtained by a combination of published values and allometric scaling tables, or from volume measurements obtained by immersion (Li 1991). However, there is an insufficient number of studies on prosimians to produce empirically derived scaling tables and immersion techniques are likely to be much less accurate for small and hairy animals. Since I was able to measure only the total mass of my experimental animals, I decided to use a mixed geometrical approach to estimate segment properties.

The raw data for the distribution of the body mass among the individual segments in the model was obtained from the literature (Smith 1987, Wells and DeMenthon 1987). This gives data for *Galago senegalensis* and *Eulemur fulvus*. These animals are very similar in shape to *Galago moholi* and *Lemur catta* respectively so can be used instead. However, for the remaining animals in my study, there are no really good analogues.

Galago garnettii has been modelled using the *Galago senegalensis* shape in other places, (Günther 1989) so I did the same in my study. Since I had no other option, I used the *Eulemur fulvus* body shape for the three dwarf lemurs. Because of the large difference in mass between the two galagos and among the four lemurs, I would not expect these approximations to be particularly good. However, until more data on mass distribution is available, it is probably the best that can be managed.

For most segments, I was able to use the relevant mass fraction from the literature directly and calculate the mass from the measured total mass of the animal. However, the choice of segments in some parts of the body were not the ones I wished to use. For example, I needed head and torso separately, and others have lumped head and upper torso together. For this case, I measured the volumes of the head and torso from x-ray photographs and estimated the mass fractions accordingly.

Again for segment centres of mass, I took the values directly from the literature. Because of my re-division of the torso and head segments, I used a nominal value of a half. This is relatively close to the values calculated for the other segmental distributions as can be seen in the diagrams at the end of this section.

The lengths of the segments were calculated directly from the kinematic analysis data for the animals.

To calculate the moments of inertia for each segment, a purely geometric approach was used. Each segment of the body was treated as a conic section with appropriate dimensions to fit the mass, length and centre of mass position criteria.

The volume of each conic section was calculated from its mass and a mean figure for body density (Apkarian et al. 1989).

(1) $$V = \frac{m}{\rho}$$

Where:

V	is the volume
m	is the mass
ρ	is the density

The volume of a conic section is given by:

(2) $$V = \frac{1}{3}\pi L(R^2 + Rr + r^2)$$

Where:

L	is the length of the segment
R	is the proximal radius
r	is the distal radius

The centre of mass of a conic segment is given by:

(3) $$x = \frac{L(1 + 2\mu + 3\mu^2)}{4(1 + \mu + \mu^2)}$$

Where:

x	is the horizontal position of the centre of mass

And where:

(4) $$\mu = \frac{r}{R}$$

So, for a given centre of mass, μ can be calculated[25] from by rearranging equation (3):

$$\mu = \frac{2 - 4x - \sqrt{-48x^2 + 48x - 8}}{8x - 6} \quad (6)$$

And then the moment of inertia about the centre of mass is given by:

$$I = m\left(\frac{AV}{L} + BL^2\right) \quad (7)$$

Where:

I is the moment of inertia

And:

$$A = \frac{9}{20\pi}\left(\frac{1 + \mu + \mu^2 + \mu^3 + \mu^4}{\sigma^2}\right) \quad (8)$$

$$B = \frac{3}{80}\left(\frac{1 + 4\mu + 10\mu^2 + 4\mu^3 + \mu^4}{\sigma^2}\right) \quad (9)$$

$$\sigma = 1 + \mu + \mu^2 \quad (10)$$

If necessary the actual values for R and r can be obtained by rearranging equations (2) and (4):

$$R = \sqrt{\frac{3V}{\pi L(1 + \mu + \mu^2)}} \quad (11)$$

The following table shows the values calculated for each of the study animals and the segments chosen.

[25] μ, A, B and σ are used simply as convenient intermediate values in the calculation.

		M. murinus	*L. catta*	*C. major*	*M. coquereli*	*G. garnettii*	*G. moholi*
Lower arm	**Mass(kg)**	$3.13\ 10^{-3}$	$1.35\ 10^{-1}$	$1.70\ 10^{-2}$	$1.75\ 10^{-2}$	$5.63\ 10^{-2}$	$9.24\ 10^{-3}$
	CM	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$
	MOI (kg.m2)	$2.35\ 10^{-7}$	$1.27\ 10^{-4}$	$2.70\ 10^{-6}$	$3.69\ 10^{-6}$	$3.60\ 10^{-5}$	$1.22\ 10^{-6}$
Upper arm	**Mass (kg)**	$2.88\ 10^{-3}$	$1.24\ 10^{-1}$	$1.56\ 10^{-2}$	$1.61\ 10^{-2}$	$5.40\ 10^{-2}$	$6.72\ 10^{-3}$
	CM	$5.18\ 10^{-1}$	$5.18\ 10^{-1}$	$5.18\ 10^{-1}$	$5.18\ 10^{-1}$	$4.66\ 10^{-1}$	$4.66\ 10^{-1}$
	MOI (kg.m2)	$1.77\ 10^{-7}$	$1.15\ 10^{-4}$	$2.46\ 10^{-6}$	$3.09\ 10^{-6}$	$2.37\ 10^{-5}$	$8.35\ 10^{-7}$
Fore-foot	**Mass (kg)**	$8.75\ 10^{-4}$	$3.78\ 10^{-2}$	$4.76\ 10^{-3}$	$4.90\ 10^{-3}$	$2.25\ 10^{-2}$	$4.20\ 10^{-3}$
	CM	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$
	MOI (kg.m2)	$1.26\ 10^{-8}$	$5.14\ 10^{-6}$	$1.96\ 10^{-7}$	$2.18\ 10^{-7}$	$1.97\ 10^{-6}$	$2.07\ 10^{-7}$
Hind-foot	**Mass (kg)**	$8.75\ 10^{-4}$	$3.78\ 10^{-2}$	$4.76\ 10^{-3}$	$4.90\ 10^{-3}$	$1.13\ 10^{-2}$	$2.52\ 10^{-3}$
	CM	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$
	MOI (kg.m2)	$1.32\ 10^{-8}$	$4.94\ 10^{-6}$	$2.17\ 10^{-7}$	$2.26\ 10^{-7}$	$1.41\ 10^{-6}$	$1.35\ 10^{-7}$
Calf	**Mass (kg)**	$3.38\ 10^{-3}$	$1.46\ 10^{-1}$	$1.83\ 10^{-2}$	$1.89\ 10^{-2}$	$6.08\ 10^{-2}$	$1.39\ 10^{-2}$
	CM	$4.01\ 10^{-1}$	$4.01\ 10^{-1}$	$4.01\ 10^{-1}$	$4.01\ 10^{-1}$	$6.08\ 10^{-1}$	$6.08\ 10^{-1}$
	MOI (kg.m2)	$3.57\ 10^{-7}$	$1.61\ 10^{-4}$	$6.86\ 10^{-6}$	$5.39\ 10^{-6}$	$4.08\ 10^{-5}$	$4.15\ 10^{-6}$
Thigh	**Mass (kg)**	$1.03\ 10^{-2}$	$4.43\ 10^{-1}$	$5.57\ 10^{-2}$	$5.74\ 10^{-2}$	$1.37\ 10^{-1}$	$4.20\ 10^{-2}$
	CM	$4.47\ 10^{-1}$	$4.47\ 10^{-1}$	$4.47\ 10^{-1}$	$4.47\ 10^{-1}$	$5.61\ 10^{-1}$	$5.61\ 10^{-1}$
	MOI (kg.m2)	$1.26\ 10^{-6}$	$7.39\ 10^{-4}$	$2.46\ 10^{-5}$	$2.83\ 10^{-5}$	$1.48\ 10^{-4}$	$1.73\ 10^{-5}$
Head	**Mass (kg)**	$6.38\ 10^{-3}$	$2.75\ 10^{-1}$	$3.46\ 10^{-2}$	$3.57\ 10^{-2}$	$1.20\ 10^{-1}$	$2.08\ 10^{-2}$
	CM	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$
	MOI (kg.m2)	$7.30\ 10^{-7}$	$2.88\ 10^{-4}$	$1.16\ 10^{-5}$	$1.17\ 10^{-5}$	$7.92\ 10^{-5}$	$3.37\ 10^{-6}$
Torso	**Mass (kg)**	$3.26\ 10^{-2}$	$1.41\ 10^{+0}$	$1.77\ 10^{-1}$	$1.82\ 10^{-1}$	$6.15\ 10^{-1}$	$1.05\ 10^{-1}$
	CM	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$
	MOI (kg.m2)	$1.82\ 10^{-5}$	$9.58\ 10^{-3}$	$3.05\ 10^{-4}$	$3.06\ 10^{-4}$	$1.94\ 10^{-3}$	$1.12\ 10^{-4}$
Tail	**Mass (kg)**	$2.19\ 10^{-3}$	$9.45\ 10^{-2}$	$1.19\ 10^{-2}$	$1.23\ 10^{-2}$	$4.84\ 10^{-2}$	$5.25\ 10^{-3}$
	CM	$3.78\ 10^{-1}$	$3.78\ 10^{-1}$	$3.78\ 10^{-1}$	$3.78\ 10^{-1}$	$3.78\ 10^{-1}$	$3.78\ 10^{-1}$
	MOI (kg.m2)	$2.33\ 10^{-6}$	$1.13\ 10^{-3}$	$4.57\ 10^{-5}$	$5.39\ 10^{-5}$	$1.11\ 10^{-4}$	$1.06\ 10^{-5}$

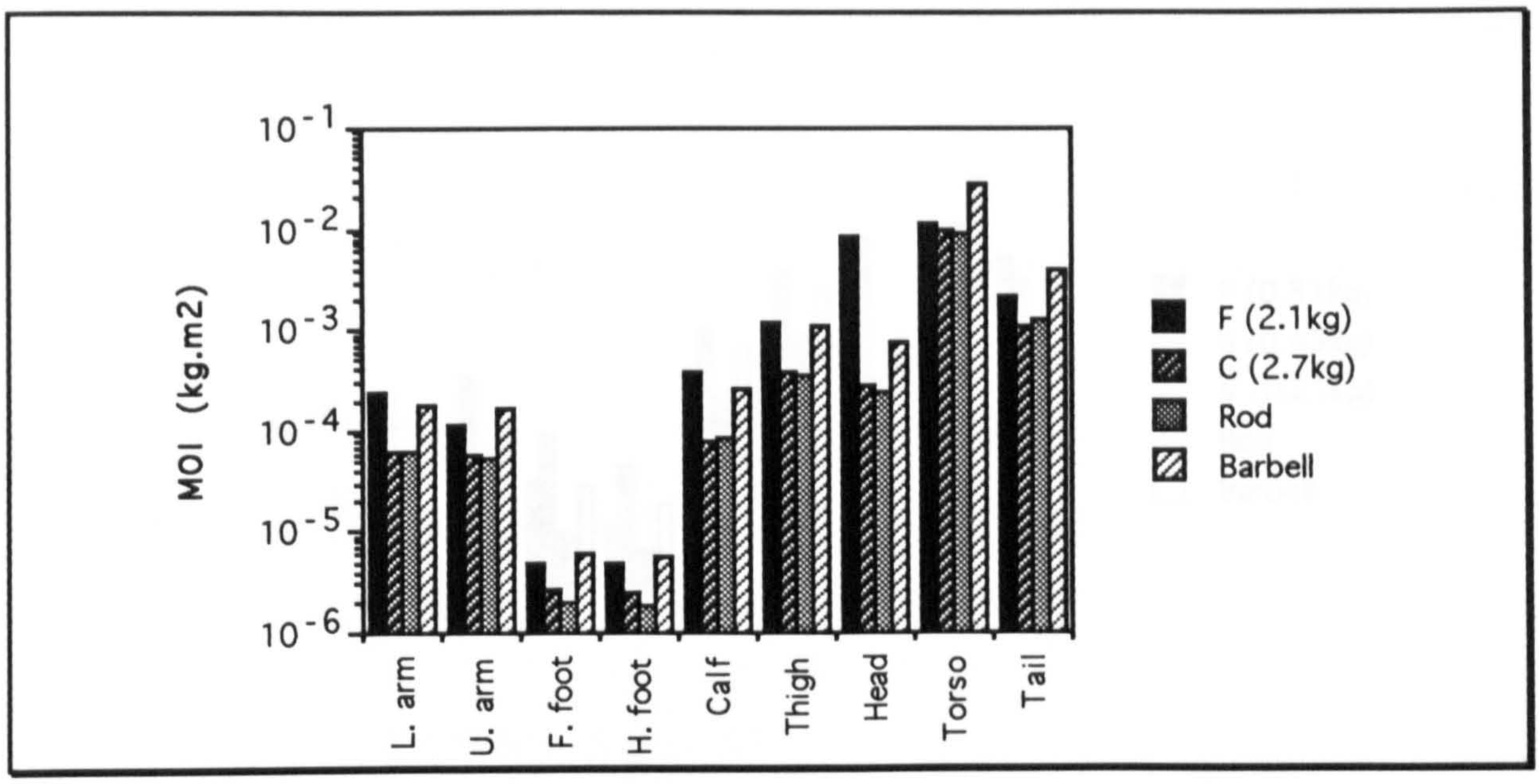
In the table, mass refers to the total mass of the segment. CM is the position of the centres of mass as the relative distance from the distal end for limbs and the relative distance caudally for the axial segments. MOI is the moment of inertia about the centre of mass. The values for limbs are the sums of the values for the left and right hand sides of the animals as used in the computer program.

To check that the calculated values of the moments of inertia are reasonable, the following graphs shows published values for *Galago senegalensis* and *Eulemur fulvus* and my calculated values. Also shown are the probable limits for the moment of inertia. The minimum is obtained by considering the limb as a uniform rod rotating about the middle: (Kleppner and Kolenkow 1973)

(12) $$I = \frac{mL^2}{12}$$

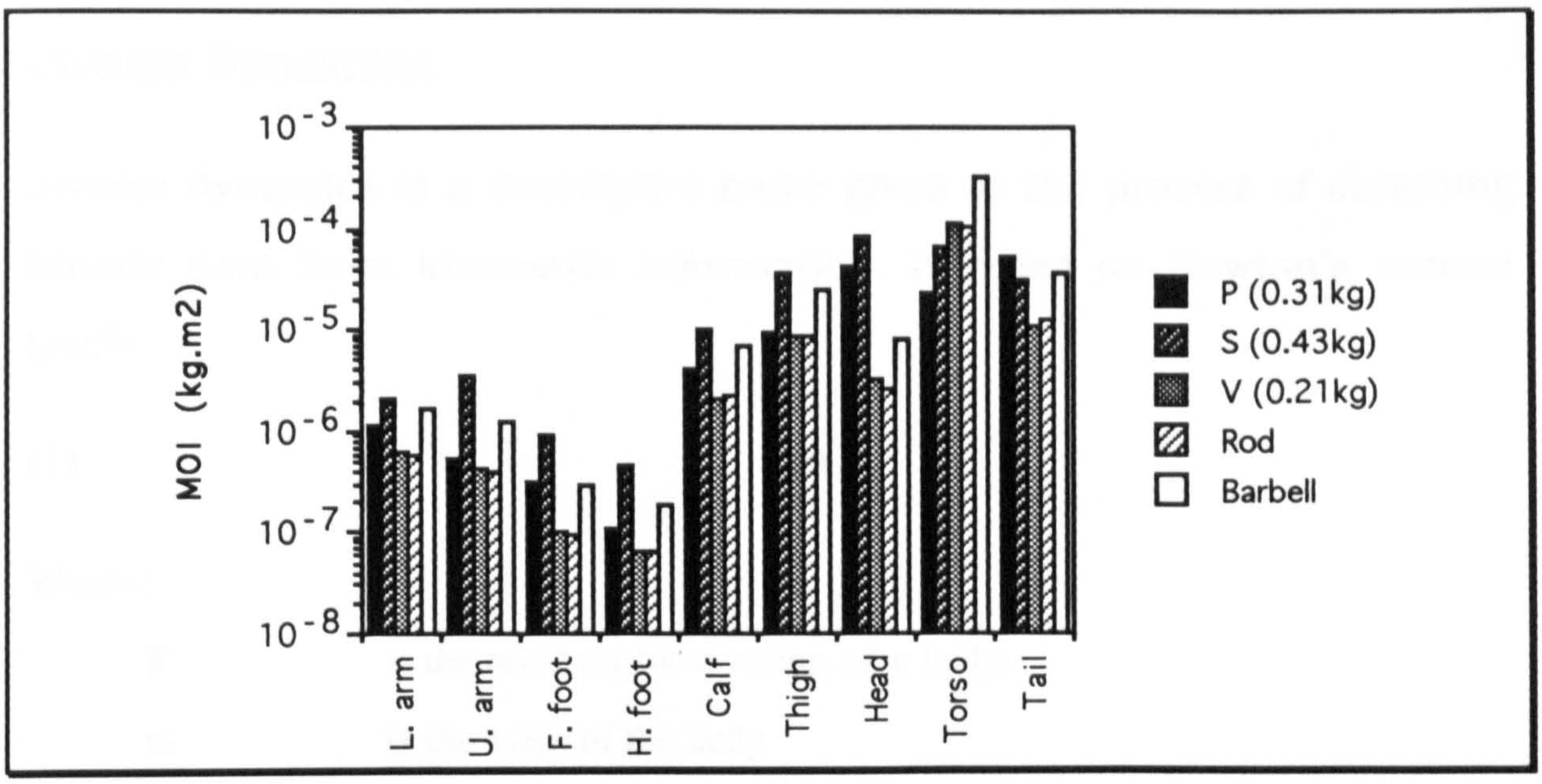
and the maximum by considering it as a barbell, with the mass concentrated at the ends (this is a very excessive assumption):

(13) $$I = \frac{mL^2}{2}$$



This graph shows the moments of inertia of the limb segments of the geometrically derived *Lemur catta* (C) compared with those experimentally measured from *Eulemur fulvus* (F) (Wells and DeMenthon 1987)[26]. For comparison, the values for the rod and barbell models are given for the *Lemur catta* mass. The values for head, torso and tail use different segmental division. F includes part of the torso in the head segment, and the tail in the torso measurement. The indicated tail measurement has been extrapolated.

[26]The results presented by Wells and DeMenthon are sufficiently detailed to be checked geometrically. In fact, their measured moments of inertia are higher even than those that would be predicted using the barbell model. This is extremely unlikely, and does suggest that there was some error in their calculations.



This graph shows the moments of inertia of the limb segments of the geometrically derived *Galago moholi* (V) compared with those experimentally measured (P and S) (Smith 1987). For comparison, the values for the rod and barbell models are given for V's mass. The values for head and torso use different segmental division. Both P and S include part of the torso in the head segment.

Inverse Dynamics

Inverse dynamics is a descriptive name given to the process of obtaining kinetic data from kinematic information. It relies on Newton's second law:[27]

(1) $$\mathbf{F} = m\mathbf{a}$$

Where:

F	is the resultant force acting on a body.
m	is the mass of the body
a	is the acceleration of the body

And the angular equivalent for when the force is not acting through the centre of mass of the body

(2) $$T = I\alpha$$

Where:

T	is the resultant torque acting on the body
I	is the moment of inertia of the body
α	is the angular acceleration of the body

The approach used for inverse dynamic analysis of a complex, linked structure such as the limbs of an animal connected by joints, is to treat each rigid segment in isolation and to apply equations (1) and (2). Forces and linear accelerations can be split into their components in the principal axes: X, Y (and Z, if working in three dimensions). Torques,

---

[27]Newton's laws of motion were first published in 1687 in his treatise "Principia mathematica". Here, they are largely descriptive, and they were not translated into a more precise mathematical form until Mach published "The science of mathematics" in 1883. Before Newton, Aristotelian mechanics reigned supreme, with its basic tenet that a force was required to keep a body in motion. Newton, and before him, Galileo, postulated that a force was only required to *change* the movement of an object. (Kleppner and Kolenkow 1973)

moments of inertia and angular accelerations can be split depending on the plane of rotation. These planes are defined mathematically as X=0, Y=0 and Z=0. They correspond to rotation about the X, Y and Z axes respectively. Thus, only the Z=0 plane is involved in two dimensional analysis.

This can be expressed mathematically as: (see Winter 1990)

(3) $$\sum_{i=1}^{n} F_i = m \sum_{i=1}^{n} a_i$$

(4) $$\sum_{i=1}^{n} T_i = I \sum_{i=1}^{n} \alpha_i$$

Where:

n is the number of forces or torques acting on the segment

Linear forces produce torques, but torques do not effect the linear properties, so the non-rotational problem needs to be solved first. This is simply a matter of knowing the mass of the free body and its acceleration (the right hand side of equation 1) and then summing up all the forces acting upon it. This is done separately in the X and Y direction. In the Y analysis, the additional force of gravity also acts.

The following diagram shows the forces present on the isolated limb segment:

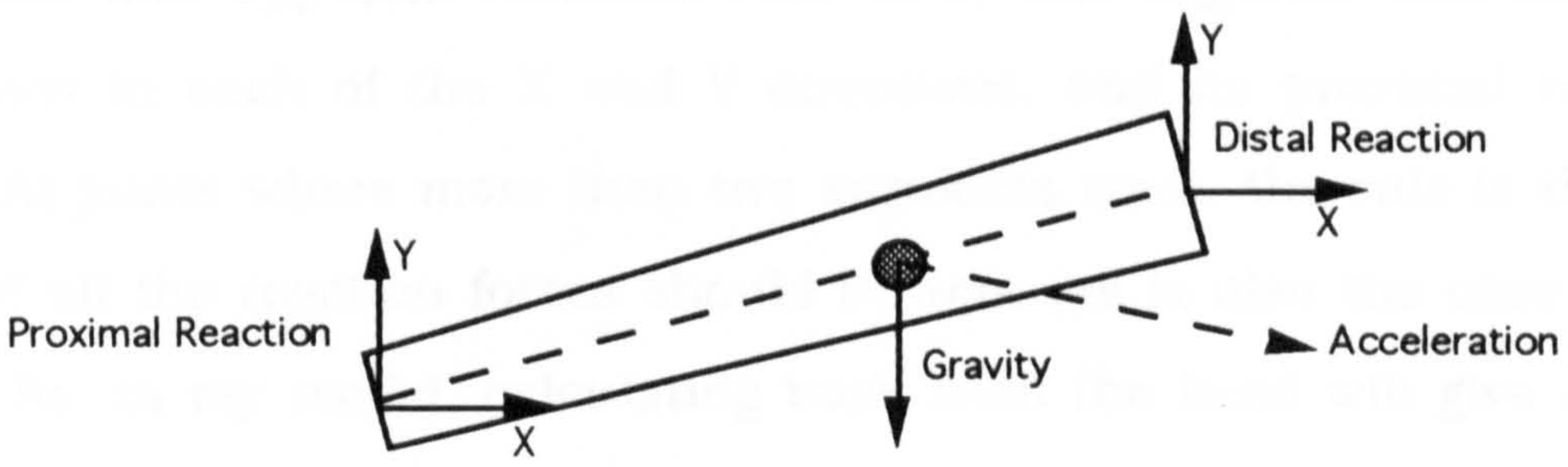


Diagram showing the forces acting on an isolated segment in a link segment model. The reaction forces can be split into their components in the X and Y directions. Gravity only acts in the negative Y direction.

So, the equation (3) becomes:

(5) $$F_{x1} + F_{x2} = ma_x$$

(6) $$F_{y1} + F_{y2} - mg = ma_y$$

Where:

F_{x1}	is the x component of the proximal reaction force
F_{x2}	is the x component of the distal reaction force
F_{y1}	is the y component of the proximal reaction force
F_{y2}	is the y component of the distal reaction force
g	is the acceleration due to gravity.

The calculation of the whole model proceeds from segments that are unattached at one end, such as the forearm (in this model). At these segments, the distal reaction force is zero. Then, there is only one unknown in each of the equations for the separate components. This is

the proximal reaction force which can now be easily calculated. This proximal reaction force becomes the negative of the distal reaction force acting on the next segment due to Newton's second law: each action has an equal and opposite reaction. And now, this segment has only one unknown in each of the X and Y directions, and its proximal reaction force. At joints where more than two segments meet, the rule is that the total of all the reaction forces should be zero (as is also the case at any joint). So, in my model, calculating back from the head will give one set of reaction forces at the neck. Calculating up the arm will give another. The negative of the sum of these two forces will be the reaction force acting on the cranial end of the torso. Calculating all the way down will give the reaction force acting between the tip of the toe and the ground.

Once the linear forces are known, the rotational parameters can then be calculated. The rotational torques present in the linked segment model are shown in the following diagram:

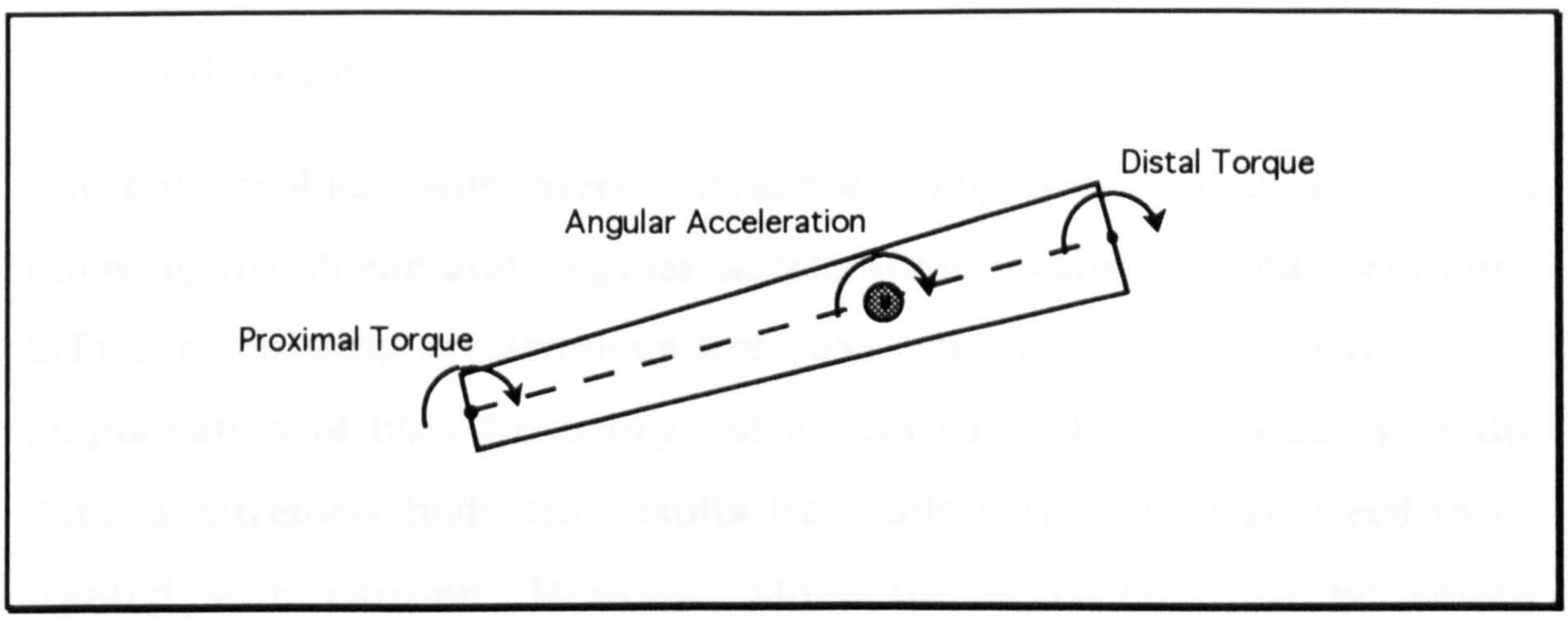


Diagram showing the torques acting on an isolated segment in a link segment model. In addition, the reaction forces at the proximal and distal ends also apply a torque (though gravity does not).

So, equation (4) becomes:

(7) $$F_{x1}D_{y1} + F_{y1}D_{x1} + F_{x2}D_{y2} + F_{y2}D_{x2} + T_1 + T_2 = I\alpha$$

Where:

D_{x1}	is the x distance from the proximal joint to the centre of mass
D_{x2}	is the x distance from the distal joint to the centre of mass
D_{y1}	is the y distance from the proximal joint to the centre of mass
D_{y2}	is the y distance from the distal joint to the centre of mass
T_1	is the torque about the proximal joint
T_2	is the torque about the distal joint

The progression of calculation of the torques is similar to that for the reaction forces. From a free end, the distal torque is zero. All the linear forces are known, as are the distances of the joints from the centre of mass in both the X and Y directions. Thus, the proximal torque is the only unknown and can be calculated. The negative of this becomes the distal torque at the next segment along. When more than two segments meet at a joint, the total torque is again, zero, and there will be only one unknown torque.

The only problem with inverse dynamic analysis is that it depends on knowing the linear and angular accelerations accurately. As mentioned before, calculating accelerations from positional data leads to a very large amplification of high frequency noise, and so, unless the quality of the data is extremely high, the results from this sort of analysis need to be treated with caution. However, since the equations can be solved analytically, producing information about forces and torques from artificially generated results (from a predictive model for example) is extremely quick and reliable.

The reliability could be checked experimentally by comparing the external forces calculated by the analysis with forces measured with a force plate. This option was not available in the present study.

Results Section

Leaping Trajectory

Theory

When choosing how to leap a gap of a particular size, an animal has two physical parameters that it can alter: its takeoff velocity and its angle of trajectory. Since leaping is a ballistic form of motion, these two parameters are related by the following equation for leaping with no height change[28]:

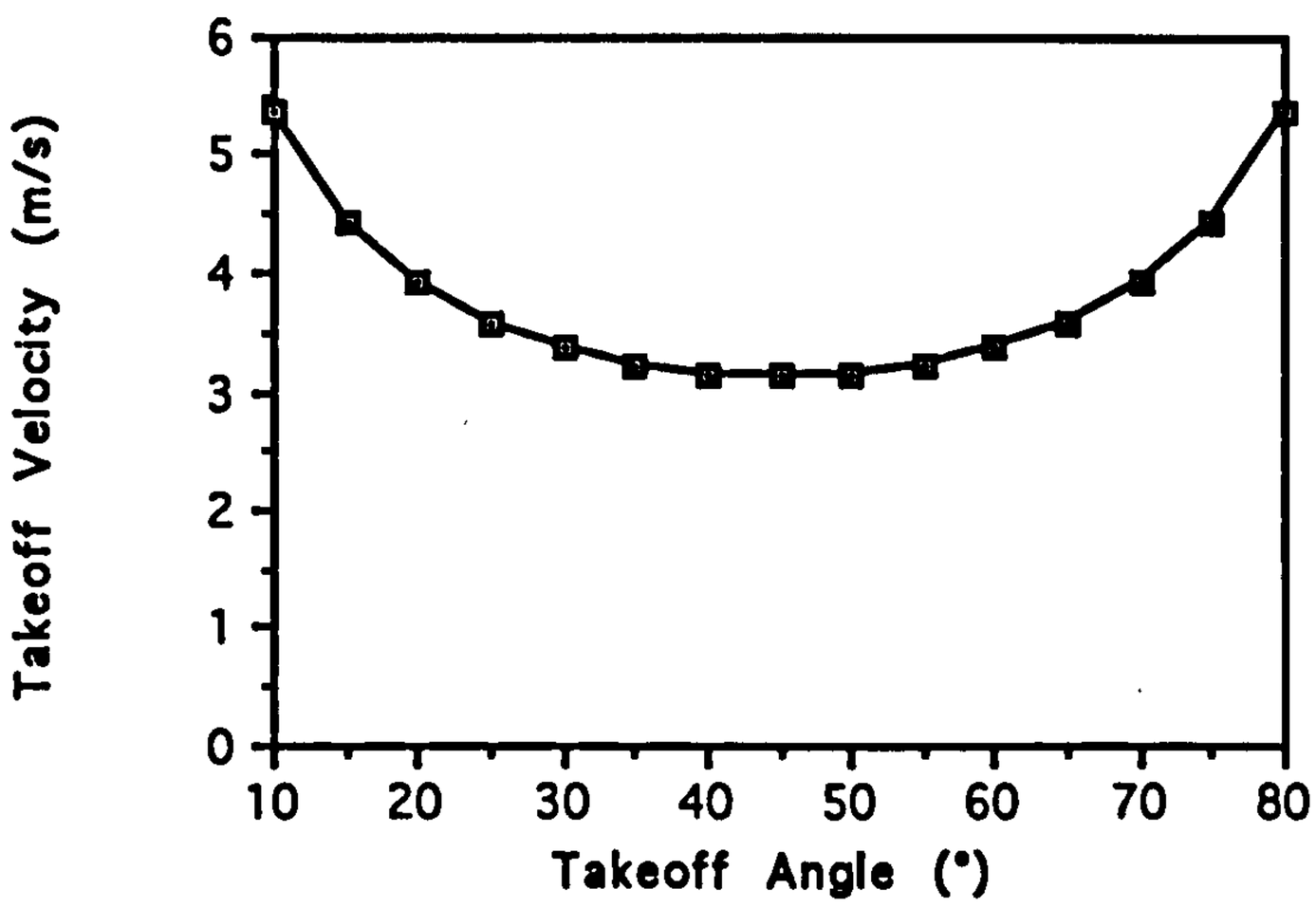
(1) $$v_{to} = \sqrt{\frac{rg}{\sin 2\theta}}$$

Where:

v_{to}	is the takeoff velocity
r	is the range of the jump
g	is the acceleration due to gravity
θ	is the takeoff angle

[28]A difference in height between takeoff and landing points makes the relationships slightly more complex and alters the value of the optimal takeoff angle but otherwise does not affect any of the general properties of the interaction between takeoff velocity and angle.

Graphically this relationship is as follows:



Graph showing the theoretical relationship between takeoff velocity and trajectory for a 1 m leap.

From this, it can be seen that the animal can either choose a shallow or a steep trajectory and a high takeoff velocity, or can minimize its takeoff velocity by choosing a takeoff angle of 45°. From an ecological standpoint, it is helpful to look at the differing energy costs of these options. This cost is simply the kinetic energy of the animal at takeoff:

(2) $$E_{KE} = \frac{1}{2}mv_{to}^2$$

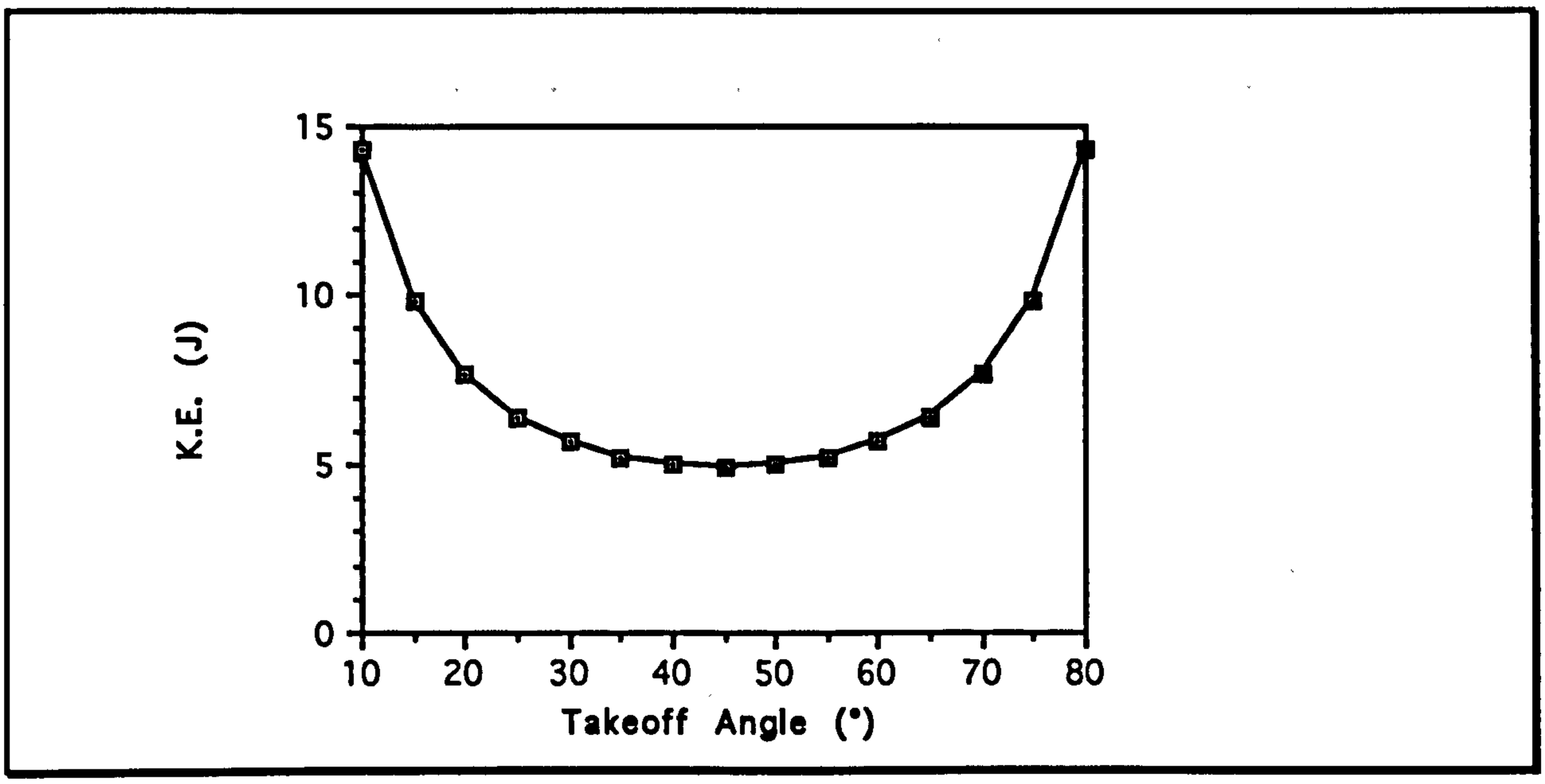
Where:

E_{KE} is the kinetic energy of the animal

So, from equations (1) and (2), the following relationship can be formulated:

(3) $$E_{KE} = \frac{mrg}{2 \sin 2\theta}$$

And again, this can be shown graphically:



Graph showing the relationship between the initial trajectory and the minimum energetic cost of a 1 m leap for a 1 kg animal.

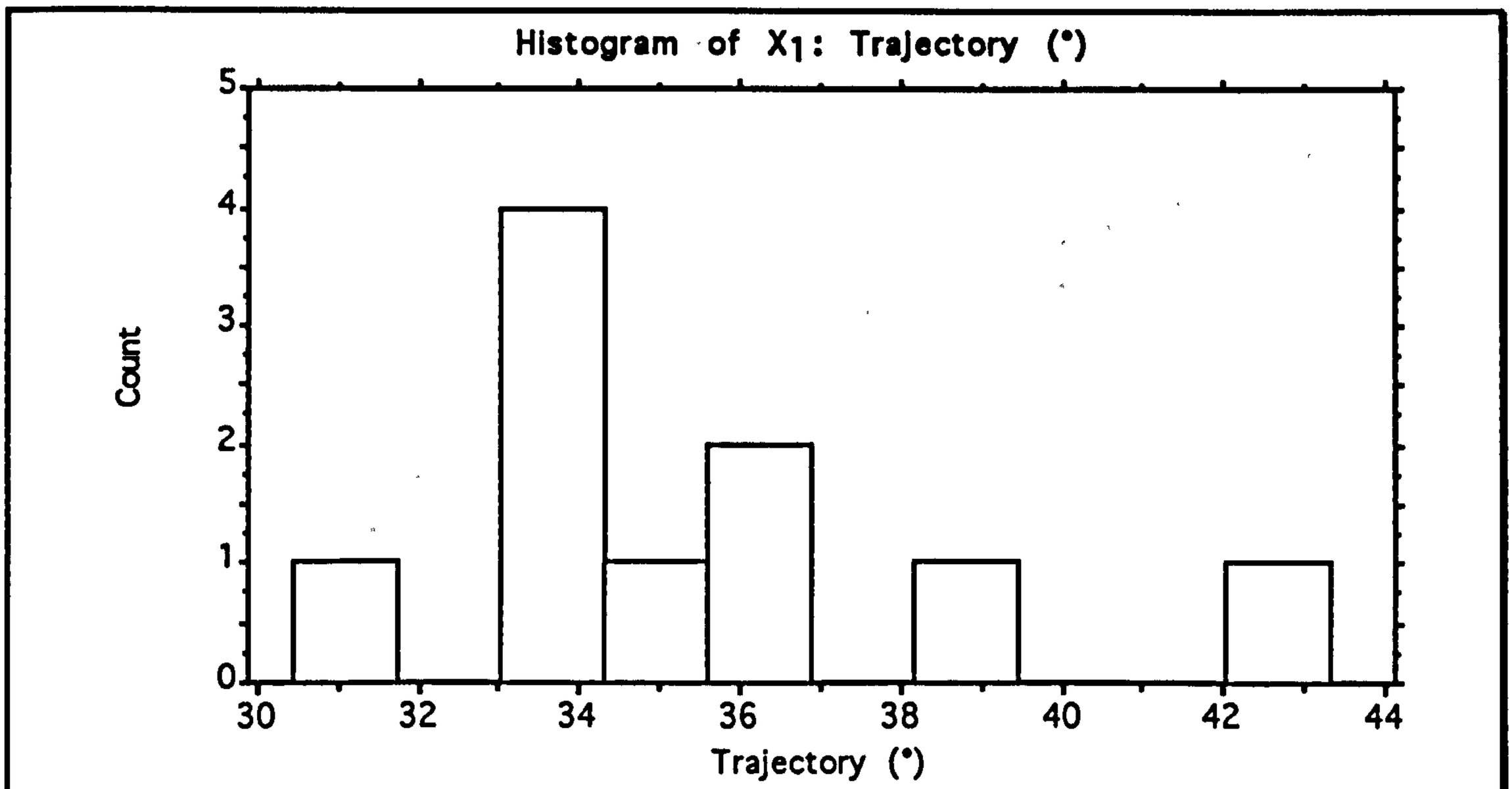
From this graph, it can be seen that the most economical takeoff angle, the one that leads to the longest jump for the least energy expenditure, is 45°, and one would expect that this is the takeoff angle that would be chosen by the animal.

Results

For each animal and for all jump distances, the leaping trajectory was calculated. For *Lemur catta*, which I only filmed leaping at a single distance, I simply calculated the 95% confidence limits of the trajectory. For the other 5 animals, the relationship between the leap distance and the trajectory was investigated by calculating the regression lines with 95% confidence limits.

The following graphs show the results obtained:

First for *Lemur catta*:

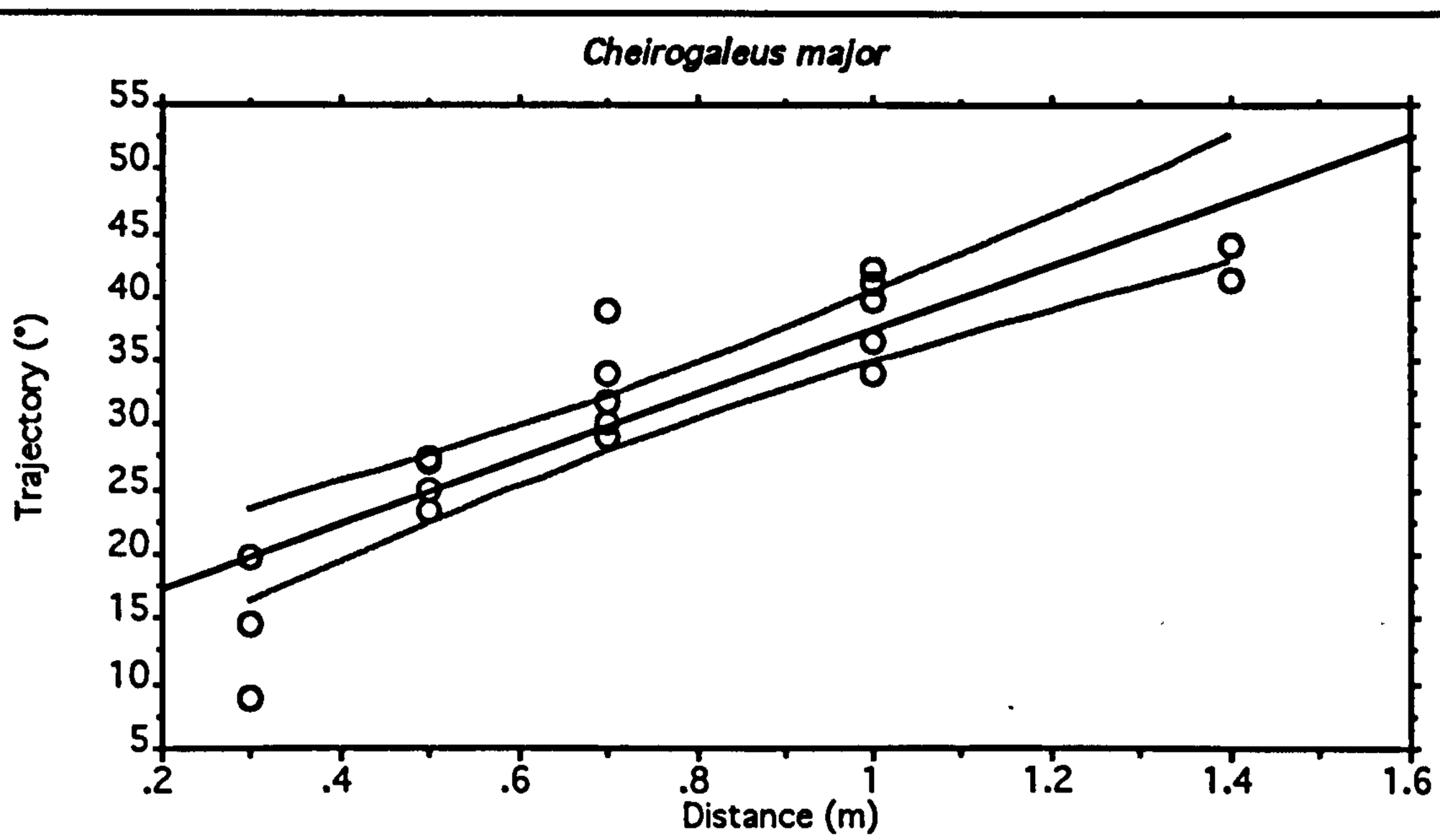


X_1: Trajectory (°)

Mean:	Std. Dev.:	Std. Error:	Variance:	Coef. Var.:	Count:
35.308	3.216	1.017	10.343	9.108	10
Minimum:	**Maximum:**	**Range:**	**Sum:**	**Sum Squared:**	**# Missing:**
30.46	42.32	11.86	353.08	12559.633	0
t 95%:	**95% Lower:**	**95% Upper:**			
2.301	33.007	37.609			

Histogram showing the distribution of trajectories observed for a set of 10 2.22 m leaps in *Lemur catta*. The mean trajectory is 35.3 ± 2.3° (95%) which is clearly less than the predicted value of 45°.

And now for the others:



Simple Regression X_1: Distance (m) Y_1: Trajectory (°)

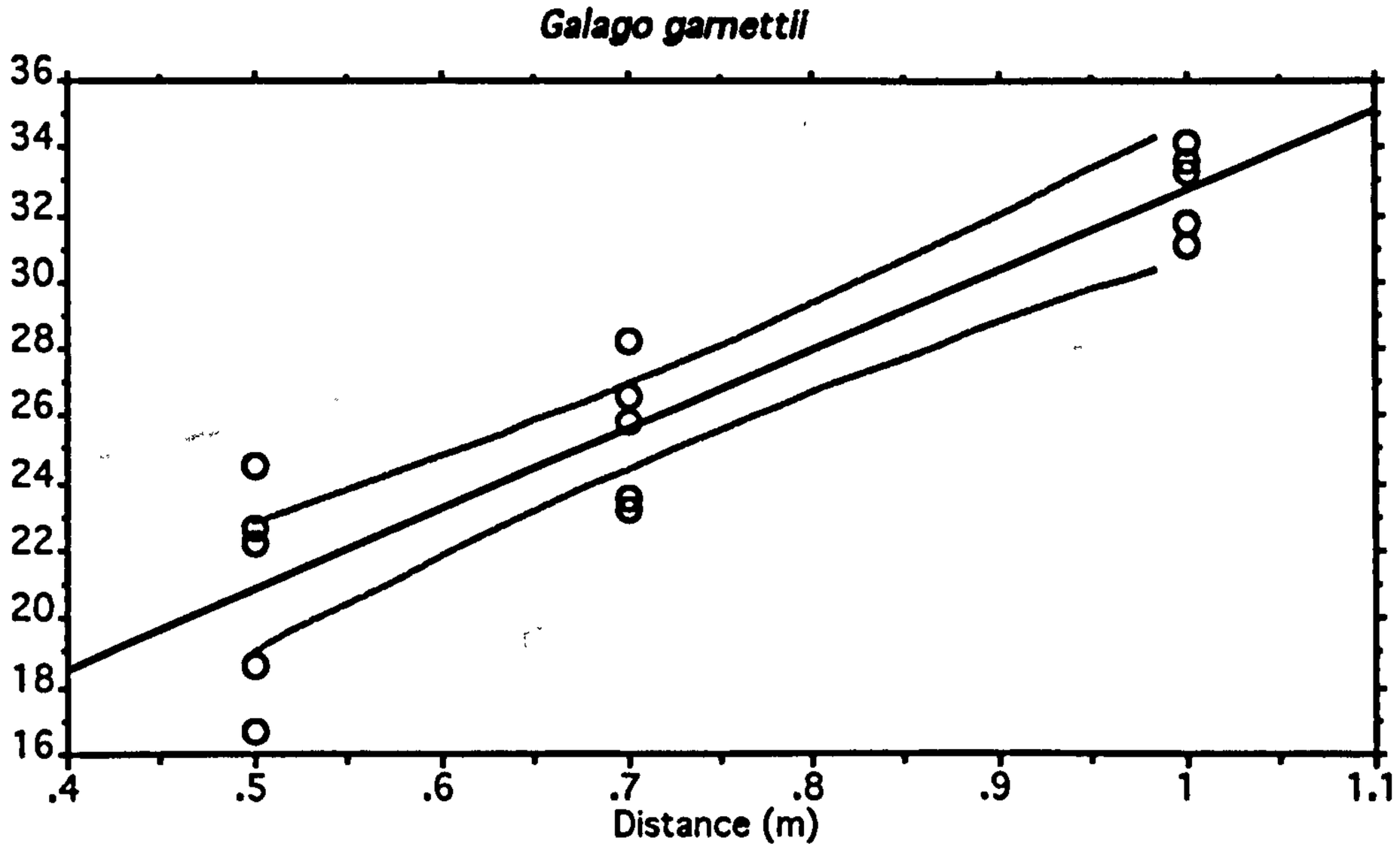
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	12.239				
SLOPE	25.255	3.153	.884	8.01	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	28.662	32.94	29.035	32.567
SLOPE	18.63	31.879	19.787	30.722

Graph showing the observed trajectories for a variety of different leaps distances in *Cheirogaleus major*. The regression line and 95% confidence limits are purely to illustrate that there is a significant trend, and are not to suggest that there is any linear relationship.



Simple Regression X_1: Distance (m) Y_1: Trajectory (°)

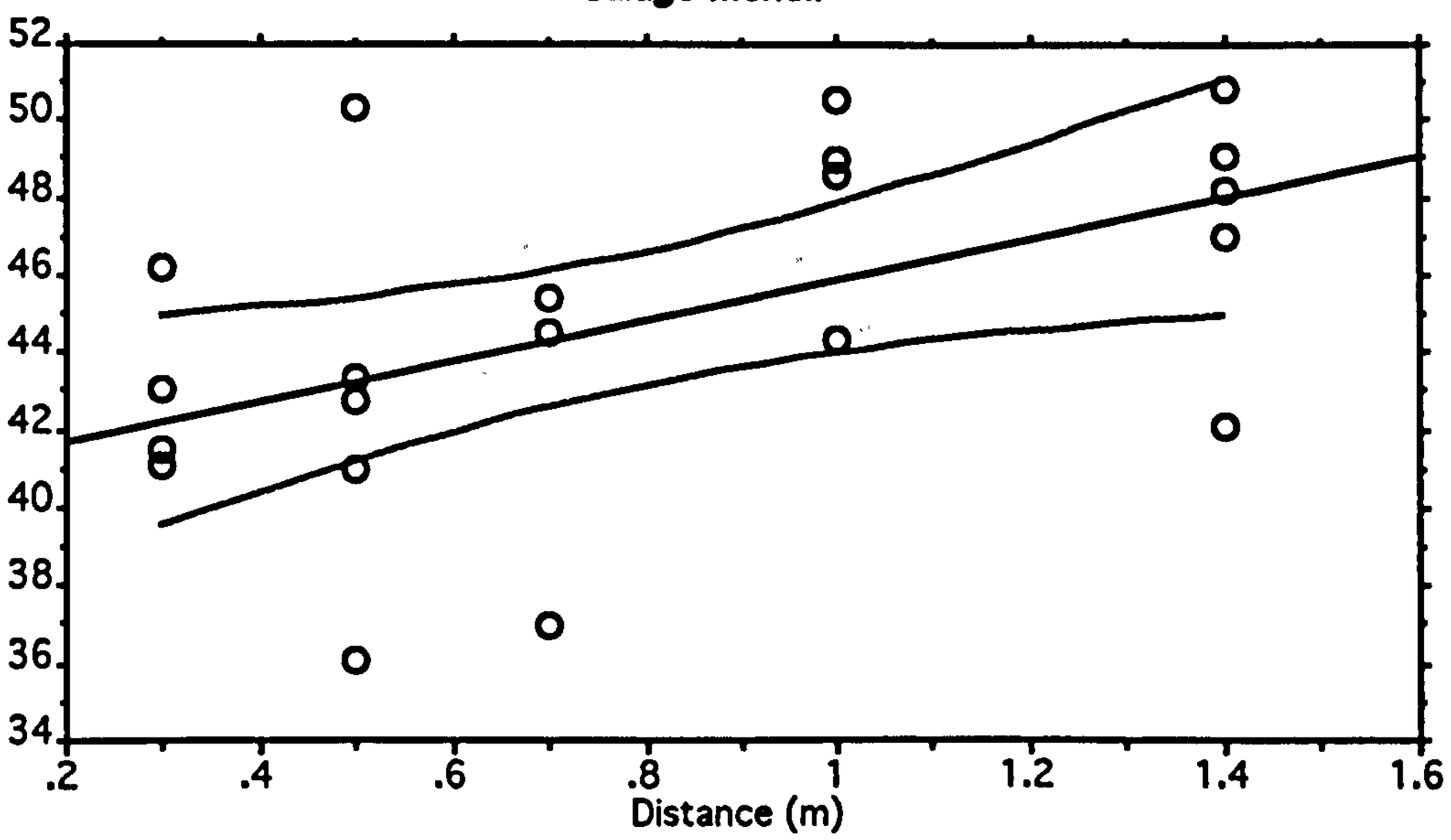
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	9.038				
SLOPE	23.686	2.832	.918	8.362	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	25.15	27.666	25.377	27.439
SLOPE	17.566	29.806	18.669	28.702

Graph showing the observed trajectories for a variety of different leaps distances in *Galago garnettii*. The regression line and 95% confidence limits are purely to illustrate that there is a significant trend, and are not to suggest that there is any linear relationship.



Simple Regression X_1: Distance (m) Y_1: Trajectory (°)

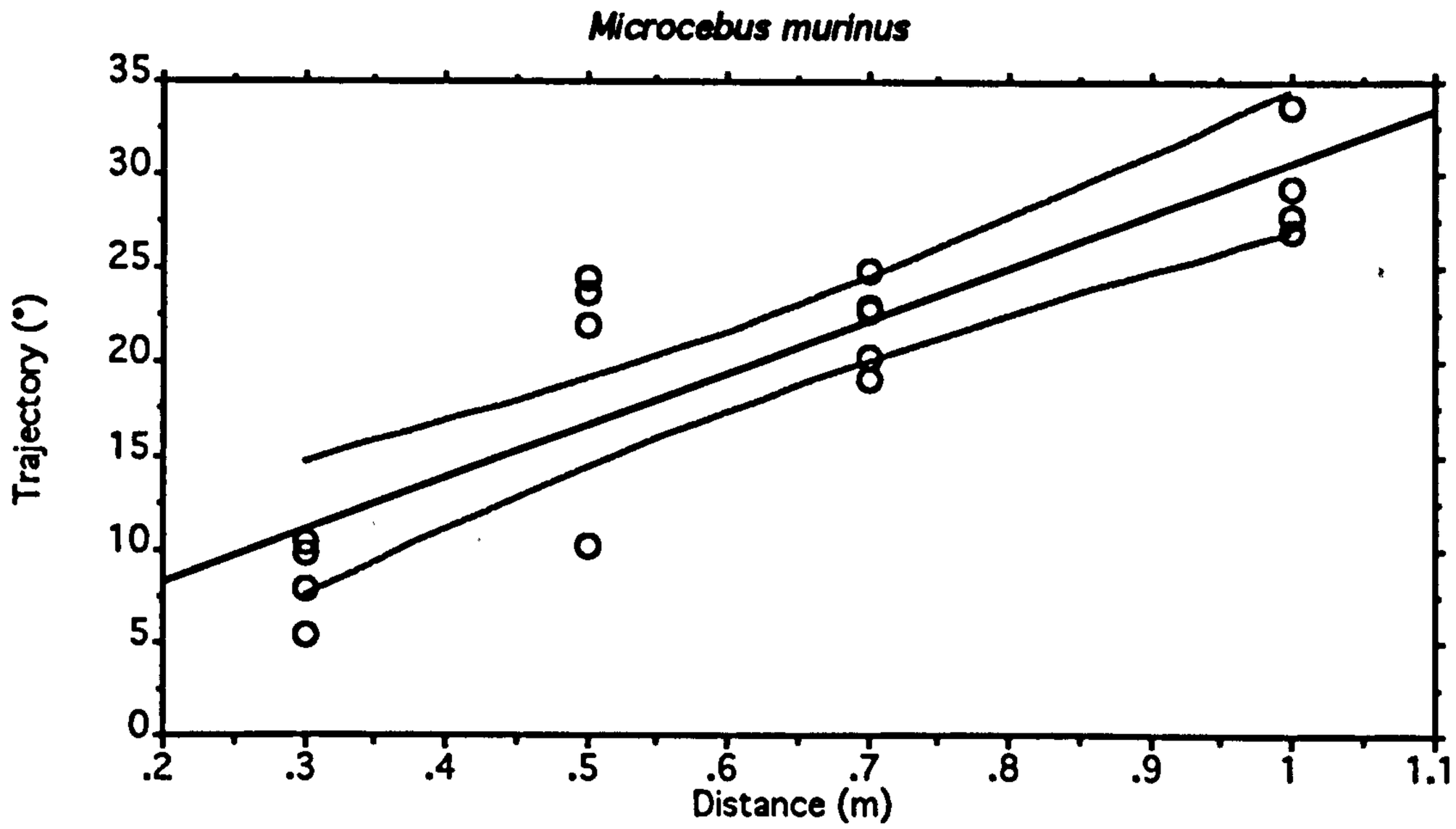
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	40.626				
SLOPE	5.26	2.027	.512	2.595	.0178

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	43.117	46.551	43.415	46.252
SLOPE	1.017	9.503	1.755	8.765

Graph showing the observed trajectories for a variety of different leaps distances in *Galago moholi*. The regression line and 95% confidence limits are purely to illustrate that there is a significant trend, and are not to suggest that there is any linear relationship. In this case the trend is noticeably less pronounced than for the others, and there are results with trajectories greater than 45°.



Simple Regression X_1: Distance (m) Y_1: Trajectory (°)

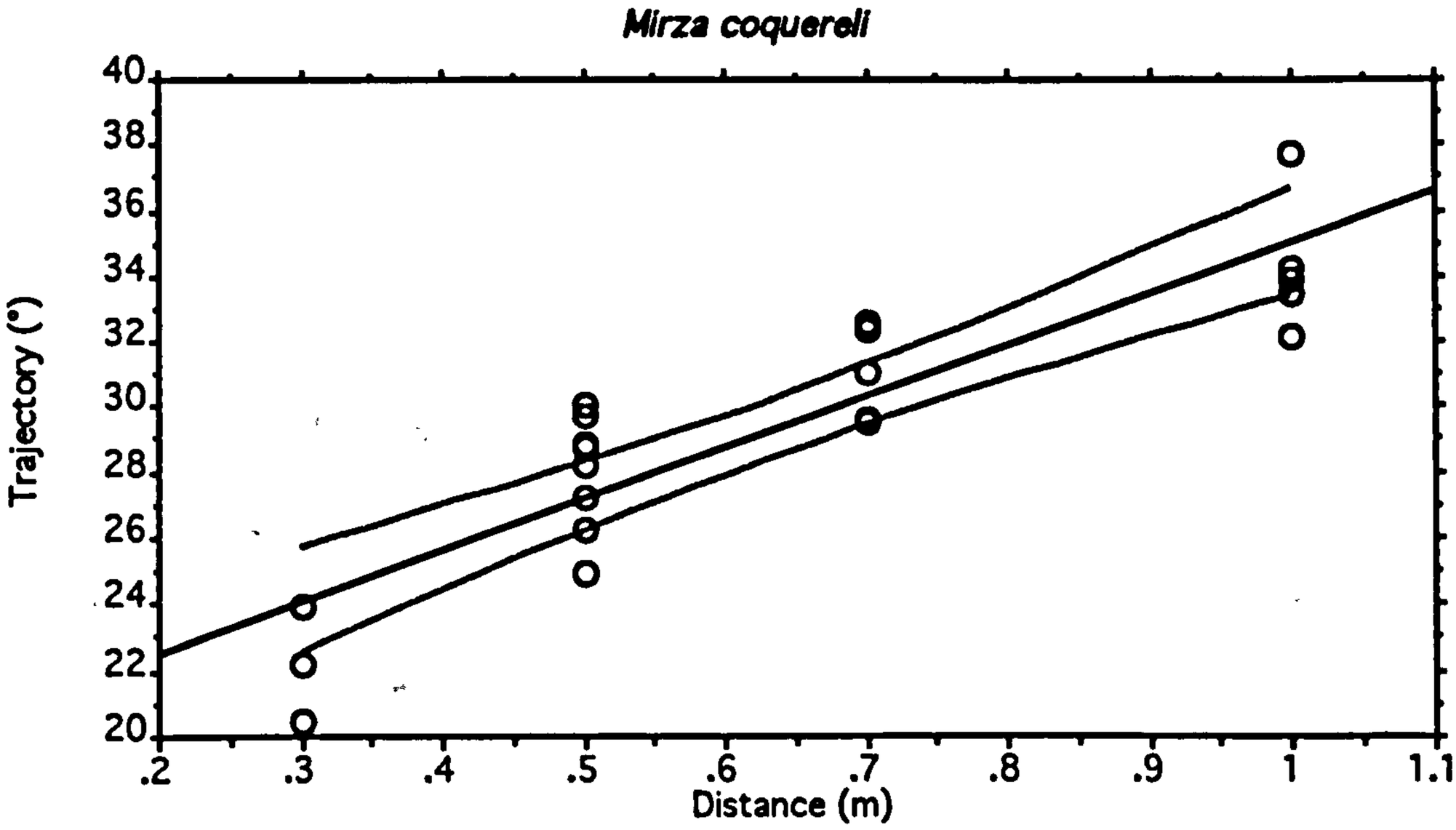
Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	2.63				
SLOPE	27.99	4.095	.863	6.835	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	18.231	22.484	18.606	22.109
SLOPE	19.308	36.673	20.84	35.141

Graph showing the observed trajectories for a variety of different leaps distances in *Microcebus murinus*. The regression line and 95% confidence limits are purely to illustrate that there is a significant trend, and are not to suggest that there is any linear relationship.



Simple Regression X_1: Distance (m) Y_1: Trajectory (°)

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Std. Value:	t-Value:	Probability:
INTERCEPT	19.429				
SLOPE	15.587	1.826	.891	8.535	.0001

Confidence Intervals Table

Parameter:	95% Lower:	95% Upper:	90% Lower:	90% Upper:
MEAN (X,Y)	28.473	30.277	28.63	30.121
SLOPE	11.764	19.41	12.429	18.746

Graph showing the observed trajectories for a variety of different leaps distances in *Mirza coquereli*. The regression line and 95% confidence limits are purely to illustrate that there is a significant trend, and are not to suggest that there is any linear relationship.

Discussion

The main thing to note about these results is that, except for *Galago moholi*, these animals do not leap at the energetically optimal angle of 45°. They leap at appreciably shallower angles. However, as the leap distance increases, they all do choose to leap at more energetically efficient angles and it would certainly be expected that they would all

have to leap at 45° for their longest leaps. Looking at the data for *Cheirogaleus major*, which is the least frequent leaper of the group, it is probable that 1.4 m is near its maximum leaping distance and so it is not surprising that it is forced to use the optimal leap trajectory.

Since the animals are not leaping at the energetically most efficient angle, then there must be other benefits associated with the shallower trajectories. One reason could be that the animals are prepared to trade-off some loss in energetic efficiency for a gain in travelling speed. The horizontal speed of a leap is given by:

(4) $$v_h = v_{to} \cos \theta$$

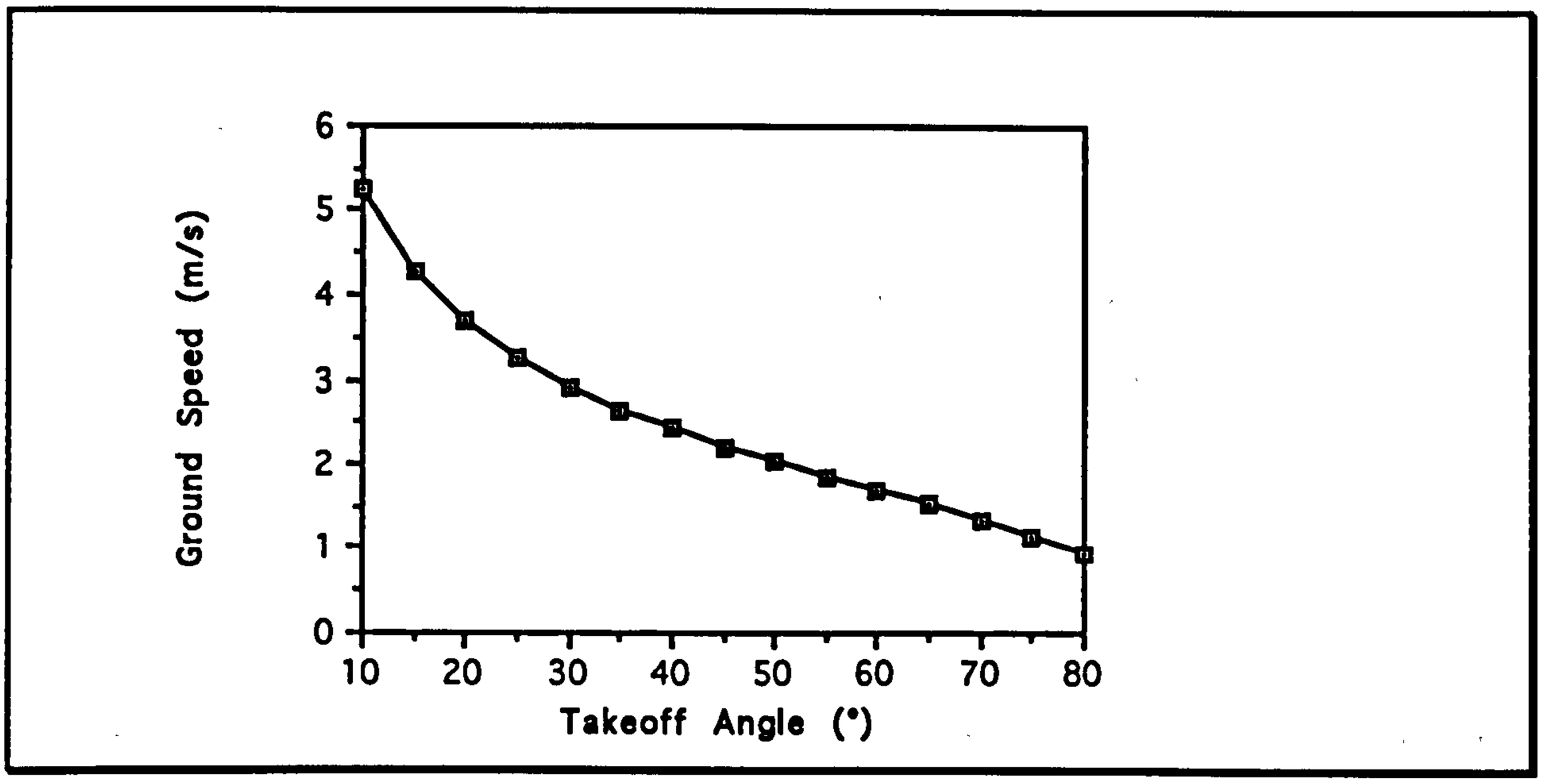
Where:

v_h is the horizontal velocity

So, from equations (1) and (4), the relationship between speed and leap angle can be calculated:

(5) $$v_h = \cos \theta \sqrt{\frac{rg}{\sin 2\theta}}$$

And shown graphically:



Graph showing the relationship between the takeoff angle and the horizontal velocity for an animal leaping 1m.

So a "time pressurized" animal has to decide on a compromise between the relatively high cost of a very quick leap and the relative slowness of an energetically efficient one. On longer leaps, the range of choice in leap angle is more restricted because the minimum takeoff velocity for the leap is a larger fraction of the maximum takeoff velocity that the animal can manage and it therefore has less scope for selecting a faster, flatter trajectory.

The importance of travel time depends very much on the ecology of the animal. If resources are widely spaced then it might be extremely important to travel quickly between them so that there would be plenty of time to exploit them once the animal has arrived. Speed is also important for escape behaviour, but since the study animals were not being chased, but were rather being induced to jump by the presence of food rewards, this is unlikely to have influenced the present results. If the local ecology for *Galago moholi* is such that economy of movement is

much more important than speed, this would explain why it alone chooses to leap at more or less 45° all the time.

Another feature of the animal's ecology that may influence their leaping behaviour is the density of the vegetation in their normal habitat. The *Cheirogaleidae* live in dense undergrowth tangles where there is only room for shallow trajectory jumps. This may lead to an in-built preference for a low takeoff angle even when there is room for greater elevation.

Where substrates are significantly bent by the force applied by the animal leaping, the optimally efficient takeoff angle may not be 45° for a leap with no height change. The animal will lose energy by bending the support. The maximum bending will occur if the force applied is perpendicular to the branch, and will be minimized if the force is along the axis of the branch since buckling requires considerably greater forces than bending. For a horizontal support, a shallow takeoff trajectory will cause less bending, and hence less energy transfer to the substrate.[29] The actual value of the optimal takeoff angle will depend on the resilience, orientation and shear modulus of the branch, but will lie somewhere between the branch orientation and 45°.

Finally, leaping animals may be very vulnerable to aerial predators whilst leaping in the open. A flatter than energetically optimal trajectory ensures that they are airborne for a shorter time. Also, if an animal always chooses exactly the same trajectory for its leaps, then a predator would be able to predict the mid-air position of an animal that it has spotted

---

[29]In the experimental setup used, the substrate was not noticeably bent by any of the animals and so can effectively be considered as a rigid support, causing no appreciable loss or gain of energy.

about to jump out of the cover of a tree. This would help to explain the relatively large amount of variation of the observed takeoff angles.

Distance Relationships

Theory

There are three key biomechanical parameters that can be readily measured for each leap. These are: the peak force generated during takeoff; the extension distance of the hind-limb; and the duration of this extension. For a given animal, these parameters would be expected to vary depending on the distance being leapt. The effect of takeoff angle has already been explained so that all the leap distances can be re-evaluated as if the leap had been done at 45°. Thus, with the trajectory fixed, the leap distance depends solely on the takeoff velocity.

The relationship with extension duration can be investigated by considering momentum[30]. The change in momentum equals the impulse applied by the animal as shown in the following equation:

$$(1) \qquad \int_{t_0}^{t_{to}} \mathbf{F}(t)\ dt = mv_{to} - mv_0$$

Where:

$\mathbf{F}(t)$	is the function of force with respect to time
t_0	is the time at the start of the leap
t_{to}	is the time at takeoff
m	is the mass of the animal
v_{to}	is the takeoff velocity
v_0	is the start velocity

[30]The momentum of a body is the product of its velocity and its mass.

This is the generic form of the equation using vectors representing the force function and the velocities. In the takeoff phase of a leap, the centre of mass of the animal moves in a straight line at the takeoff angle. Thus, the two dimensional problem can be simplified down to just one dimension. However, in doing this, it must be remembered that as well as applying a force to accelerate the centre of mass along this trajectory, the animal must also apply a vertical force to compensate for the pull of gravity. This force is independent of the distance leapt, and adds an appreciable amount to the total force that needs to be generated at lower distances. The results shown subsequently are for the resultant force derived from the acceleration of the centre of mass. The comparative magnitude of the peak forces required in the leap compared to the force due to gravity has been shown to vary from 13.5 times in the case of *Galago moholi* to 4.5 times for *Lemur catta* (Günther 1989). In my experiments, the value is about 5 for all the animals except *Cheirogaleus major* which has a value of 8. However, as will be shown at the end of the chapter, these values depend on the distance leapt, and one of the problems with calculations from kinematics instead of force plate measurements is that peaks tend to be missed due to the relatively coarse temporal resolution.

In this case, the start velocity is zero since the animals leap from rest. The start time can be arbitrarily set to zero, so that the takeoff time is the duration of the extension. Also, for our idealized 45° leap, the takeoff velocity is related to the leap distance by the following equation:

(2) $$v_{to} = \sqrt{rg}$$

Where:

r is the range of the leap

g is the acceleration due to gravity

So:

(3) $$\int_{t_0}^{t_{to}} F(t)\ dt = m\sqrt{rg}$$

Where:

F(t) is the resultant magnitude force function.

The extension distance relates to the takeoff force according to Newton's second law (acting in a straight line):

(4) $$F(t) = ma$$

Where:

a is the acceleration of the centre of mass

And since:

(5) $$a = \frac{d^2s}{dt^2}$$

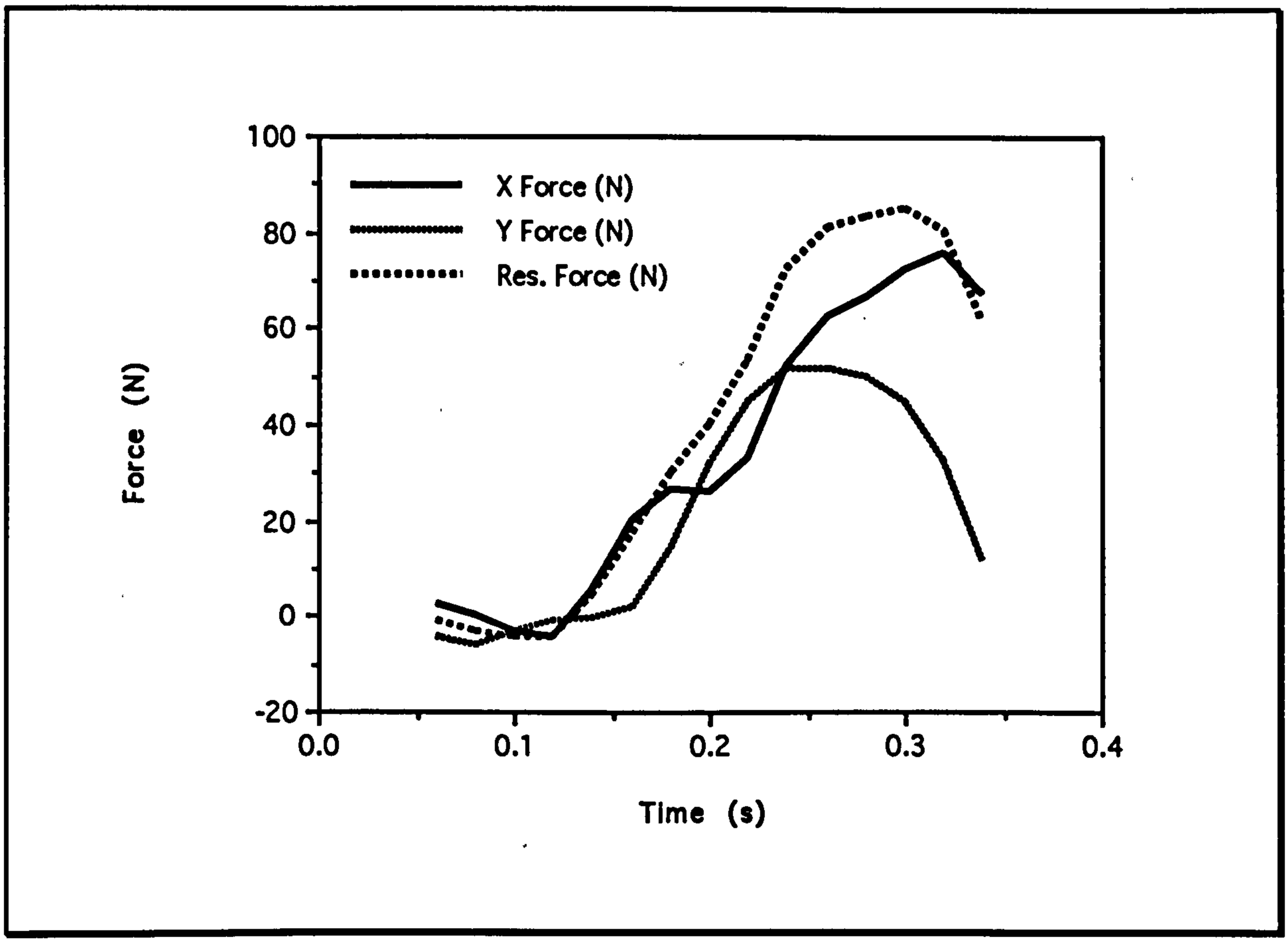
Where:

s is the extension distance

From (4) and (5)

(6) $$\frac{d^2s}{dt^2} = \frac{F(t)}{m}$$

To progress any further, the function of force with respect to time needs to be defined. An actual force/time graph, irrespective of the mass of the animal or the distance being leapt has a shape something like this:



Graph showing the component and resultant forces[31] calculated for *Lemur catta* leaping 2.22 m. The mass of the animal was 2.7 kg and the trajectory 36°.

For modelling purposes relatively simple force functions can be obtained from the maximum recorded force for a leap as the limit for a simple polynomial relationship mapped to the leap duration with an arbitrary start time of zero. Three such functions are illustrated below:

(7) $$F(t) = F_{max}$$

(8) $$F(t) = \frac{F_{max}t}{t_{to}}$$

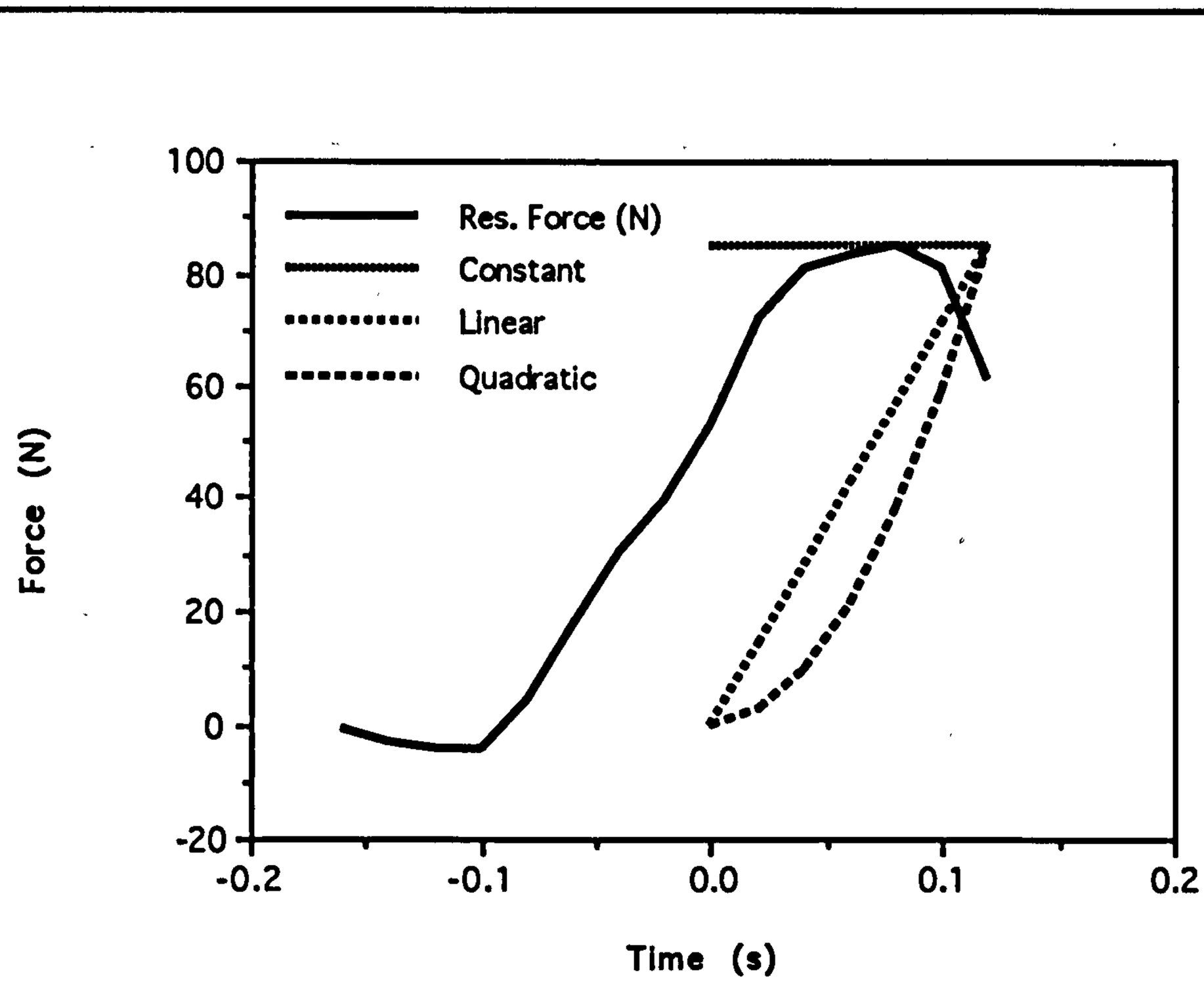
---

[31]The resultant force has been calculated by summing the components of the X and Y forces in the direction of the leaping trajectory:

$$F_{resultant} = F_x\cos\theta + F_y\cos(90 - \theta)$$

(9) $$F(t) = \frac{F_{max}t^2}{t_{to}^2}$$

The suitability of these functions can be seen from the following graph:



Graph showing the values predicted by the various force models that might be chosen. Zero on the time scale is where the takeoff phase is judged to begin since this is the first time when the hind-limb is seen to extend, and the centre of mass of the animal to have a positive velocity along the trajectory.

There are a number of interesting points to note about this comparison. Firstly, the start of the leap is defined by when the animal starts to move along its takeoff trajectory, not when it first starts to produce a resultant force along this trajectory. This is because the animal starts its leap by flexing its leg muscles allowing the force of gravity to move the centre of mass downwards. It can then apply a force to decelerate this downward movement so that when it starts accelerating its centre of mass along the

takeoff trajectory, it has already generated a significant amount of tension in its leg muscles. This helps alleviate the slow buildup to maximum muscle tension, and means that a higher mean muscle tension can be maintained during the course of the takeoff phase.

Since this mechanism means that the start force is appreciably higher than zero, the *constant* force model best fits the observed data, though it does overestimate the impulse somewhat. The *linear* model underestimates the impulse, and the *quadratic* model even more so. Since reality lies between the constant and linear force models, evaluating both should produce a reasonable estimate of the range within which observed values would be expected to fall.

Now equation (3) can be solved for both these force models:

Constant:

(10) $$\int_0^{t_{to}} F_{max}\, dt = \Big[F_{max} t\Big]_{t_{to}}^{0}$$

(11) $$m\sqrt{rg} = F_{max}\, t_{takeoff}$$

Linear:

(12) $$\int_0^{t_{to}} \frac{F_{max} t}{t_{to}}\, dt = \left[\frac{F_{max}\, t^2}{2t_{to}}\right]_{t_{to}}^{0}$$

(13) $$m\sqrt{rg} = \tfrac{1}{2} F_{max}\, t_{takeoff}$$

And likewise, equation (6) can also be solved:

Constant:

(14) $$\frac{d^2s}{dt^2} = \frac{F_{max}}{m}$$

(15) $$s = \frac{F_{max}\, t^2}{2m} + Ct + D$$

Linear:

(16) $$\frac{d^2s}{dt^2} = \frac{F_{max}t}{t_{to}m}$$

(17) $$s = \frac{F_{max}\, t^3}{6t_{to}m} + Ct + D$$

For this problem, at t = 0, both v = 0 and s = 0, so C = 0 and D = 0. Also, for the case in which I am interested, t = $t_{takeoff}$, so equations (15) and (17) both simplify:

Constant:

(18) $$s = \frac{F_{max}t_{to}^2}{2m}$$

Linear:

(19) $$s = \frac{F_{max}t_{to}^2}{6m}$$

It is interesting to note that the two different force models do not alter the indices of the various powers in the eventual relationships, only the values of the constants. So that the relationship between the peak force, the extension distance and the takeoff duration will be equivalent for the two cases.

To see this relationship, equation (11) can be rearranged and substituted into equation (18) to eliminate t_{to}:

(20) $$t_{to}^2 = \frac{m^2 rg}{F_{max}^2}$$

(21) $$s = \frac{mrg}{2F_{max}}$$

And for the linear model:

(22) $$s = \frac{2mrg}{3F_{max}}$$

This provides two equations relating leap distance to the three measured parameters. The interrelationship between these parameters is therefore not unique, but if we make the not unreasonable assumption that the animal will generally choose to use its full hind-limb extension for all leaps, thereby minimizing the forces needed, then a third relationship can be obtained:

(23) $$s = s_{max}$$

Where:

s_{max} is the maximum extension of the hind-limb

Now, from (21) and (23):

(24) $$F_{max} = \frac{mrg}{2s_{max}}$$

And from (24) and (11):

(25) $$t_{to} = \frac{2s_{max}}{\sqrt{rg}}$$

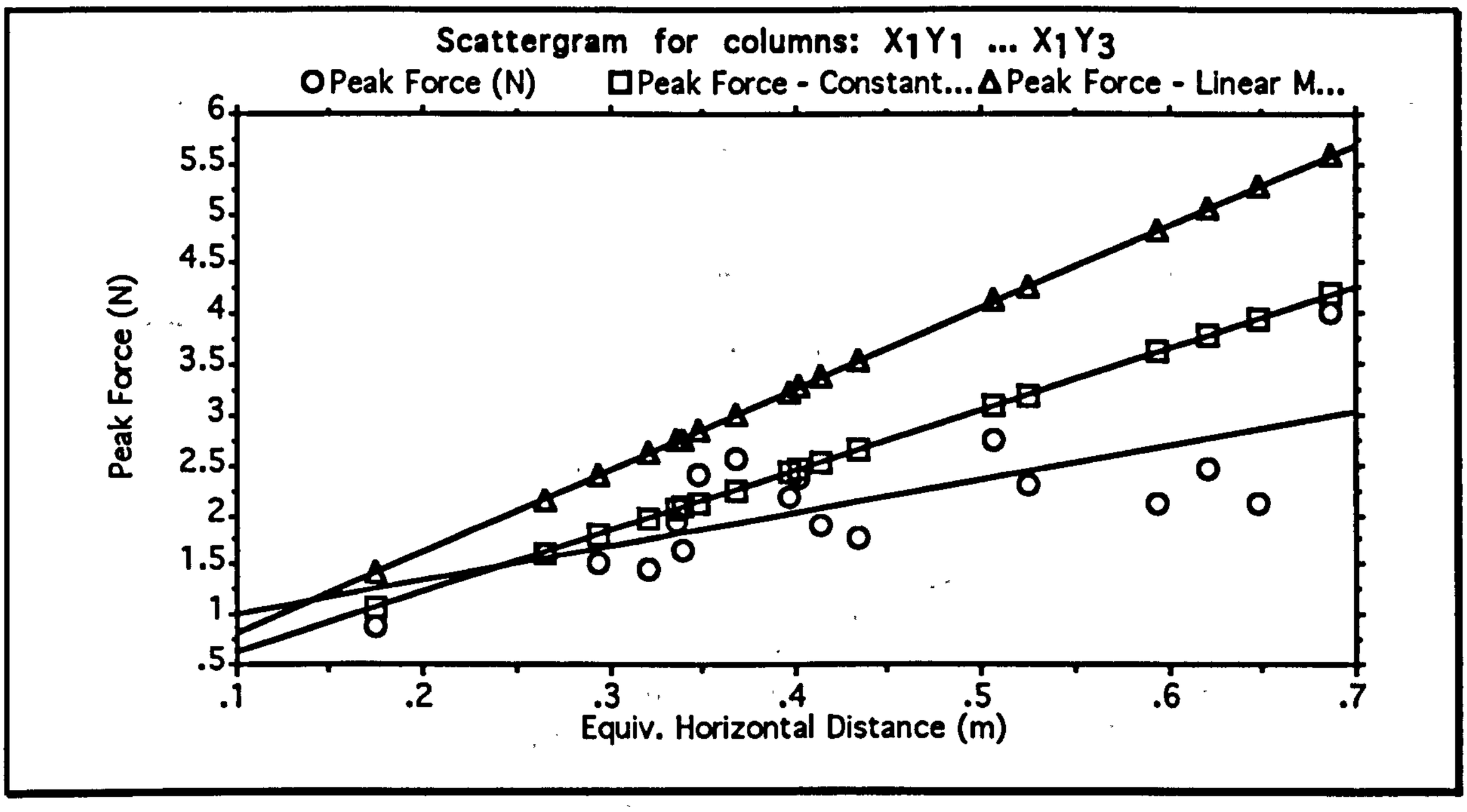
Similarly, for the linear model:

(26) $$F_{max} = \frac{2mrg}{3s_{max}}$$

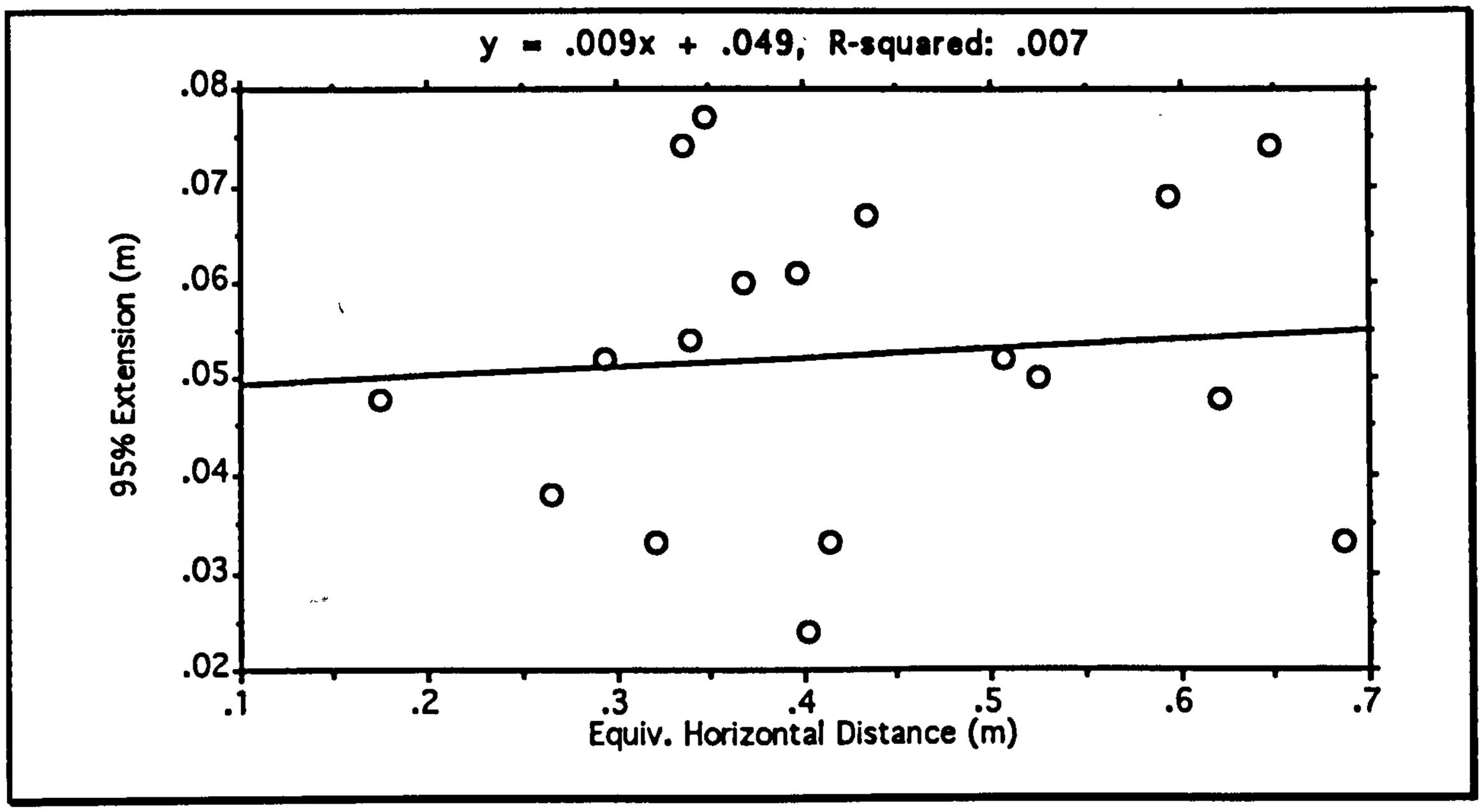
(27) $$t_{to} = \frac{3s_{max}}{\sqrt{rg}}$$

Results

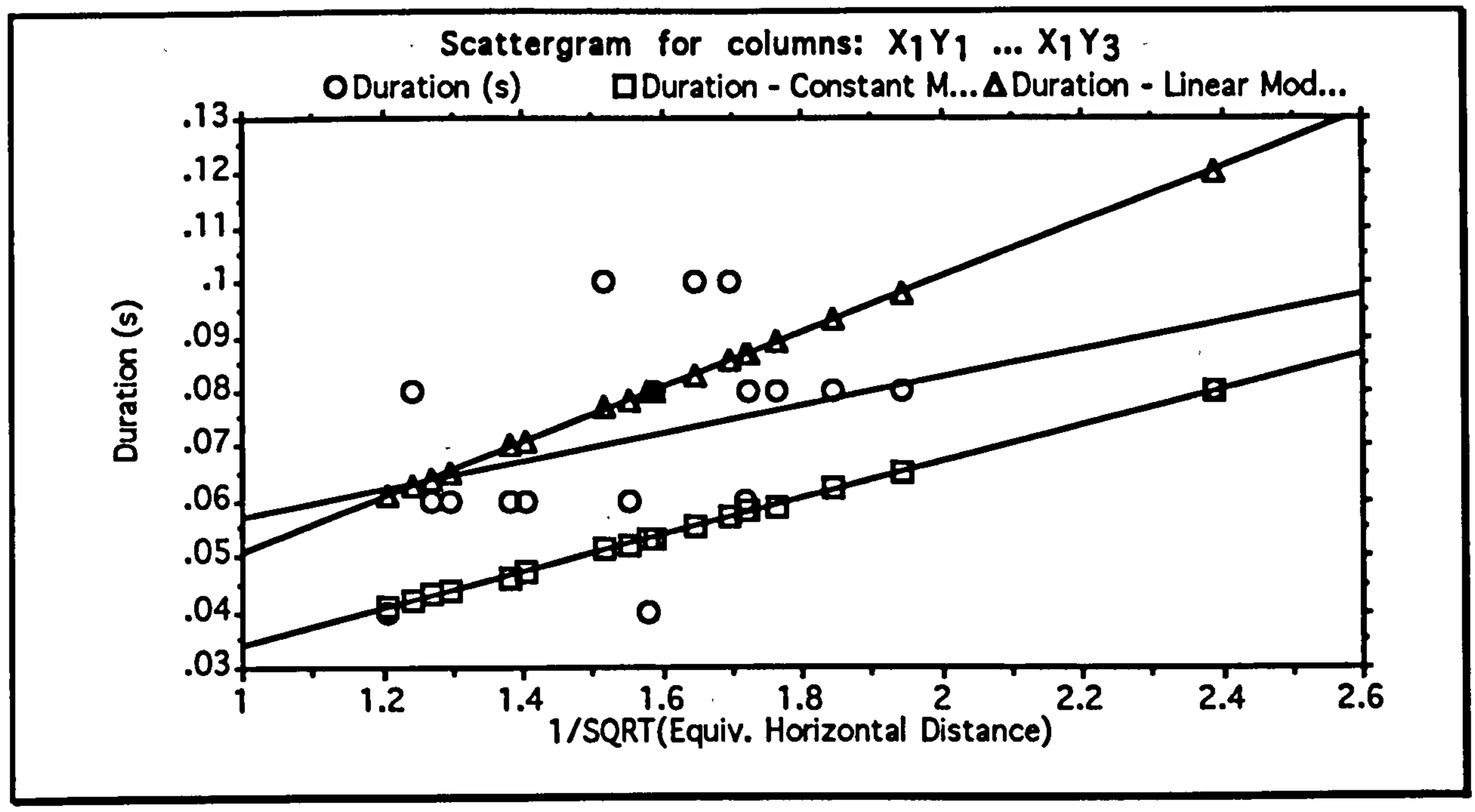
The results for each animal follow. The effective distance is calculated from the measured takeoff velocity at a trajectory of 45°. Since we are assuming that the extension distance does not vary with the leap distance, I have used the mean value for the extension distance to calculate the predicted values throughout.



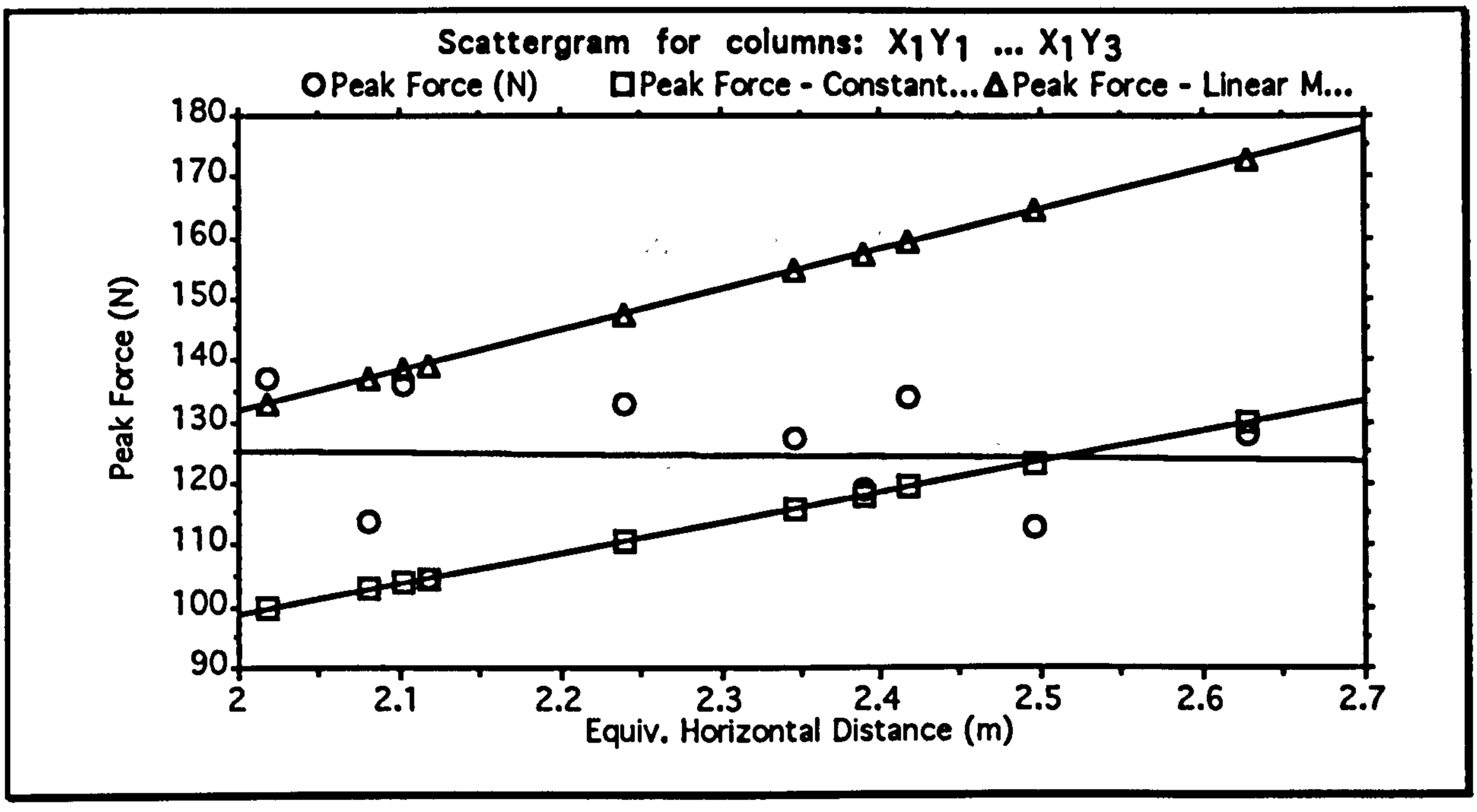
Graph showing the calculated peak force exerted by *Microcebus murinus* for a variety of leap distances. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0006.



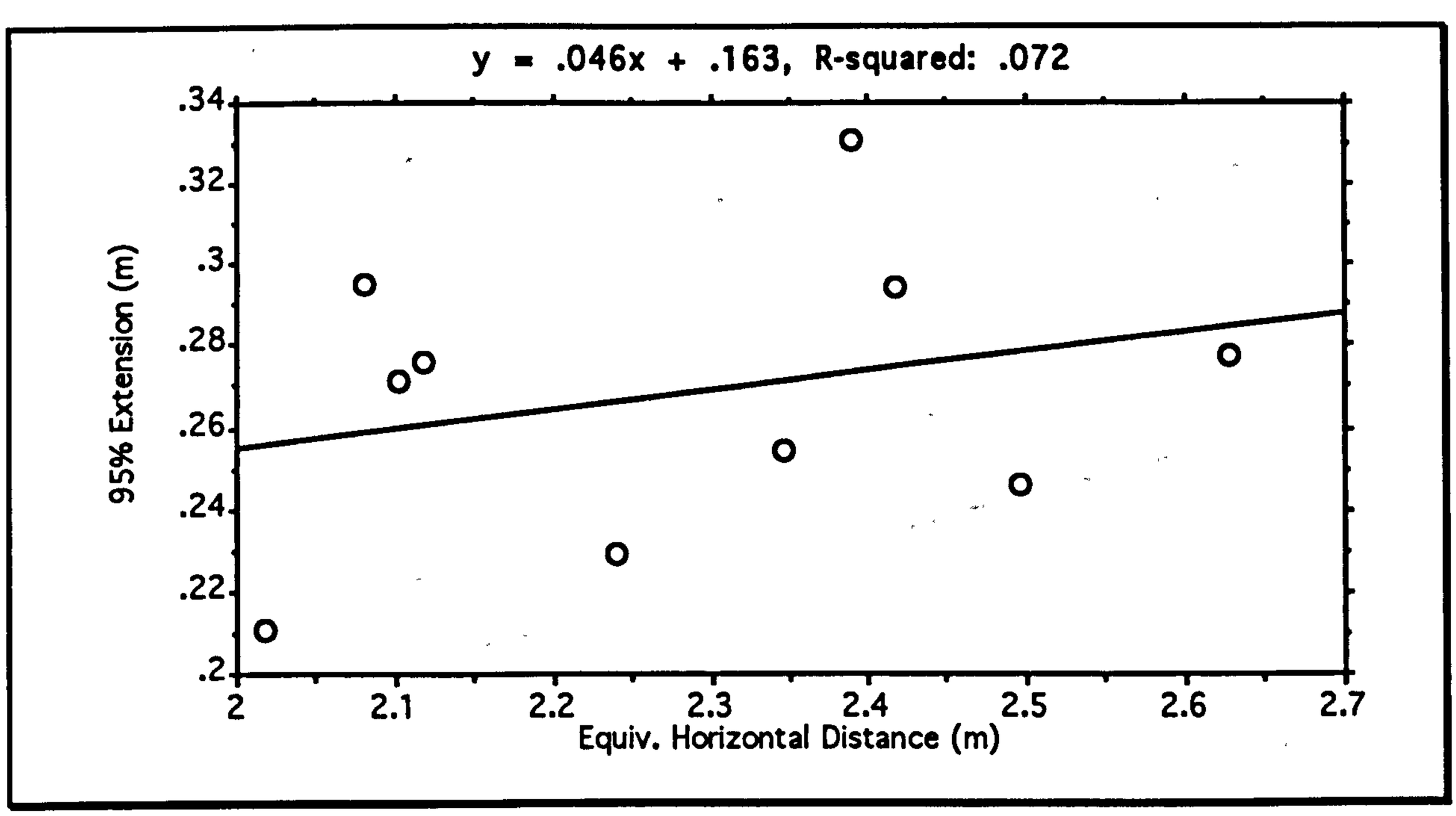
Graph showing the extension distance for *Microcebus murinus* for a variety of leap distances. There is no significant relationship.



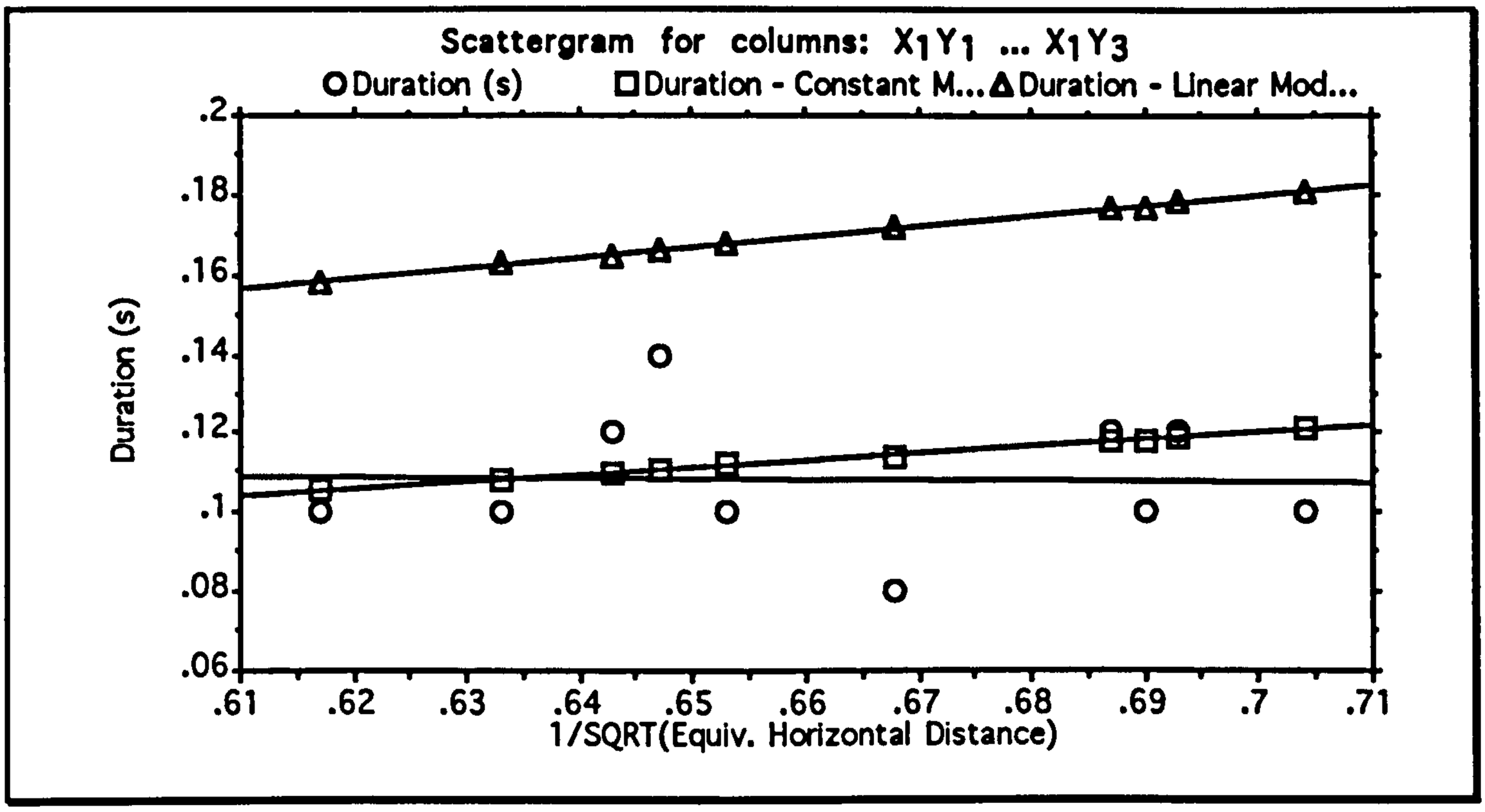
Graph showing the extension duration of *Microcebus murinus* for a variety of leap distances. The leap distance has been transformed by raising it to the power minus one half as predicted by the model. The two straight lines indicate the values predicted by the two force models. There is no significant relationship.



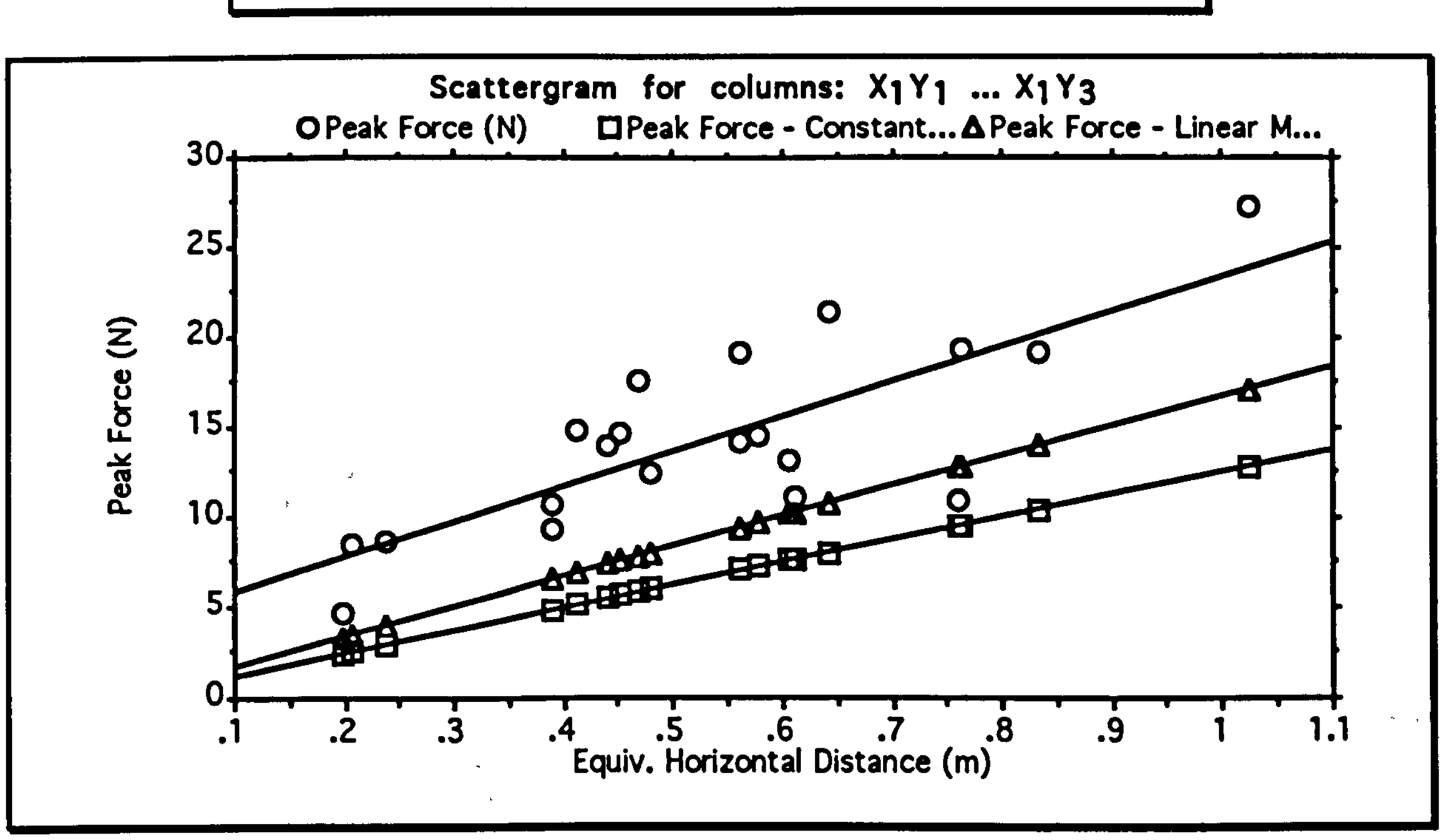
Graph showing the calculated peak force exerted by *Lemur catta* for a variety of leap distances. The two straight lines indicate the values predicted by the two force models. There is no significant relationship.



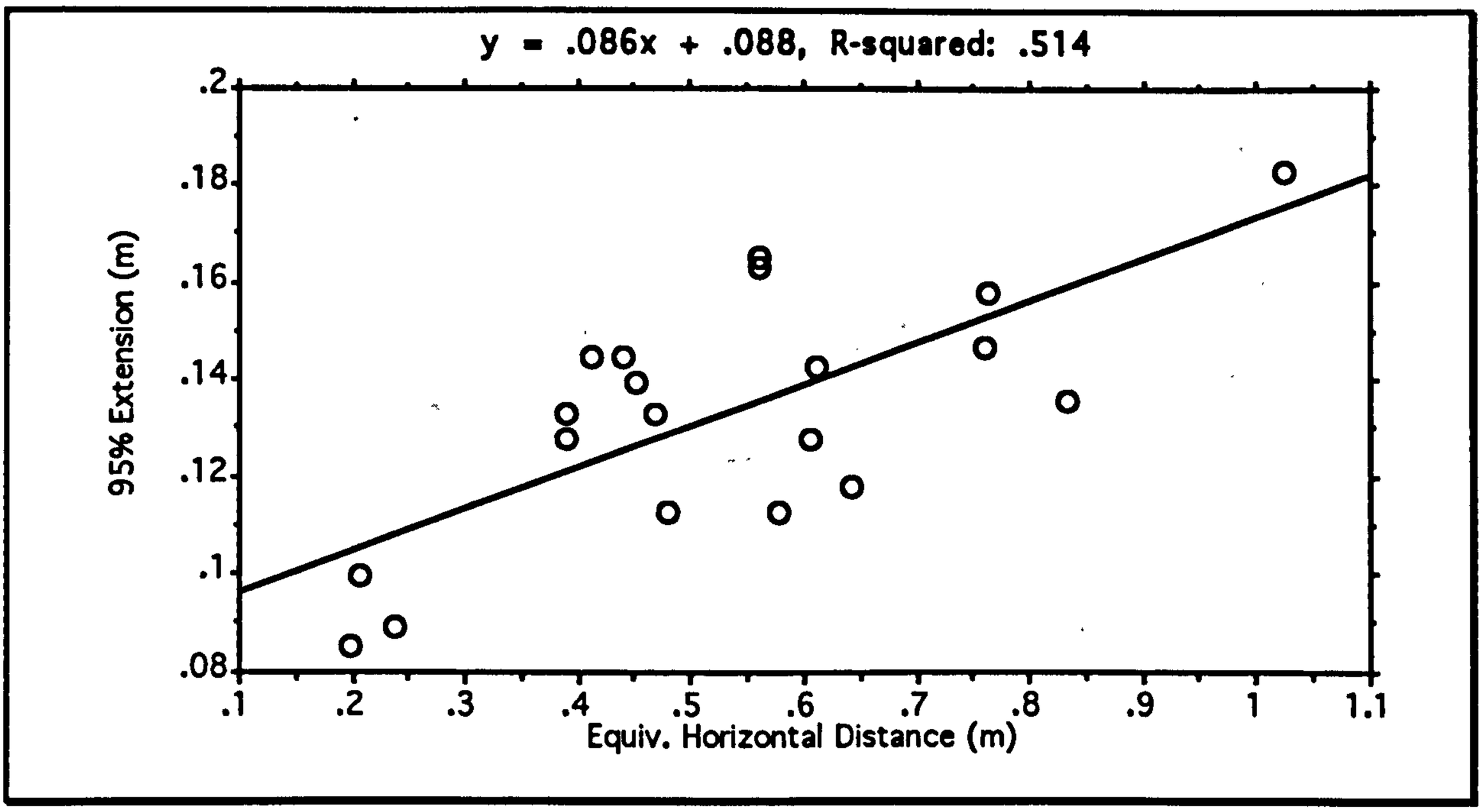
Graph showing the extension distance for *Lemur catta* for a variety of leap distances. There is no significant relationship.



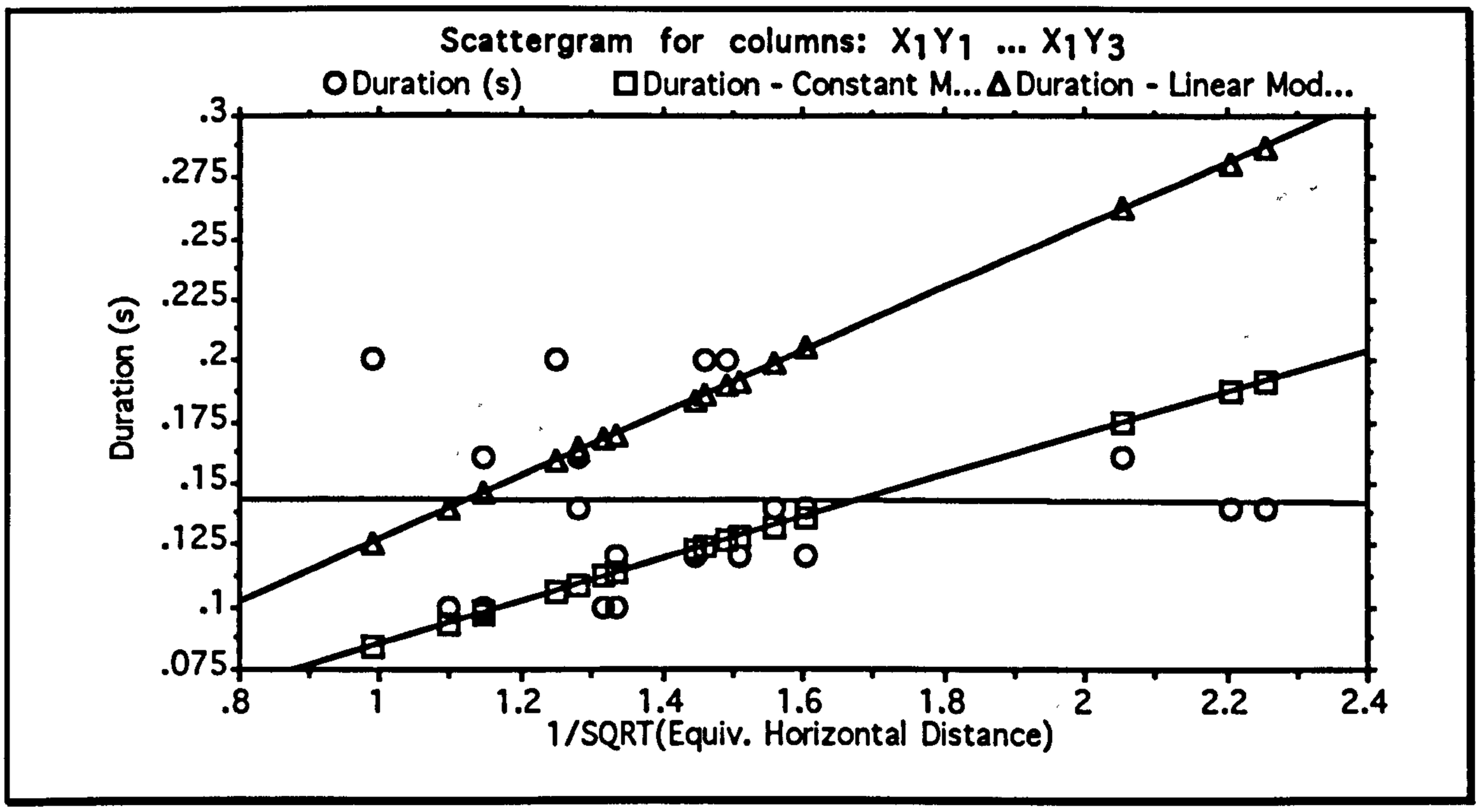
Graph showing the extension duration of *Lemur catta* for a variety of leap distances. The leap distance has been transformed by raising it to the power minus one half as predicted by the model. The two straight lines indicate the values predicted by the two force models. There is no significant relationship.



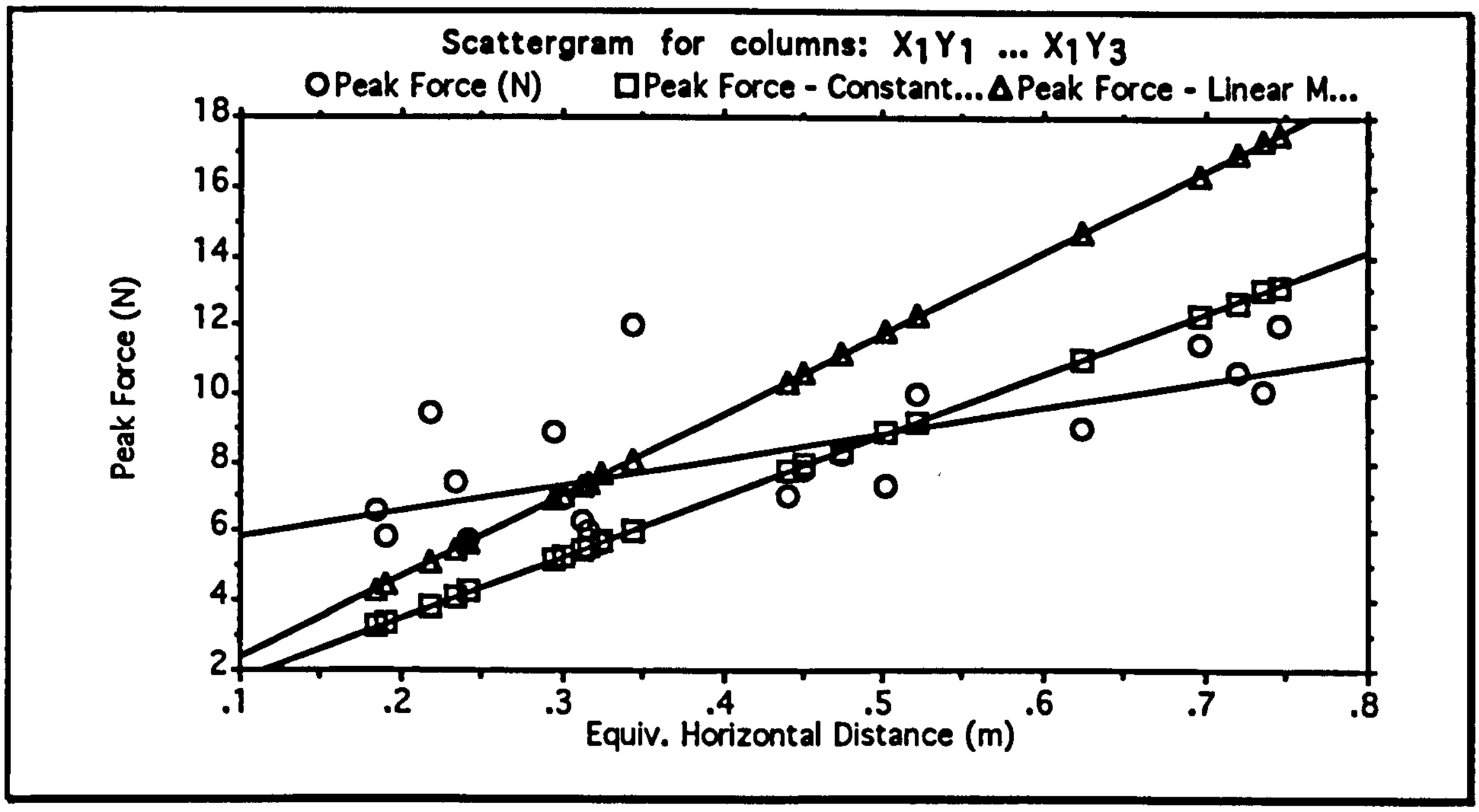
Graph showing the calculated peak force exerted by *Chetrogaleus major* for a variety of leap distances. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0001.



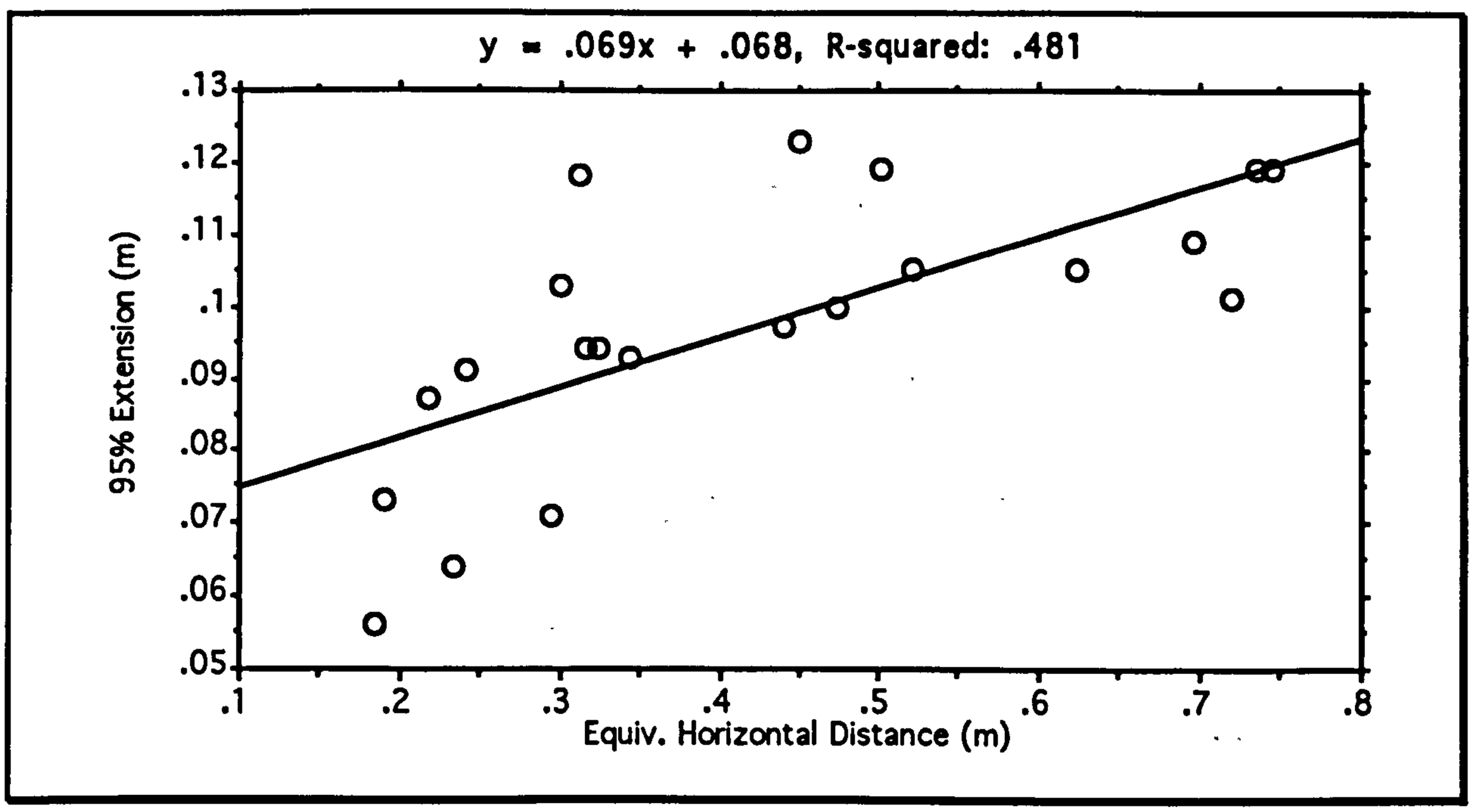
Graph showing the extension distance for *Chetrogaleus major* for a variety of leap distances. The significance of the regression line is 0.0004.



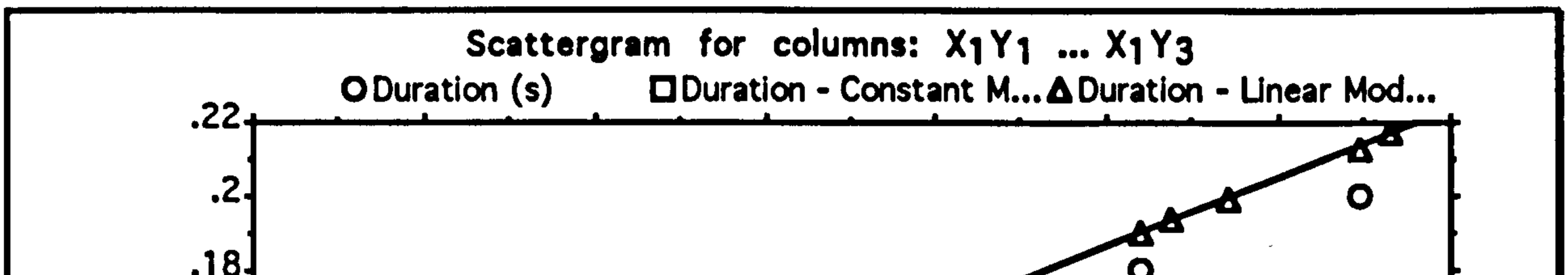
Graph showing the extension duration of *Chetrogaleus major* for a variety of leap distances. The leap distance has been transformed by raising it to the power minus one half as predicted by the model. The two straight lines indicate the values predicted by the two force models. There is no significant relationship.



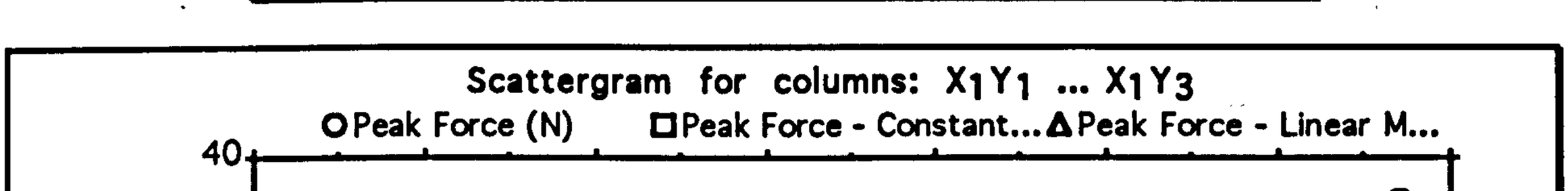
Graph showing the calculated peak force exerted by *Mirza coquereli* for a variety of leap distances. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0008.



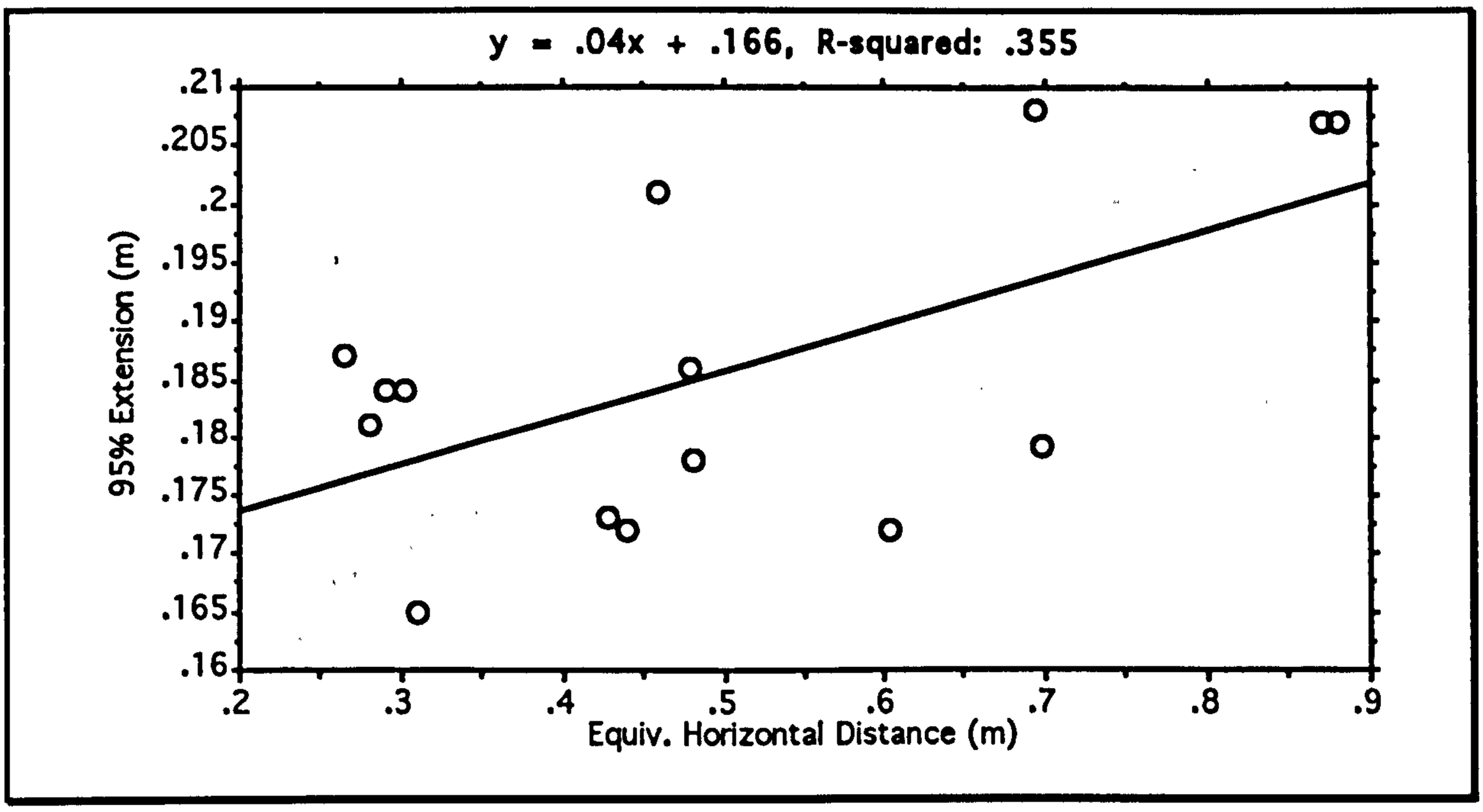
Graph showing the extension distance for *Mirza coquereli* for a variety of leap distances. The significance level of the regression line is 0.0005.



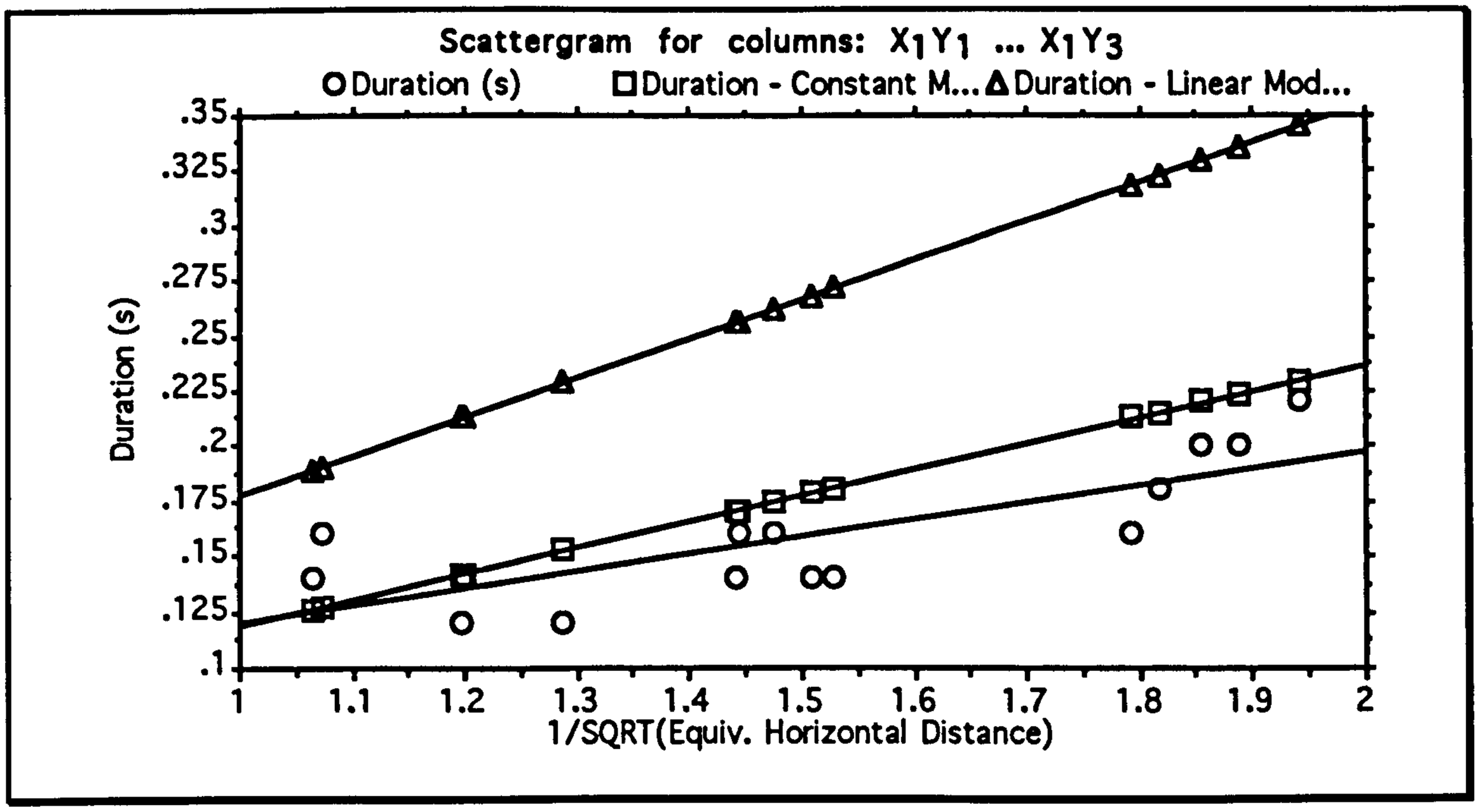
Graph showing the extension duration of *Mirza coquereli* for a variety of leap distances. The leap distance has been transformed by raising it to the power minus one half as predicted by the model. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0034.



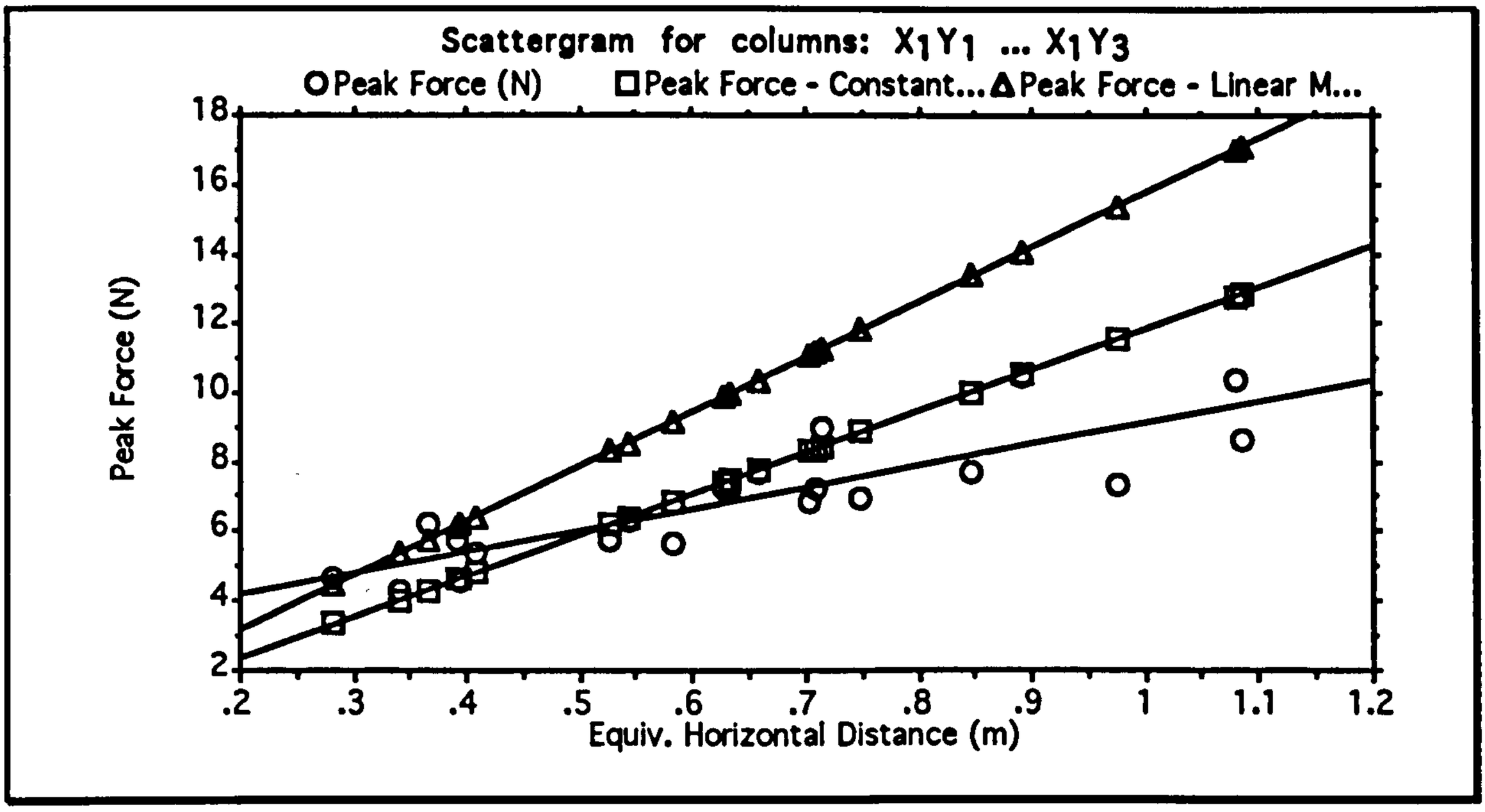
Graph showing the calculated peak force exerted by *Galago garnettii* for a variety of leap distances. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0016



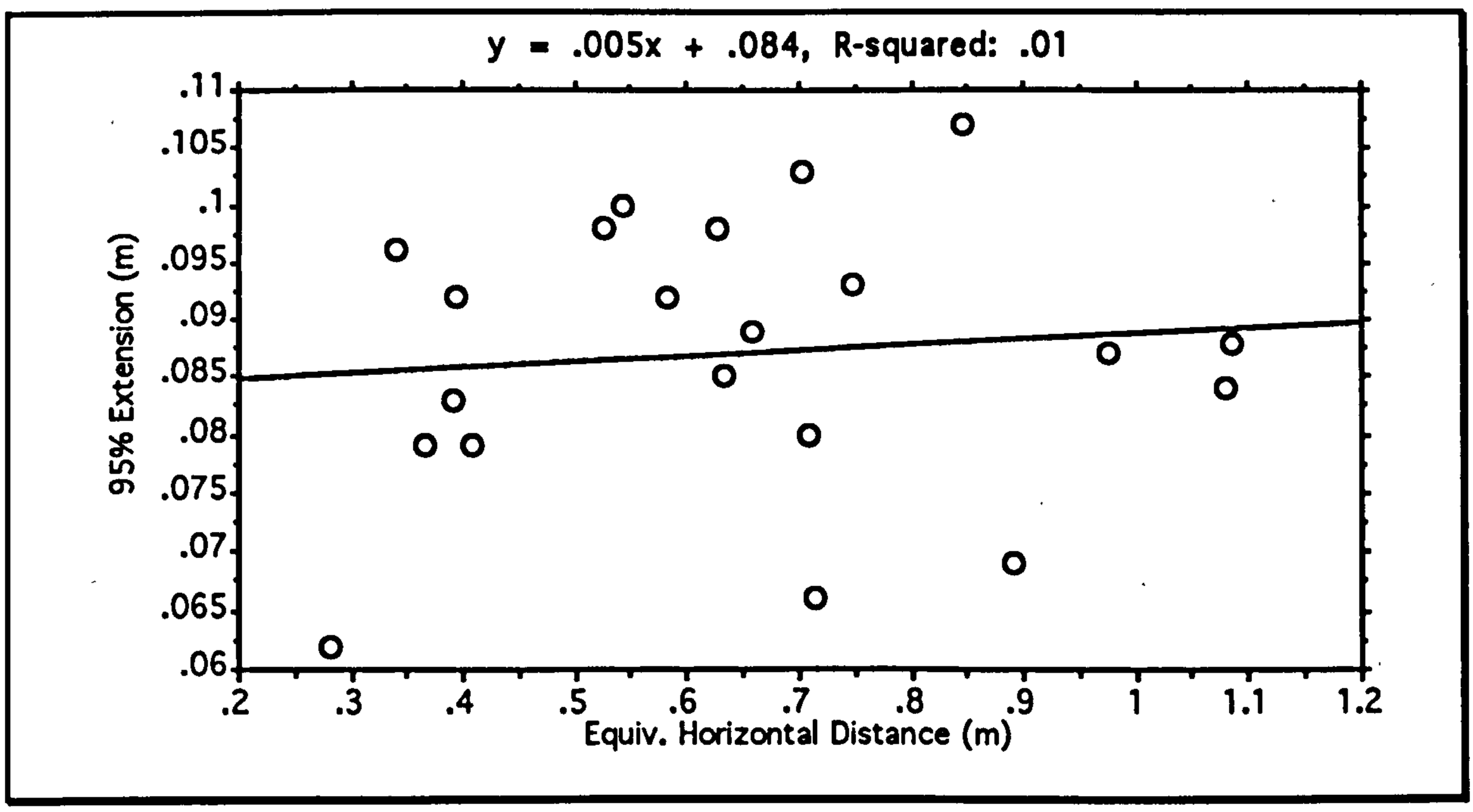
Graph showing the extension distance for *Galago garnettii* for a variety of leap distances. There is no significant relationship.



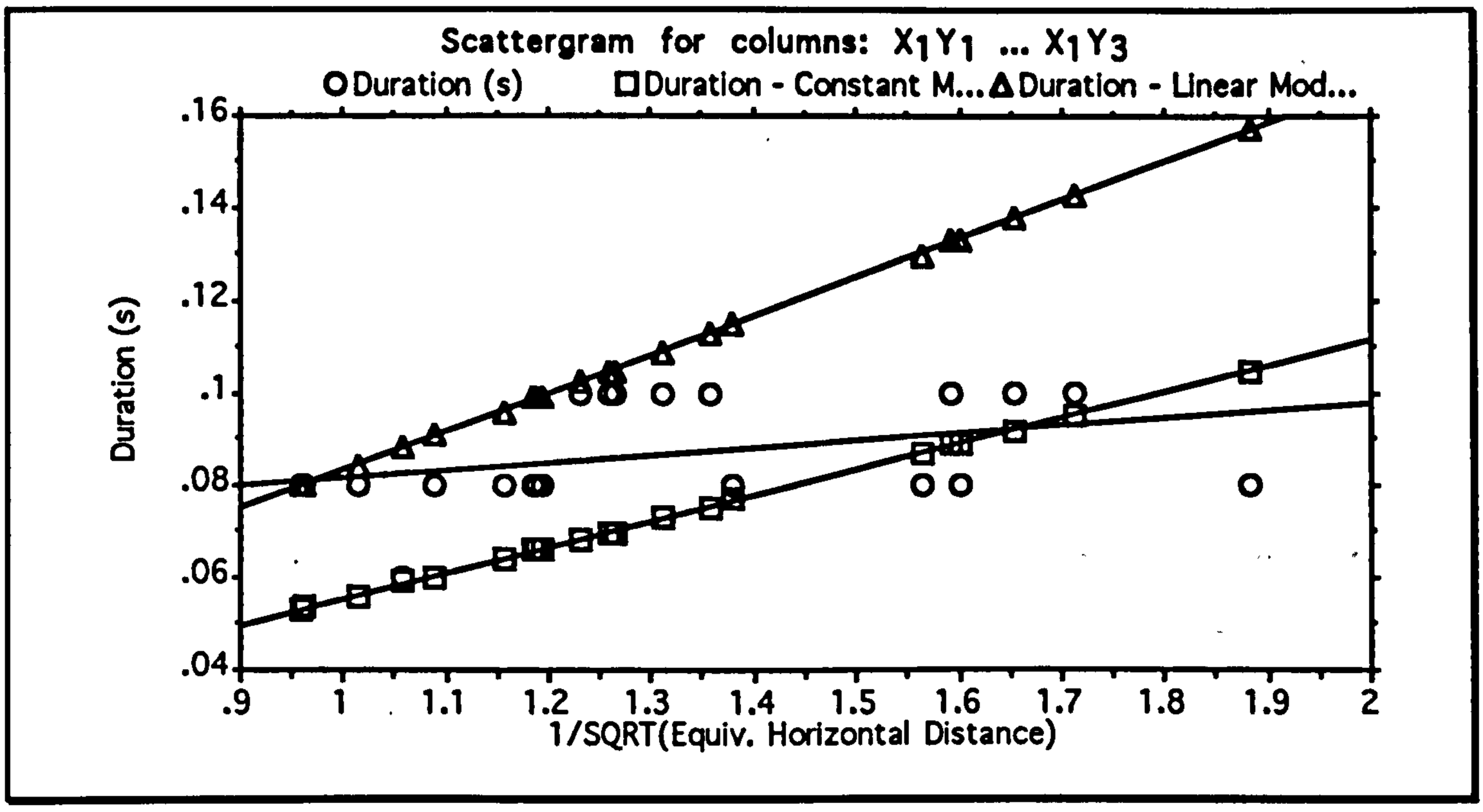
Graph showing the extension duration of *Galago garnettii* for a variety of leap distances. The leap distance has been transformed by raising it to the power minus one half as predicted by the model. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0005



Graph showing the calculated peak force exerted by *Galago moholi* for a variety of leap distances. The two straight lines indicate the values predicted by the two force models. The significance level of the regression line is 0.0001



Graph showing the extension distance for *Galago moholi* for a variety of leap distances. There is no significant relationship.



Graph showing the extension duration of *Galago moholi* for a variety of leap distances. The leap distance has been transformed by raising it to the power minus one half as predicted by the model. The two straight lines indicate the values predicted by the two force models. There is no significant relationship.

For *Microcebus murinus*, the agreement between the measured values and the model is reasonably good. There is a tendency to underestimate peak forces, especially for small, quick animals, due to the relatively slow framing rate used in this study. There is quite a large amount of variation in the hind-limb extension distance, but this is largely independent of the leap distance, so modelling it by a constant function seems reasonable. The relationship with the duration of the extension phase shows most of the values clumped within the expected range. It is difficult to measure this duration at all precisely, and this would explain the poor level of correlation.

For *Lemur catta* the result is also in good agreement with the model's predictions. For this animal, all the leap distances were nominally the same, with only differences in leaping trajectory and precise landing

points providing any variation. Thus I would not expect to see any significance in the correlations because measurement errors will swamp any distance effects.

Cheirogaleus major, however, fits the model much less well. There is a notable increase of extension distance with leap range and instead, extension time remains more or less constant. This reversal will affect the predicted force curves too, with recorded peak forces being noticeably higher than those predicted. Qualitatively, the leaping style of *Cheirogaleus major* is different from the other animals. It is the least enthusiastic jumper and for the longer leaps, it curls itself up into a very tight ball, and uses its back muscles to throw the upper torso forward during the leap.

Mirza coquereli also shows an increase in the hind-limb extension with leap distance. This affects the other curves much more predictably though with peak force at longer lengths being overestimated and underestimated at shorter lengths. The time curve is still relatively well predicted, and any errors in the force curve are relatively small.

The predictions for *Galago garnettii* and *Galago moholi* are again reasonable. There is appreciable scatter about the modelled curves but it seems acceptable given the generally high level of noise in the data.

Gravity Effects

The comparative magnitude of the force due to gravity can be calculated from equation (1):

$$\int_{t_0}^{t_{to}} \mathbf{F}(t)\ dt + \int_{t_0}^{t_{to}} mg\ dt = mv_{to} - mv_0 \qquad (28)$$

Where:

g is the acceleration vector due to gravity.

This can be divided into its X and Y components, and using the constant force model as before, produces the following 2 equations:

(29) $$F_x t_{to} = \frac{m v_{to}}{\sqrt{2}}$$

(30) $$F_y t_{to} = \frac{m v_{to}}{\sqrt{2}} + m g t_{to}$$

So, as would be expected, gravity has no effect on the force required in the X direction (from equation (29)). Rearranging (30) gives:

(31) $$F_y = m\left(\frac{v_{to}}{\sqrt{2 t_{to}}} + g\right)$$

So, the importance of gravity depends on the relative magnitudes of g and $\frac{v_{to}}{\sqrt{2 t_{to}}}$. The value of v_{to} depends solely on the leap distance, so that as leap distance increases, the support component decreases. The value of t_{to} depends on both the leap distance (it falls as leap distance increases) and on the length of the animal's hind-limbs. If the hind-limbs are longer, then the takeoff duration will increase. This explains why the the force of gravity acting on a large animal is larger proportion of the force it needs to apply for the leap. However, even for the largest animal in my study, *Lemur catta*, $\frac{v_{to}}{\sqrt{2 t_{to}}}$ has a value of approximately 4 times the value of g.

Mass Relationships

Theory

Relationships of parameters with the mass of the animal depend on the scaling model used. These relate linear measurements on the animal to the animal's overall mass for animals of similar shapes. Three so called "Similarity" models are commonly encountered:

Geometric Similarity

Linear distances are scaled isotropically (Hill 1950). The scaling with mass is a simple geometric relationship with all axes scaling equally. This can be described by the following equation:

(1) $$L \propto m^{\frac{1}{3}}$$

Where:

L is the characteristic length

m is the mass of the animal

Elastic Similarity

Linear distances are scaled anisotropically, with characteristic lengths increasing less rapidly than diameters so that the **bending** stress on the skeleton due to its own weight is kept constant for animals of different masses (McMahon 1973). This can be described by the following equation:

(2) $$L \propto m^{\frac{1}{4}}$$

Constant Stress Similarity

Linear distances are scaled anisotropically, with characteristic lengths increasing even less rapidly than diameters so that the **breaking** stress on the skeleton due to its own weight is kept constant for animals of different masses (McMahon 1973). This can be described by the following equation:

(3) $$L \propto m^{\frac{1}{3}}$$

The extension distance during the takeoff phase of the leap is primarily a function of the length of the hind-limb so can be related to the mass of the animal by these three equations depending which scaling model is more appropriate. This enables certain predictions about the mass dependency of the peak force, the extension distance and the extension duration to be made.

Considering the geometric similarity model:

From (1):

(4) $$s \propto m^{\frac{1}{3}}$$

Where:

s is the extension length

From (11) in the previous chapter:

(5) $$m\sqrt{rg} = F_{max}t_{to}$$

Where:

r is the leap distance

F_{max} is the peak force

t_{to} is the takeoff duration

From (18) in the previous chapter:

(6) $$s = \frac{F_{max} t_{to}^2}{2m}$$

Substituting (4) in (6) gives:

(7) $$m^{\frac{1}{3}} \propto \frac{F_{max} t_{to}^2}{2m}$$

And rearranging:

(8) $$t_{to} \propto m^{\frac{2}{3}} F_{max}^{\frac{1}{2}}$$

Substituting into (5)

(9) $$\sqrt{rg} \propto m^{\frac{2}{3}} F_{max}^{\frac{1}{2}}$$

And rearranging:

(10) $$F_{max} \propto rgm^{\frac{2}{3}}$$

By substituting (10) into (8) and rearranging:

(11) $$t_{to} \propto \frac{m^{\frac{1}{3}}}{\sqrt{rg}}$$

Considering the elastic similarity model:

From (2):

(12) $$s \propto m^{\frac{1}{4}}$$

And by a similar rearrangement as before:

(13) $$F_{max} \propto rgm^{\frac{3}{4}}$$

And:

(14) $$t_{to} \propto \frac{m^{\frac{1}{4}}}{\sqrt{rg}}$$

For the constant stress similarity model:

(15) $$s \propto m^{\frac{1}{3}}$$

(16) $$F_{max} \propto rgm^{\frac{4}{3}}$$

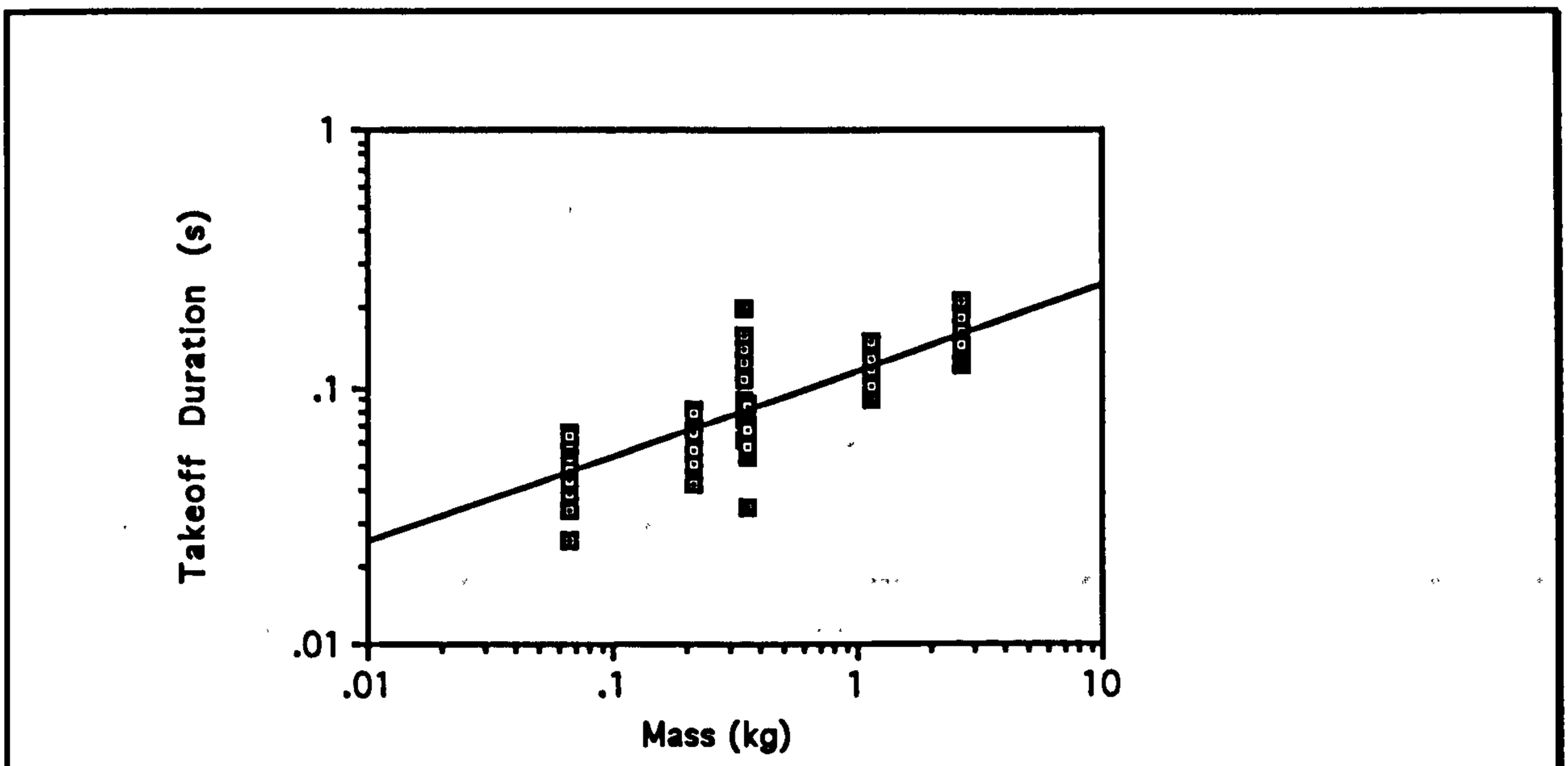
(17) $$t_{to} \propto \frac{m^{\frac{1}{3}}}{\sqrt{rg}}$$

Results

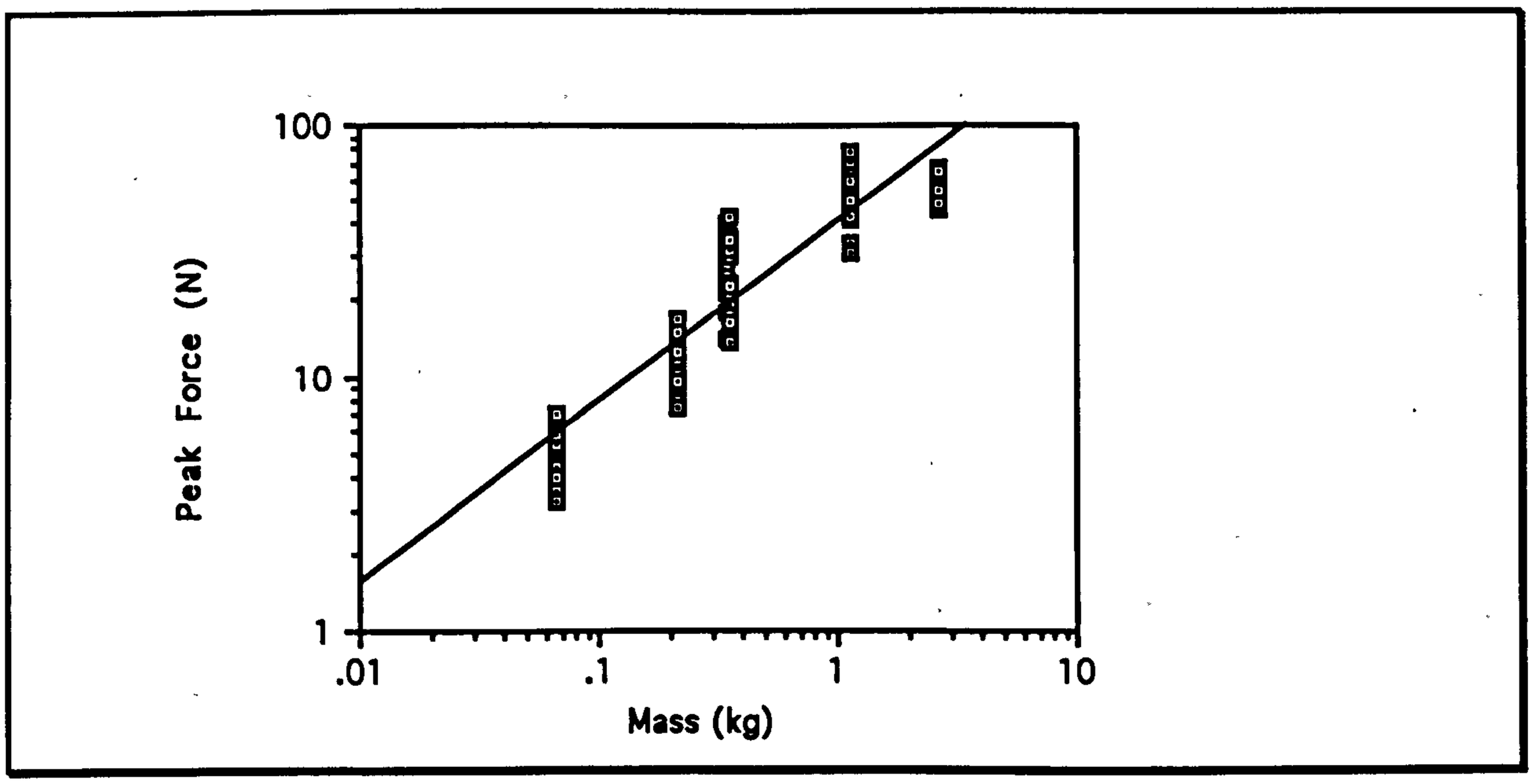
The six different species investigated covered a 40 fold mass range so are eminently suitable for investigating mass dependent effects. The predictions of the similarity models vary by the index of the mass. To calculate this, the the peak force, extension distance and extension duration are plotted against mass on log scale graphs. The gradient of a straight line fitted to these transformed points is the power of the mass that best fits the data. The r^2 value is an indication of how precisely this value is estimated by the data.

All the data points have been normalized for a 1 m leap. This has been done using the modelling assumptions in the previous chapter. Thus, the extension distance is assumed to be independent of the leap range; the peak force is divided by the effective horizontal leaping distance as this relationship is assumed to be linear; and the extension duration is

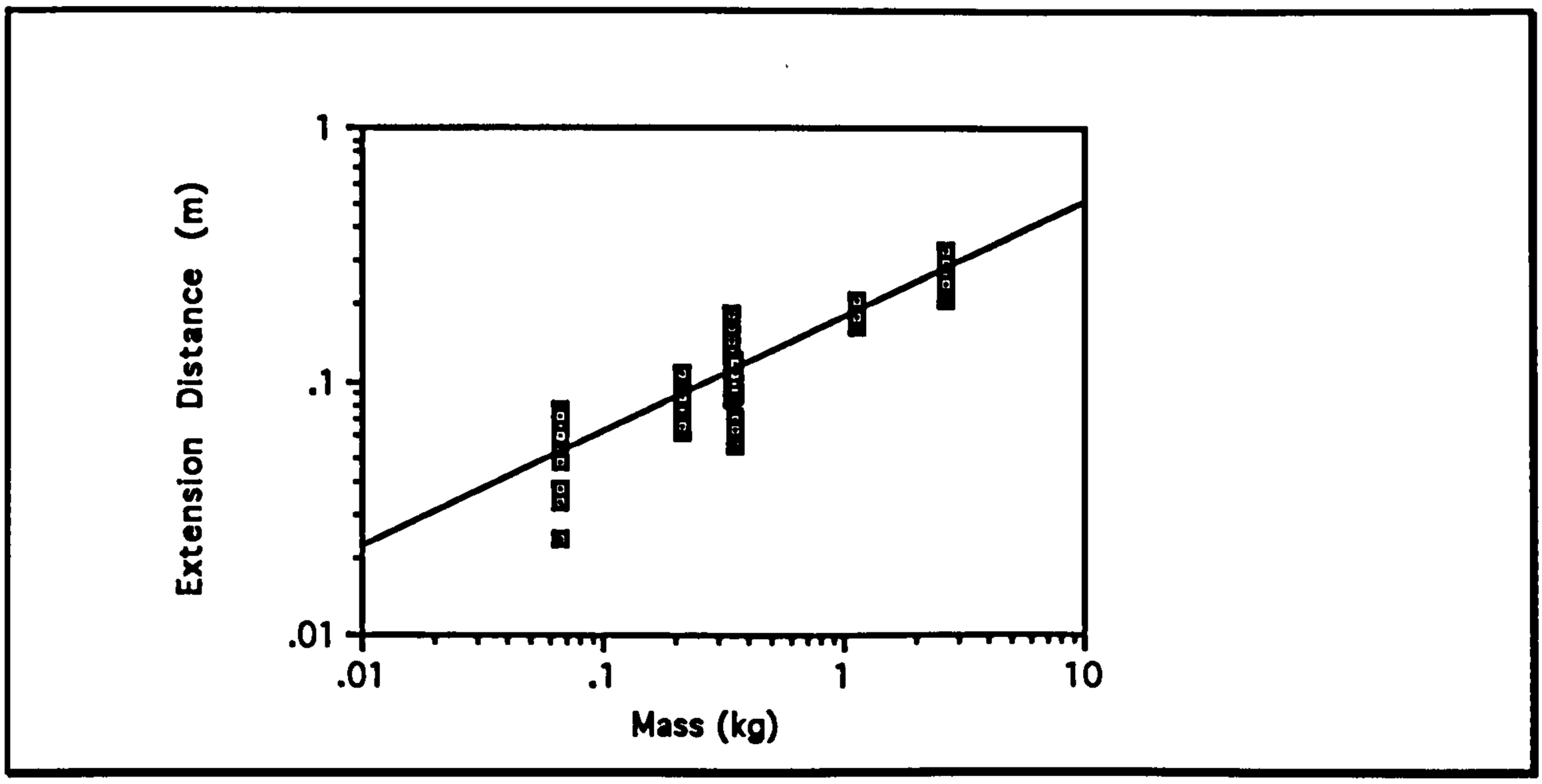
multiplied by the square root of the effective leap distance to reflect the power to the minus half relationship there.



This graph shows the relationship between the mass and the duration of the takeoff phase of the leap. The leap duration has been normalized for a 1 m leap. The best fit power is 0.329 and the r^2 value is 0.661



This graph shows the relationship between the mass and the peak force during the leap. The peak force has been normalized for a 1 m leap. The best fit power is 0.717 and the r^2 value is 0.829



This graph shows the relationship between the mass and the extension distance in the takeoff phase of the leap. The extension has been normalized for a 1 m leap. The best fit power is 0.451 and the r^2 value is 0.818

The results for the takeoff duration are very close to the 0.33 power predicted by the geometric similarity model. The peak force value of 0.72 is closer to the value predicted by elastic similarity. The extension distance value of 0.45 is rather larger than any of the predicted values, though is closest to the geometric model prediction.

It seems that the geometric model predicts the rates of change best. However it is interesting to note that both the force and the extension distance increase more rapidly with increasing mass than would be predicted. One of the features of all the scaling models is that if they are faithfully followed by animals of different sizes then all the animals will be able to leap exactly the same distance.

This can be justified as follows (based on Hill 1950):

The maximum force that can be generated by a muscle depends upon its cross-sectional area.

(18) $$F_{max} \propto d^2$$

Where:

d is a characteristic diameter

The extension distance depends upon the length of the hind limb.

(19) $$s \propto L$$

Where:

L is a characteristic length

Rearranging equation (21) from the previous chapter gives:

(20) $$r = \frac{2 F_{max} s}{mg}$$

So, inserting equations (18) and (19) and combining the constants gives:

(21) $$r \propto \frac{d^2 L}{m}$$

But, no matter what scaling model is being used, d^2L is always the volume, and the volume is always directly proportional to the mass, so r in this equation is constant irrespective of the mass or the scaling model.

So, if, as would seem likely, a larger animal wishes to be able to leap further than a smaller animal[32], then it needs to be more specialized than the smaller animal and have correspondingly longer limbs and stronger hind-limb muscles so that both F_{max} and s will increase faster with increasing body mass than simple scaling would predict. This is precisely what these results show.

[32]Home range size in primates in general actually correlates positively with group mass (Clutton-Brock and Harvey 1977). Except for *Lemur catta*, the animals in this study are solitary, so the individual mass is equivalent to the group mass.

Predictive Leaping Model

The previous three chapters have described results obtained from measuring the joint positions, frame by frame, in a number of video sequences of leaping. To check these results, over 100 graphs showing the position of the centre of mass of the animal against time were plotted. These showed sufficient similarity in their shape that it was decided that it might be possible to produce a general model for leaping based on the goal of producing a centre of mass trajectory that matched the ones measured. This would then be a predictive model that would produce dynamic joint position data from a set of static start conditions. This is an important objective since it would allow the analysis of the mechanics involved in activities that are rarely, or never seen. For example, leaping behaviour in fossil forms, or looking at very long leaps.

Theory

The model provides a method for calculating the hind-limb positions with time for an animal leaping a given distance. It is designed to be useful in situations where the dimensions of the animal are known, to calculate the maximum possible leaping distance, by calculating the power requirements, and the bone stresses.

It has been created by considering the design goals of a leaping animal. For efficiency, and also maximum performance, the animal should aim to move its centre of mass along a straight line path inclined at 45° during the takeoff phase of the leap. Any deviation from this path will incur an energetic cost, and will lead to a reduced maximum performance. In addition, the animal needs to maximize the duration of the takeoff phase to reduce the peak forces necessary to produce the required impulse since it is ultimately the impulse that the animal can apply to the

substrate that will determine the length of the leap. However, at the same time, the animal needs to convert the purely rotational motion produced by its muscles into a linear thrust, and this becomes progressively less efficient as the limb is extended. Internal energy should also be minimized by requiring smooth movements of all the parts of the mechanism.

The model incorporates the effects of change in overall centre of mass with the change in limb position, but assumes a uniform extension for the hind-limb joints, and completely ignores the rôle of the tail, the upper limbs and torso bending.[33]

The leap model is considered to be driven entirely by extension of the hind-limb, and requires, as input parameters, the start position of the jump with the hind-limb fully flexed, and an equation describing the position of the centre of mass with respect to time. For the simplest approach, a constant force formula is used, but the model could have been adapted for other equations, and could, indeed, have used force plate data had this been available. It is also assumed that the extension of the hind-limb occurs uniformly at each joint, and that at takeoff, the limb is approaching full extension.

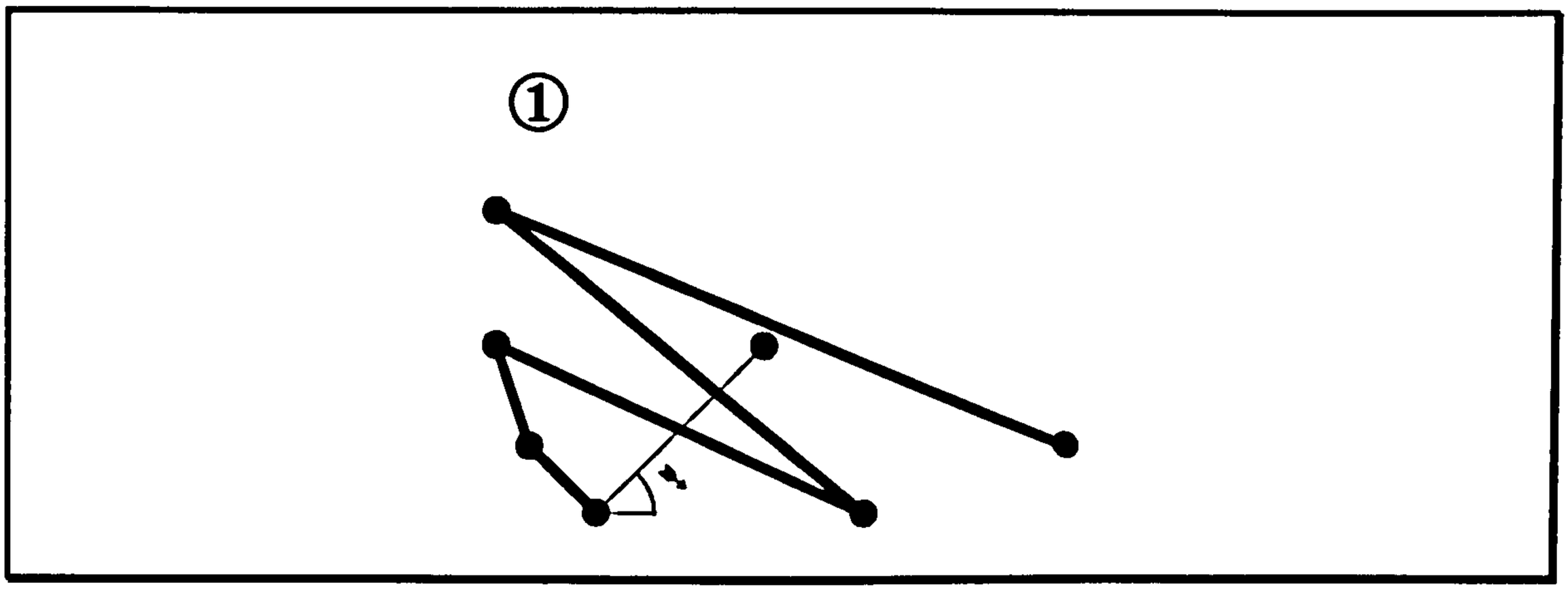
The model can best be described by following through the steps used to calculate the joint positions at any time during the takeoff phase:

The hind-limb and torso of the animal are represented as a set of five segments linked by rotational joints. First of all, from the start position

---

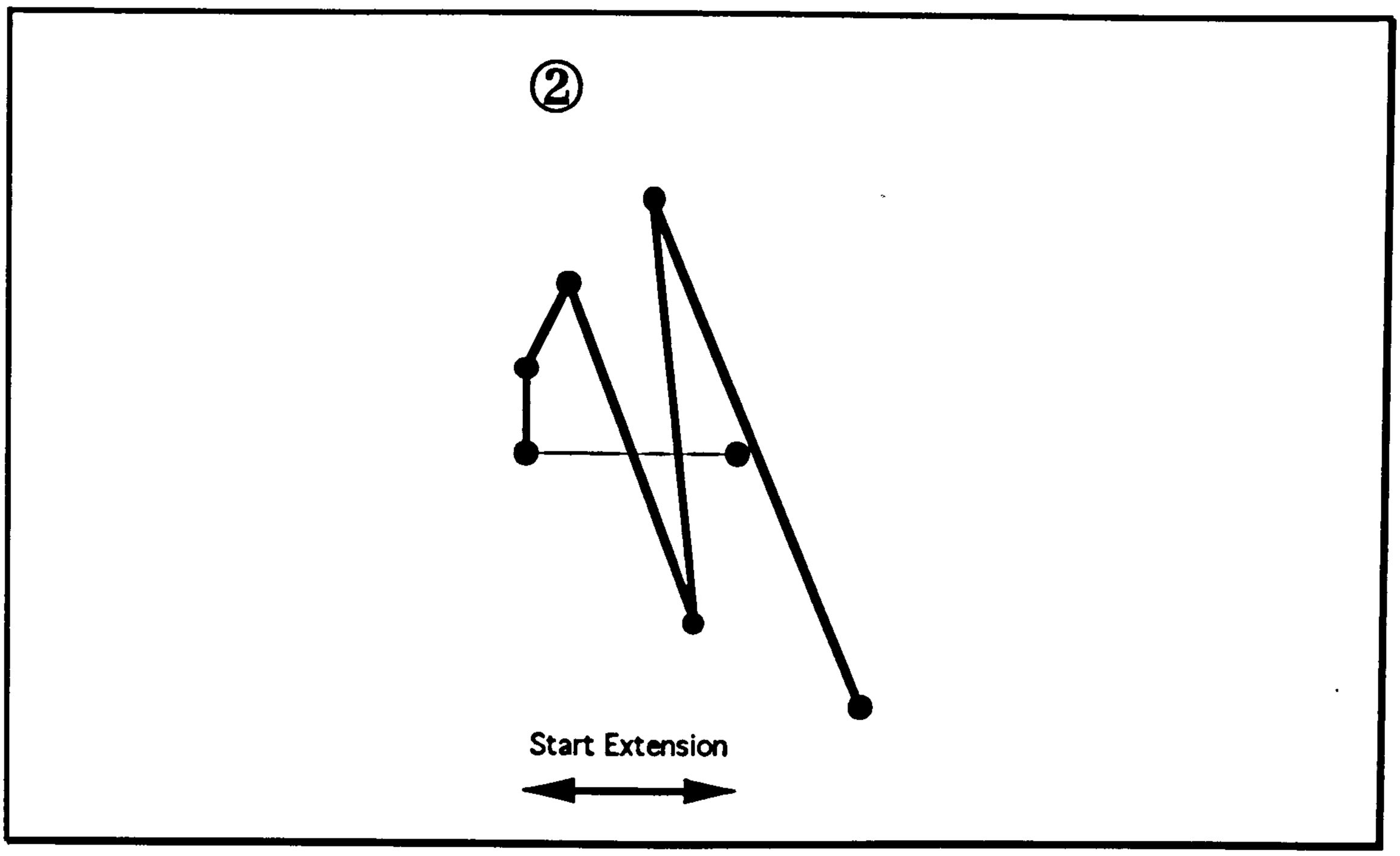
[33]Given the goals described, there are a number of different ways that the problem can be solved. The solution given here was obtained by considering how the movements of the limbs would effect the centre of mass and applying an iterative process to obtain the correct movement. A simpler, semi-analytical approach was tried initially, but this required unrealistic start conditions so had to be discarded.

and from the mass properties of the segments (torso, hip, calf, fore-foot and hind-foot), the overall centre of mass is calculated.



Stage one of the model calculation involves the measurement of the centre of mass position from the joint locations and the centres of mass of each segment at the start position.

The whole structure is then rotated so that the centre of mass is positioned on the x axis. This is used as a local coordinate system, with the centre of mass moving along the x axis. The results can be converted back to the global coordinates as a final step simply by rotating the model by 45°. The toe-tip is already assumed to be positioned at the origin. Next, the distance of the centre of mass from the origin is measured in this position, and in the end position, with all the joints fully extended. This gives the maximum possible extension distance available for the takeoff. The fraction of this distance to use for the modelling run is set as one of the input parameters.

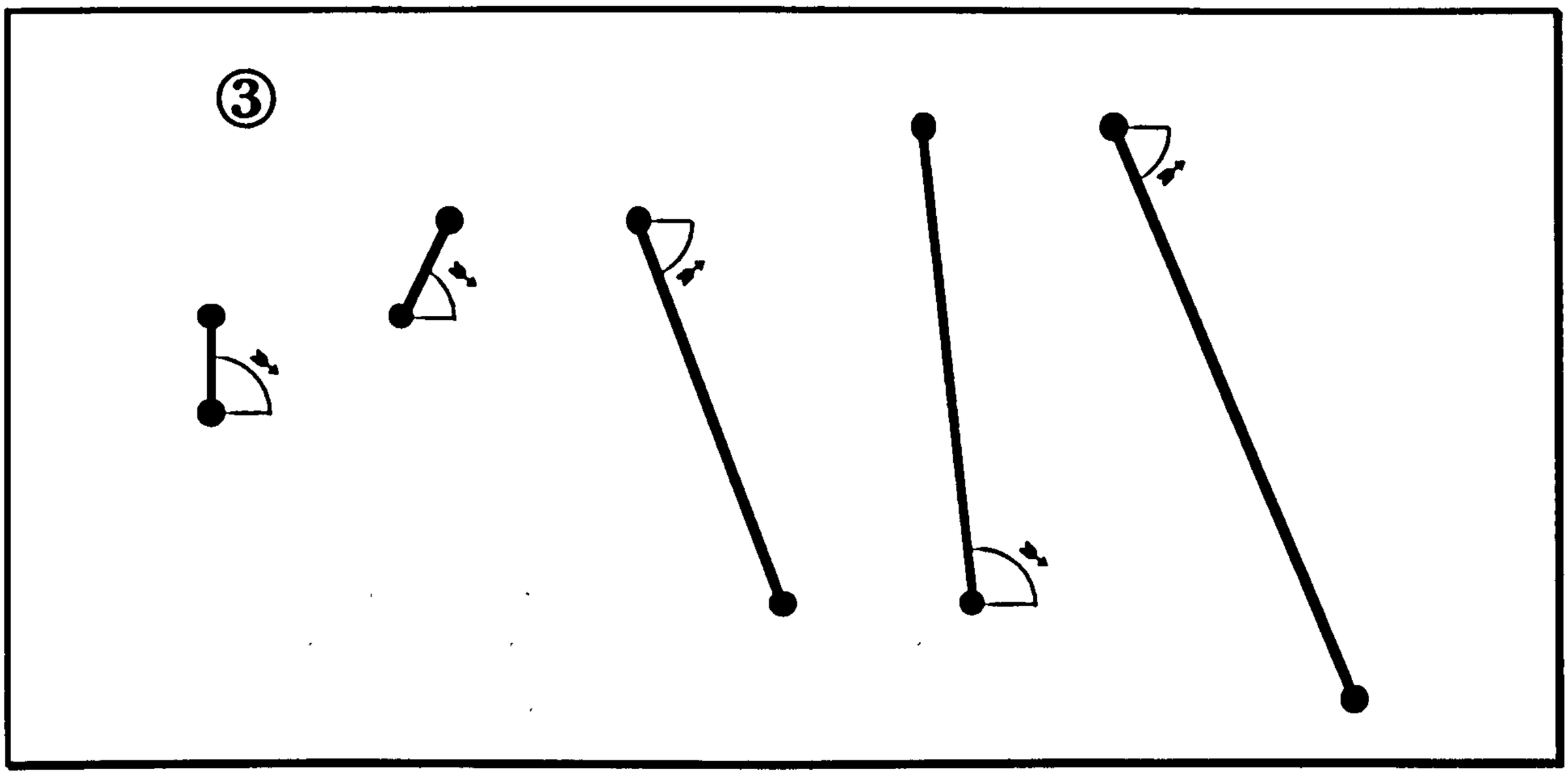


The structure is rotated so that the line joining the position of the toe tip (0,0) is on the X axis. The start extension distance is the distance of the centre of mass from the toe tip.

From the extension distance, and from the equation used to describe the force function, the duration of the takeoff phase is calculated. With this, and using the number of output times specified in the modelling parameters, a table containing a list of times of interest is built up: these are the times for which the program needs to find the positions of the joints. Again using the force equation, the distance of the centre of mass from the origin is calculated at each of these times.

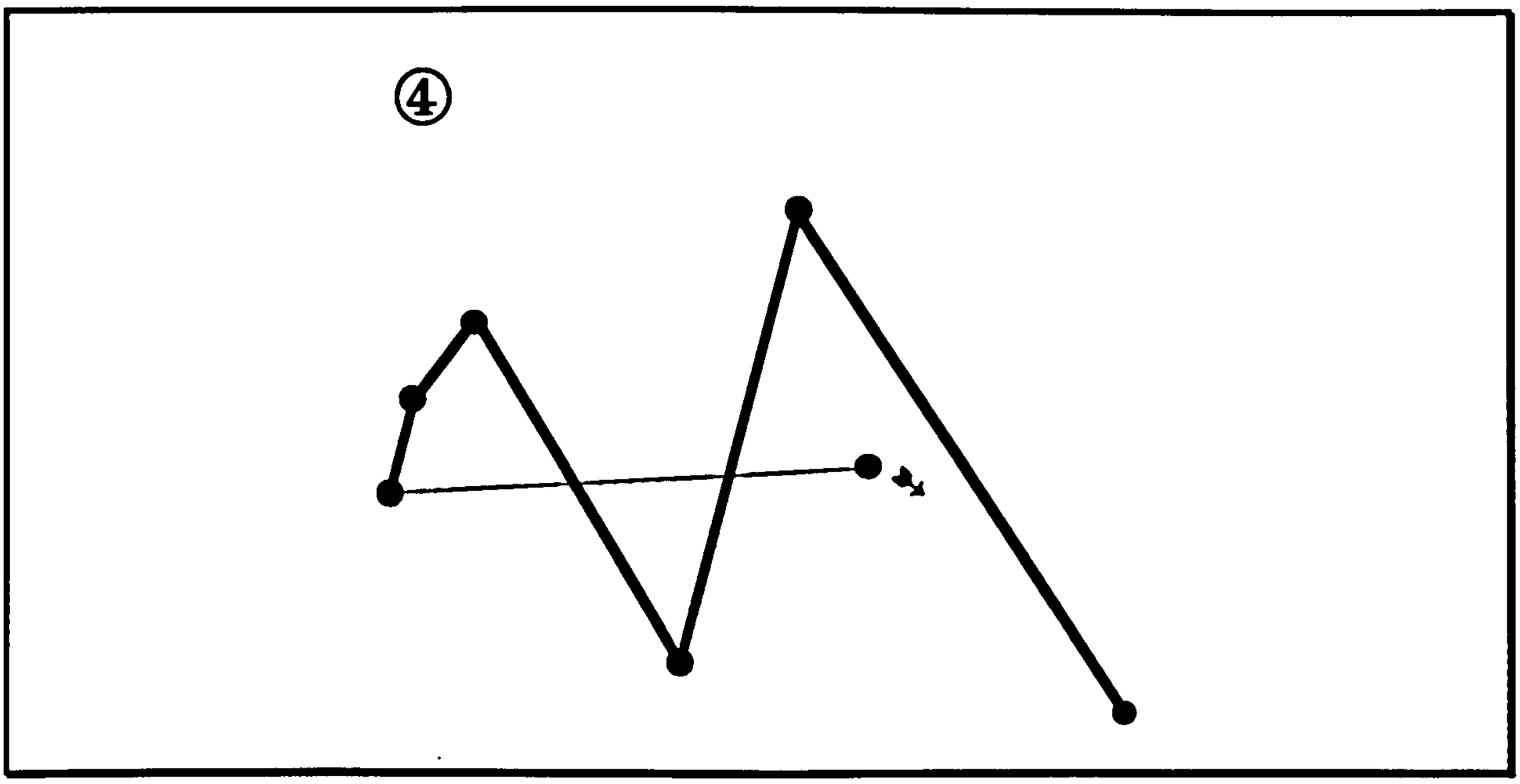
However, the positions of the joints cannot now be analytically calculated. There is no unique solution to the placement of the limbs since there are only two known points - the centre of mass position and the toe tip position - and four joints and five links between these points (Hunt 1978). This can be overcome by using the design goal requiring smooth movement of the segments. Each segment is rotated from its starting

angle to the zero finishing angle at a rate proportional to its initial angle so that they all finish together.



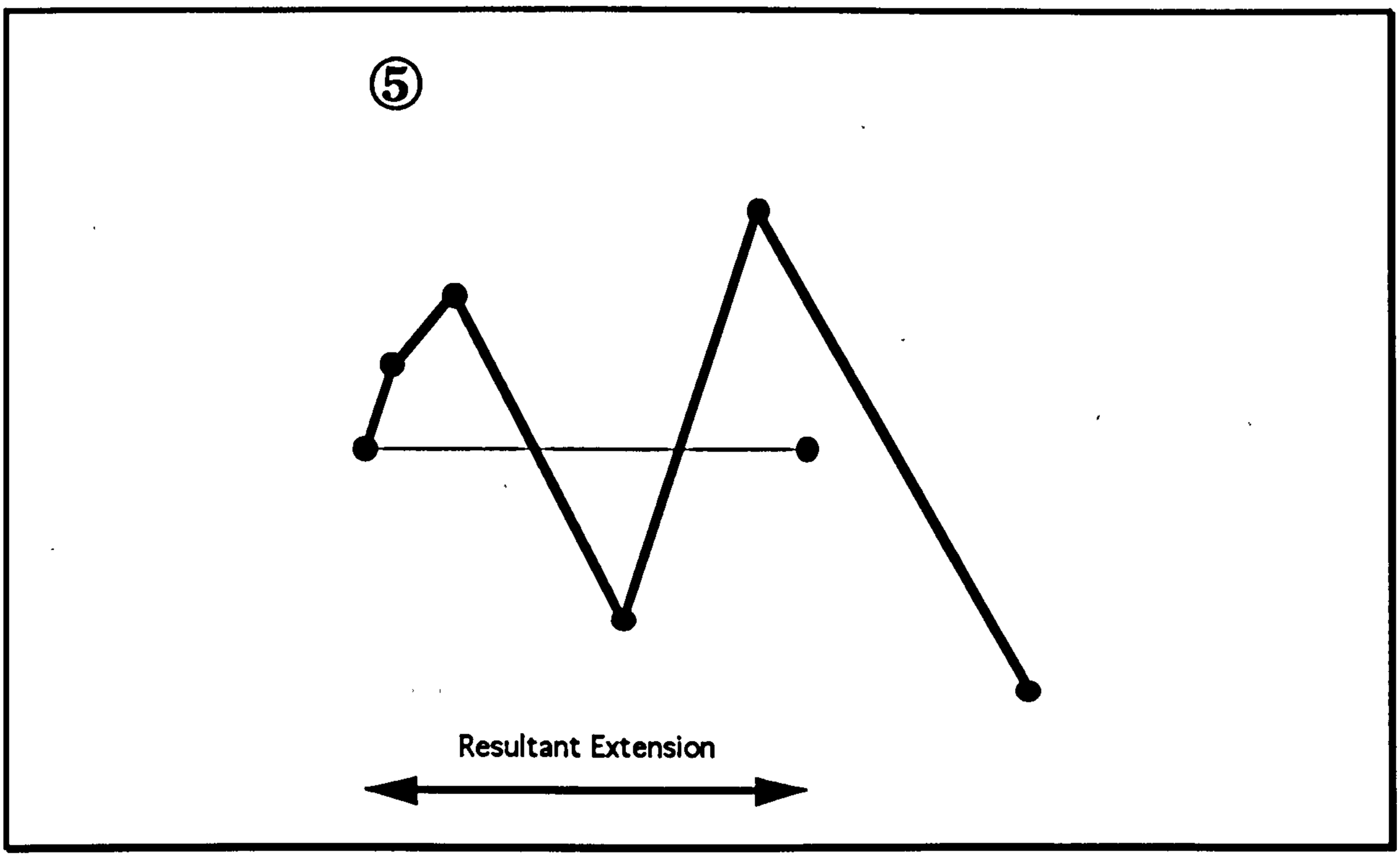
The angles from the X axis to the segment angles for each separate segment are calculated. By rotating the segments at a speed proportional to this angle, all the segments end up lined up together along the X axis together: a fully extended position similar to the takeoff position seen with leaping prosimians.

This needs to be adjusted at each step by recalculating the position of the centre of mass and rotating the structure to reposition it on the local x axis.



At each intermediate position calculated by extending the joints, there may be a discrepancy between the new centre of mass position and the X axis. This needs to be corrected for by rotating the structure again.

The exact amount of extension and rotation required to produce the desired distance is found by repeatedly performing the calculation with different values and gradually improving the precision of the result. The algorithm used converges relatively rapidly. Full details of this process are given in the technical development section in the second part of this thesis.



After the extension and rotation has been performed, the new position of the centre of mass can be measured. This is unlikely to be the required value the first time, so depending upon whether it is too large or too small, the whole process is repeated with a smaller or larger amount of rotation. This cycle is continued until the difference between the obtained extension distance and the desired extension distance is sufficiently small.

Once the predictive modelling program has produced a list of joint positions at a set of times during the takeoff phase of the leap at an arbitrary temporal resolution and precision, this information can be fed into the gait analysis program as if it had been measured from an actual animal, and the inverse dynamic analysis can be performed to calculate the forces and torques that are present during the movement. The problems of double differentiation leading to the magnification of high frequency errors is minimized by choosing a reasonable compromise between precision and sample rate. If the sample rate is very high, then the errors due to the numerical nature of the solution and the rounding error within the computer can still lead to the actual values being

swamped. 30 to 100 samples at a time tolerance of 10^{-7} produces results with barely noticeable noise in the acceleration curves.

Results

Runs were performed using the mass characteristics for each of the test animals, but with the figures for the torso made up of the original torso value plus the values for the head, tail and forearms, so that the total mass of the animal was unchanged. The following table shows the figures actually used:

		M. murinus	*L. catta*	*C. major*	*M. coquerelli*	*G. garnettii*	*G. moholi*
Forefoot	**Mass** (kg)	$8.75\ 10^{-4}$	$3.78\ 10^{-2}$	$4.76\ 10^{-3}$	$4.90\ 10^{-3}$	$2.25\ 10^{-2}$	$4.20\ 10^{-3}$
	CM	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$	$5.73\ 10^{-1}$
	MOI ($kg.m^2$)	$1.26\ 10^{-6}$	$5.14\ 10^{-6}$	$1.96\ 10^{-7}$	$2.18\ 10^{-7}$	$1.97\ 10^{-6}$	$2.07\ 10^{-7}$
Hind-foot	**Mass** (kg)	$8.75\ 10^{-4}$	$3.78\ 10^{-2}$	$4.76\ 10^{-3}$	$4.90\ 10^{-3}$	$1.13\ 10^{-2}$	$2.52\ 10^{-3}$
	CM	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$	$4.78\ 10^{-1}$
	MOI ($kg.m^2$)	$1.32\ 10^{-8}$	$4.94\ 10^{-6}$	$2.17\ 10^{-7}$	$2.26\ 10^{-7}$	$1.41\ 10^{-6}$	$1.35\ 10^{-7}$
Calf	**Mass** (kg)	$3.38\ 10^{-3}$	$1.46\ 10^{-1}$	$1.83\ 10^{-2}$	$1.89\ 10^{-2}$	$6.08\ 10^{-2}$	$1.39\ 10^{-2}$
	CM	$4.01\ 10^{-1}$	$4.01\ 10^{-1}$	$4.01\ 10^{-1}$	$4.01\ 10^{-1}$	$6.08\ 10^{-1}$	$6.08\ 10^{-1}$
	MOI ($kg.m^2$)	$3.57\ 10^{-7}$	$1.61\ 10^{-4}$	$6.86\ 10^{-6}$	$5.39\ 10^{-6}$	$4.08\ 10^{-5}$	$4.15\ 10^{-6}$
Thigh	**Mass** (kg)	$1.03\ 10^{-2}$	$4.43\ 10^{-1}$	$5.57\ 10^{-2}$	$5.74\ 10^{-2}$	$1.37\ 10^{-1}$	$4.20\ 10^{-2}$
	CM	$4.47\ 10^{-1}$	$4.47\ 10^{-1}$	$4.47\ 10^{-1}$	$4.47\ 10^{-1}$	$5.61\ 10^{-1}$	$5.61\ 10^{-1}$
	MOI ($kg.m^2$)	$1.26\ 10^{-6}$	$7.39\ 10^{-4}$	$2.46\ 10^{-5}$	$2.83\ 10^{-5}$	$1.48\ 10^{-4}$	$1.73\ 10^{-5}$
Torso	**Mass** (kg)	$4.71\ 10^{-2}$	$2.04\ 10^{+0}$	$2.56\ 10^{-1}$	$2.64\ 10^{-1}$	$8.94\ 10^{-1}$	$1.47\ 10^{-1}$
	CM	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$	$5.00\ 10^{-1}$
	MOI ($kg.m^2$)	$1.82\ 10^{-5}$	$9.58\ 10^{-3}$	$3.05\ 10^{-4}$	$3.06\ 10^{-4}$	$1.94\ 10^{-3}$	$1.12\ 10^{-4}$

The start position was obtained by examining one of the longest leaps for each species, and using the position where the animal's hind-limb was

maximally flexed. Because of smoothing, this was the mean position from three consecutive frames where the animal was stationary. This was considered to be a good estimate of a typical leap start posture. The start positions used are given in the following table:

		Toe Tip	Mid-Tarsal Joint	Ankle	Knee	Hip	Nose Tip	Body COM
M. murinus	x	$0.00\ 10^{+0}$	$-9.62\ 10^{-3}$	$-1.99\ 10^{-2}$	$-5.12\ 10^{-4}$	$-4.52\ 10^{-2}$	$6.77\ 10^{-2}$	$-2.26\ 10^{-3}$
	y	$0.00\ 10^{+0}$	$2.66\ 10^{-3}$	$5.72\ 10^{-3}$	$3.47\ 10^{-2}$	$1.78\ 10^{-2}$	$8.22\ 10^{-3}$	$1.98\ 10^{-2}$
L. catta	x	$0.00\ 10^{+0}$	$-3.16\ 10^{-2}$	$-6.07\ 10^{-2}$	$1.72\ 10^{-2}$	$-1.01\ 10^{-1}$	$2.49\ 10^{-1}$	$2.13\ 10^{-2}$
	y	$0.00\ 10^{+0}$	$4.88\ 10^{-3}$	$2.01\ 10^{-2}$	$1.05\ 10^{-1}$	$8.51\ 10^{-2}$	$5.17\ 10^{-2}$	$6.95\ 10^{-2}$
C. major	x	$0.00\ 10^{+0}$	$-1.55\ 10^{-2}$	$-3.07\ 10^{-2}$	$1.77\ 10^{-2}$	$-5.38\ 10^{-2}$	$1.01\ 10^{-1}$	$4.72\ 10^{-3}$
	y	$0.00\ 10^{+0}$	$1.02\ 10^{-2}$	$2.08\ 10^{-2}$	$7.00\ 10^{-2}$	$3.46\ 10^{-2}$	$1.02\ 10^{-2}$	$3.41\ 10^{-2}$
M. coquereli	x	$0.00\ 10^{+0}$	$-2.25\ 10^{-2}$	$-4.00\ 10^{-2}$	$1.98\ 10^{-2}$	$-7.28\ 10^{-2}$	$1.04\ 10^{-1}$	$-5.40\ 10^{-3}$
	y	$0.00\ 10^{+0}$	$6.79\ 10^{-3}$	$1.64\ 10^{-2}$	$4.97\ 10^{-2}$	$5.60\ 10^{-2}$	$4.71\ 10^{-3}$	$3.78\ 10^{-2}$
G. garnettii	x	$0.00\ 10^{+0}$	$-8.96\ 10^{-3}$	$-4.46\ 10^{-2}$	$2.99\ 10^{-2}$	$-6.41\ 10^{-2}$	$1.82\ 10^{-1}$	$2.16\ 10^{-2}$
	y	$0.00\ 10^{+0}$	$2.41\ 10^{-2}$	$4.49\ 10^{-2}$	$9.57\ 10^{-2}$	$9.02\ 10^{-2}$	$3.03\ 10^{-2}$	$6.60\ 10^{-2}$
G. moholi	x	$0.00\ 10^{+0}$	$-2.35\ 10^{-2}$	$-3.75\ 10^{-2}$	$2.71\ 10^{-2}$	$-5.20\ 10^{-2}$	$8.52\ 10^{-2}$	$1.99\ 10^{-3}$
	y	$0.00\ 10^{+0}$	$8.51\ 10^{-3}$	$2.67\ 10^{-2}$	$4.29\ 10^{-2}$	$5.80\ 10^{-2}$	$6.02\ 10^{-2}$	$5.27\ 10^{-2}$

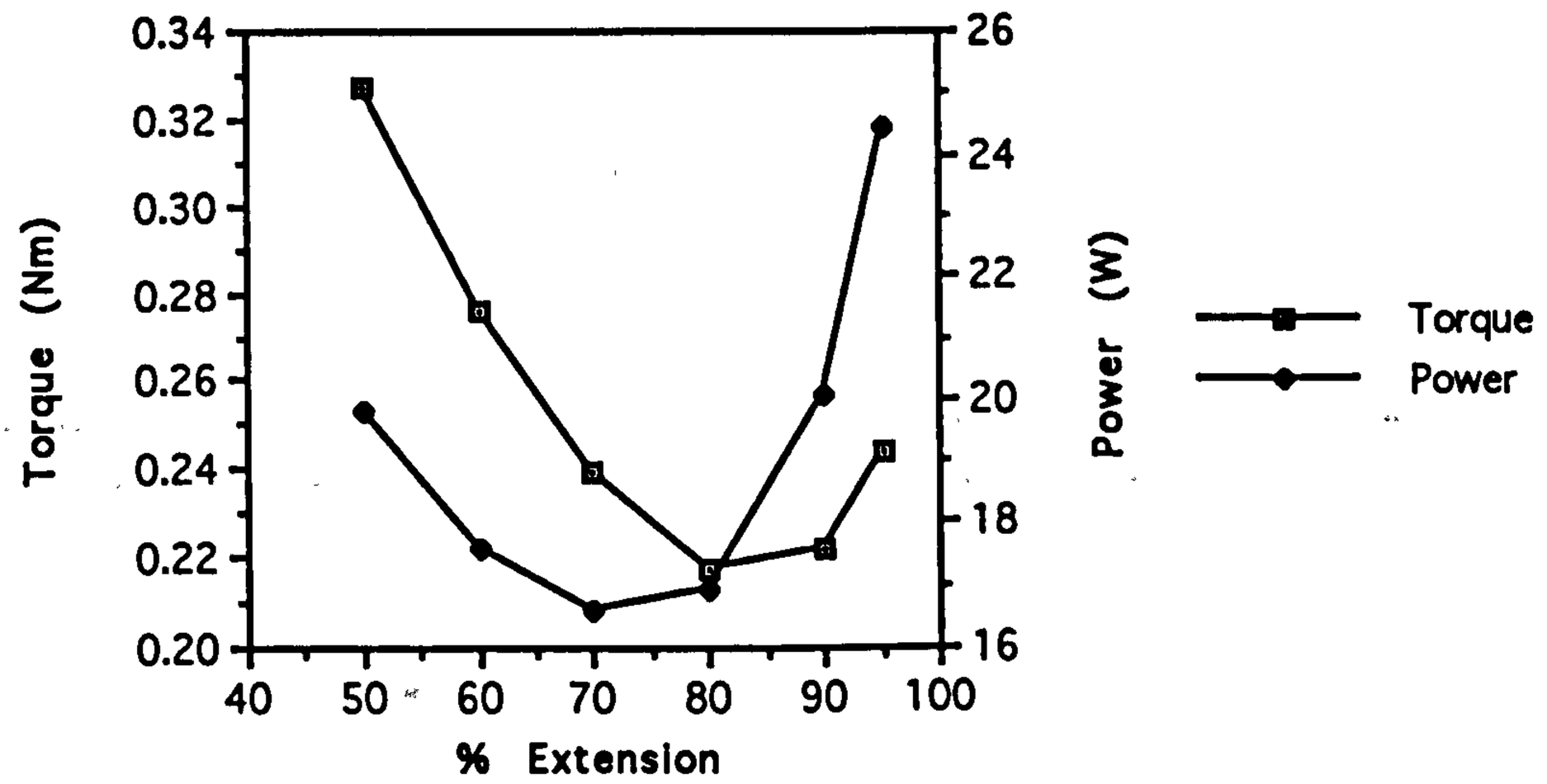
The other interesting input parameters for the model are the extension fraction of the hind-limb and the distance jumped. The extension fractions used were: 50%, 60%, 70%, 80%, 90%, 95%[34]; the distances were: 1 m, 2 m, 4 m and 8 m. It was soon clear that the effect of distance is a simple scale factor (see following graphs), so distance effects are only shown for a single extension fraction, and extension fraction effects only for a single distance.

[34]99% extension fraction was also tried at first, but the torque and power required for this last few percent totally swamped the other values since they all approach 100% extension asymptotically.

Extension Effects

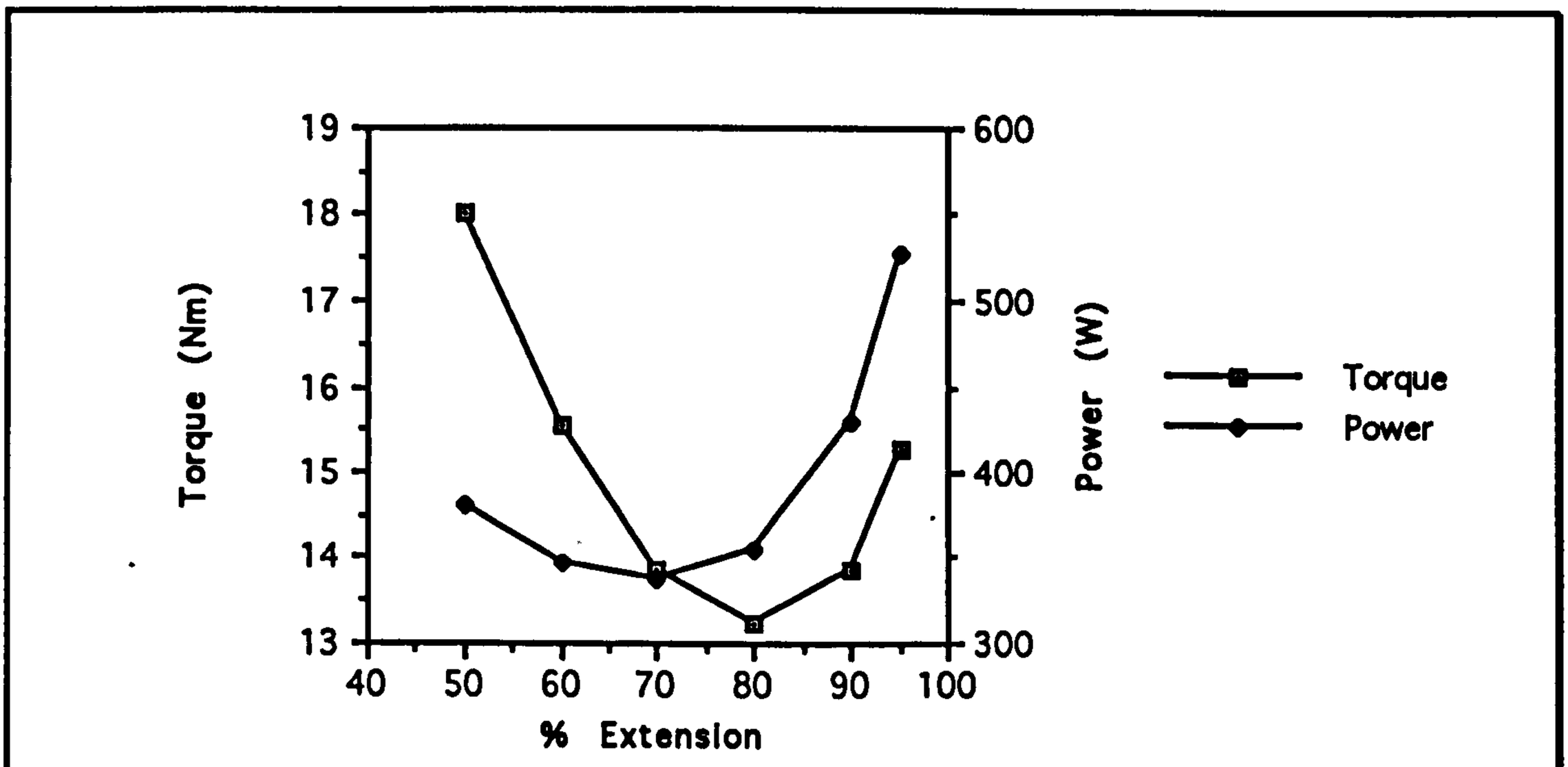
The following graphs show the effects of fractional hind-limb extension the model for all the subject animals:

Microcebus murinus



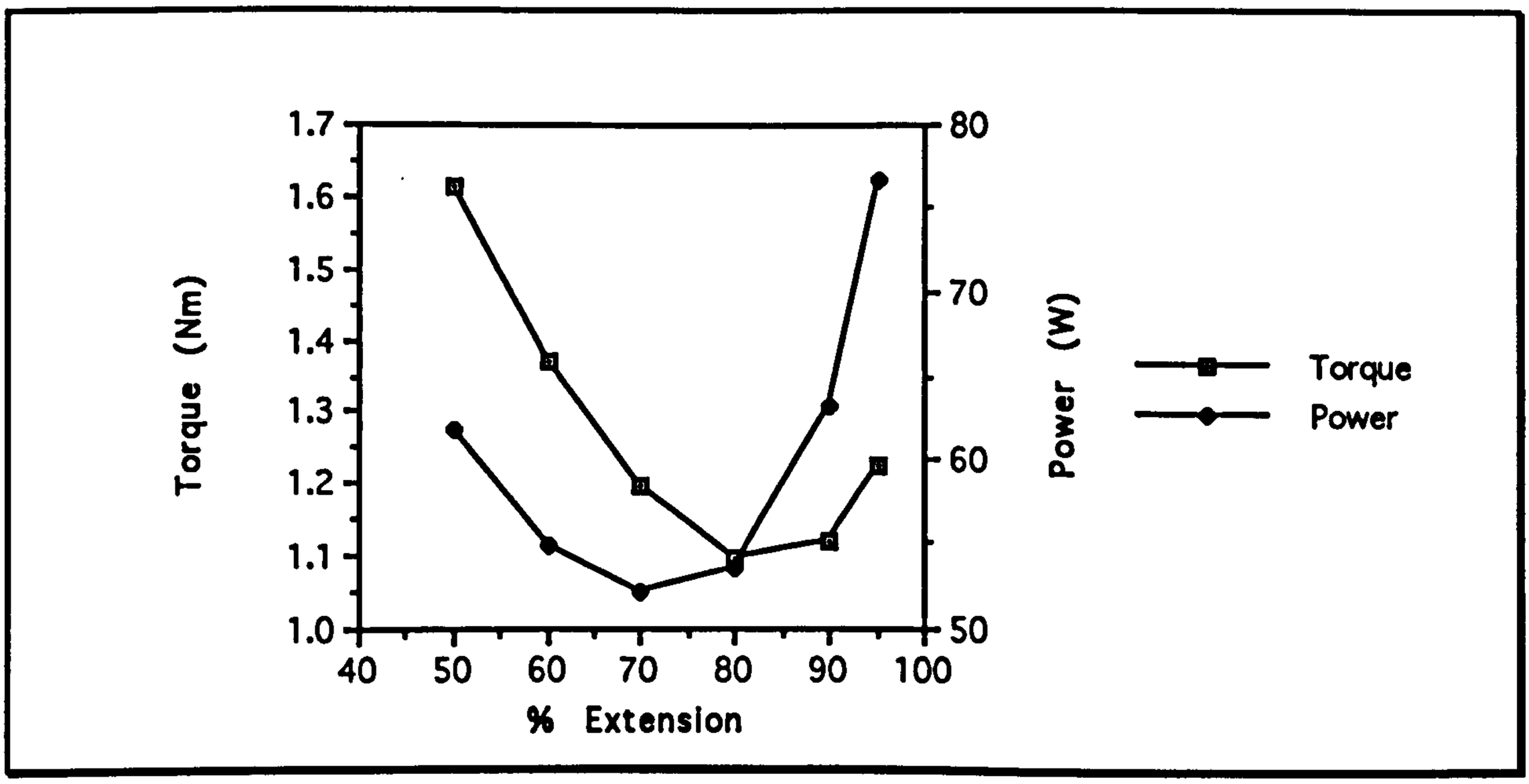
Graph showing the effect of hind-limb extension on the peak torque and the peak power generated about the hip joint for *Microcebus murinus* in a simulated 1 m leap at a 45° trajectory.

Lemur catta



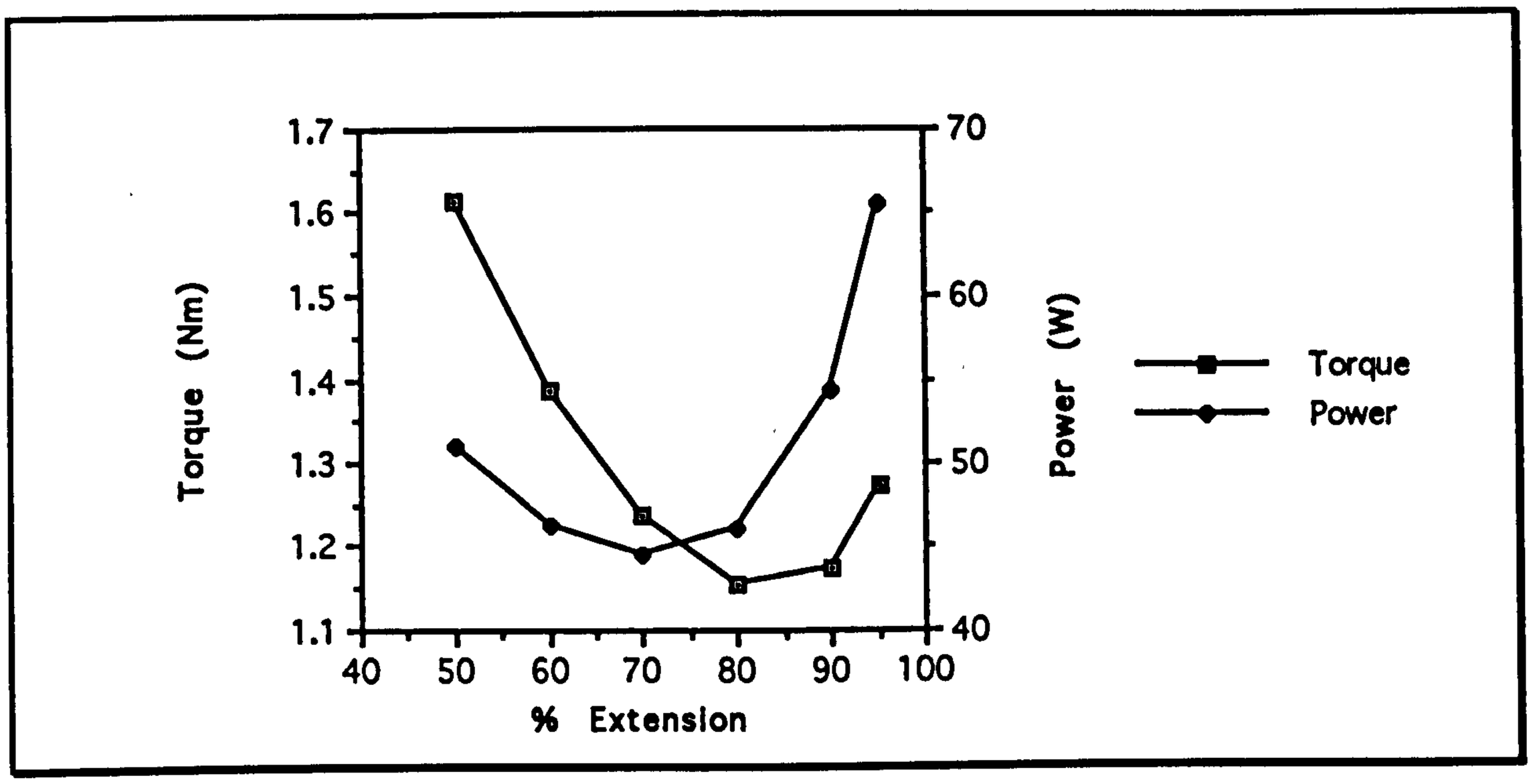
Graph showing the effect of hind-limb extension on the peak torque and the peak power generated about the hip joint for *Lemur catta* in a simulated 1 m leap at a 45° trajectory.

Cheirogaleus major



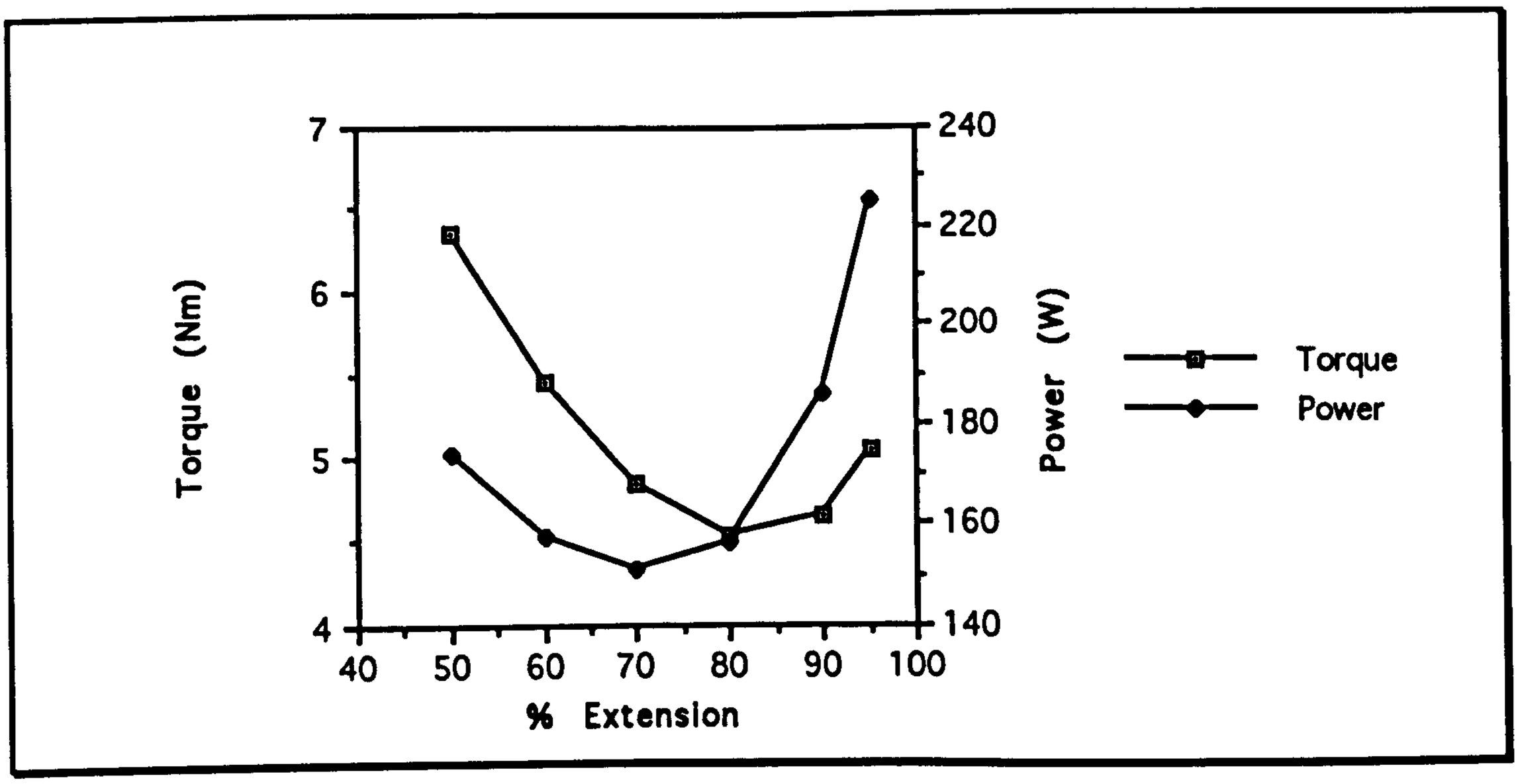
Graph showing the effect of hind-limb extension on the peak torque and the peak power generated about the hip joint for *Cheirogaleus major* in a simulated 1 m leap at a 45° trajectory.

Mirza coquereli



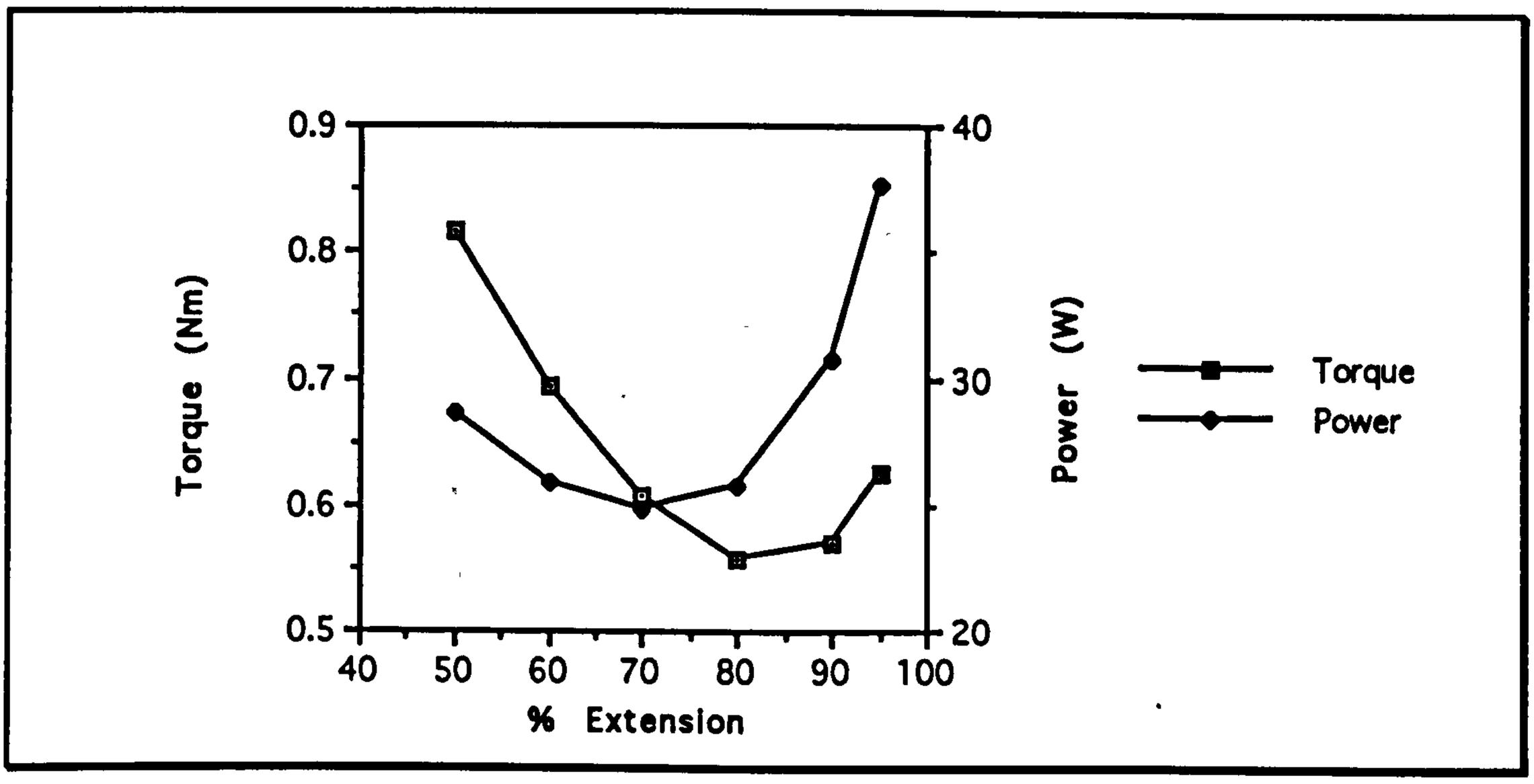
Graph showing the effect of hind-limb extension on the peak torque and the peak power generated about the hip joint for *Mirza coquereli* in a simulated 1 m leap at a 45° trajectory.

Galago gamettii

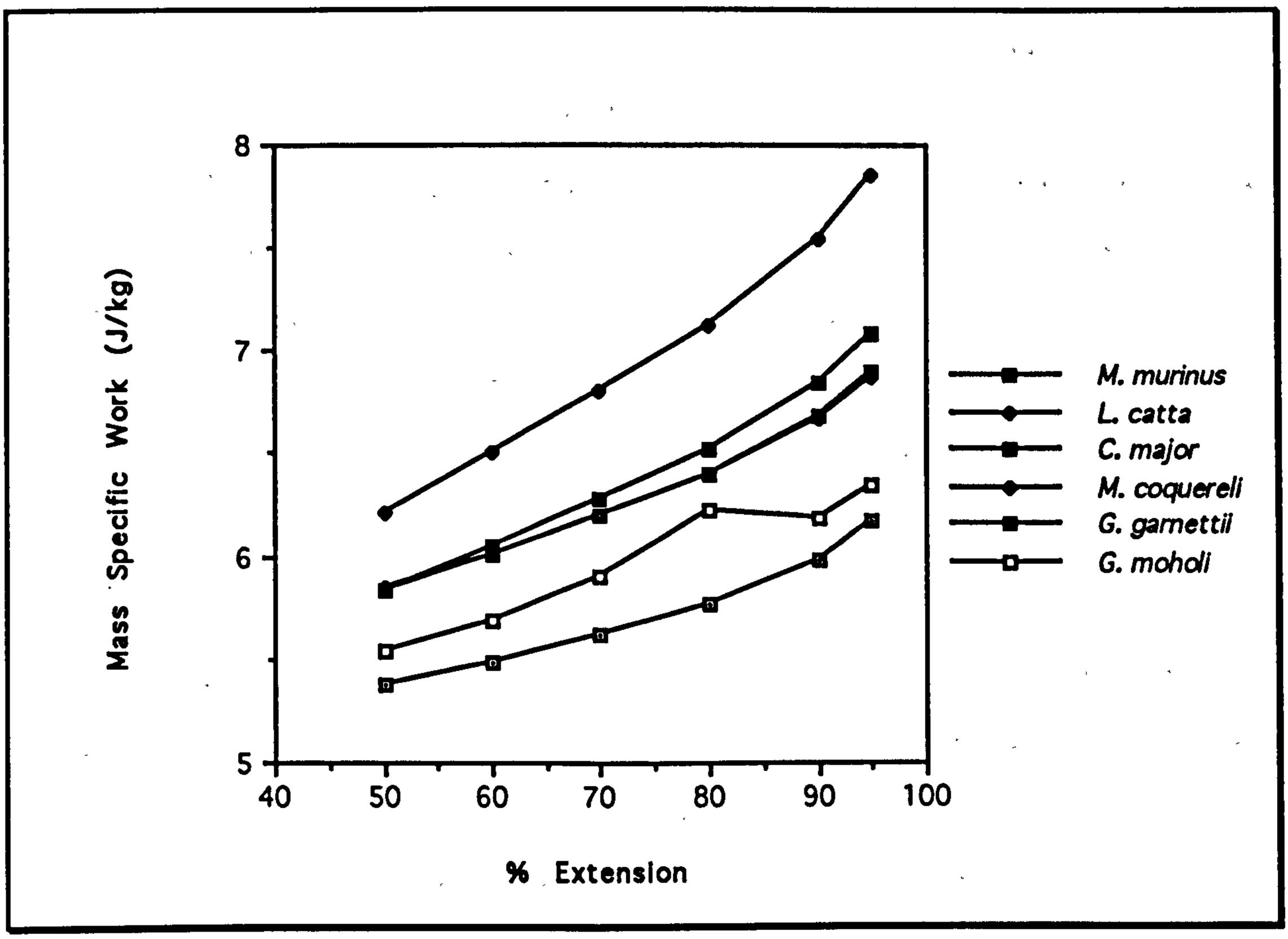


Graph showing the effect of hind-limb extension on the peak torque and the peak power generated about the hip joint for *Galago garnettii* in a simulated 1 m leap at a 45° trajectory.

Galago moholi



Graph showing the effect of hind-limb extension on the peak torque and the peak power generated about the hip joint for *Galago moholi* in a simulated 1 m leap at a 45° trajectory.



Graph showing the effect of extension fraction on the mass specific work done during the leap. In general there is a small increase with extension fraction, and it approaches 100% asymptotically. *Galago moholi* has a distinct shape, with its dip from 80% to 90% extension.

Varying the extension fraction from 50% to 95% causes a 50% change in both the peak torque and the peak power required for a given leap. It is thus an important parameter in the model. The results for all the animals indicate that the peak torque is minimized with an extension fraction of 80%, but that the peak power is lowest at 70%. The total work curve is lowest at low extension fractions (50% in this set of results), but is very flat until 95% or greater extension fractions. This may seem to contradict the earlier statement that leaping animals need to maximize their takeoff distance in order to reduce the forces required to leap. However, what is actually happening is that during the last 20% of the leap, the animal needs to move the segments of its hind limb through relatively large

angles to produce any of the desired longitudinal movement[35] and this large rotational movement means that the fairly small moments of inertia of the hind limb start to become important. The energy required simply to rotate the limb becomes prohibitive. It also means that the energy that needs to be absorbed to stop the limb from rotating may become quite large. This probably does not involve any work on the part of the animal since it can be performed by the passive structures such as ligaments that limit extension in the limb, but if too much energy has to be absorbed in too short a time then the animal is risking injury.

The work/extension fraction curve for *Galago moholi* is slightly different from all the others: the value for total work drops from 80% to 90% extension. This is due to the combined effects of positive and negative work about the hind-limb joints. To produce the figure for the total amount of work done in the leap, I have summed together the values for the work done at each joint. In general, for all the animals, positive work is done at hip, ankle and mid-tarsal joints, and negative work is done at the knee joint and at the contact point. However, for *Galago moholi*, negative work is only done at the knee joint for the larger extension distances, and it is this increase in the negative work at the hip-joint that produces the total work reduction. The handling of negative work itself is problematic. It can only be cancelled out by positive work elsewhere in the system only if the there is some sort of energy transfer mechanism. In the case of the hind-limb, such a rôle could be postulated for the major two-joint muscles such as the *gastrocnemius* muscle and the hamstrings

---

[35]The longitudinal movement depends on the cosine of the current angle of the segment. from the direction that the animal wishes to go in. When almost fully extended, this angle is small, and for small angles, the cosine curve is very flat: a large change in angle is required for a relatively small change in longitudinal movement.

(Wells 1988). Otherwise, the negative work can be performed by muscles doing positive work. Unfortunately, the metabolic efficiency for negative work is quite different from the efficiency for positive work. Example figures are 0.22 for positive work (Dickinson 1929) and -1.6 for negative work (Abbott and Bigland 1953). Thus, some sort of scaling is required for calculating a total energy figure where negative work is considered to be important.

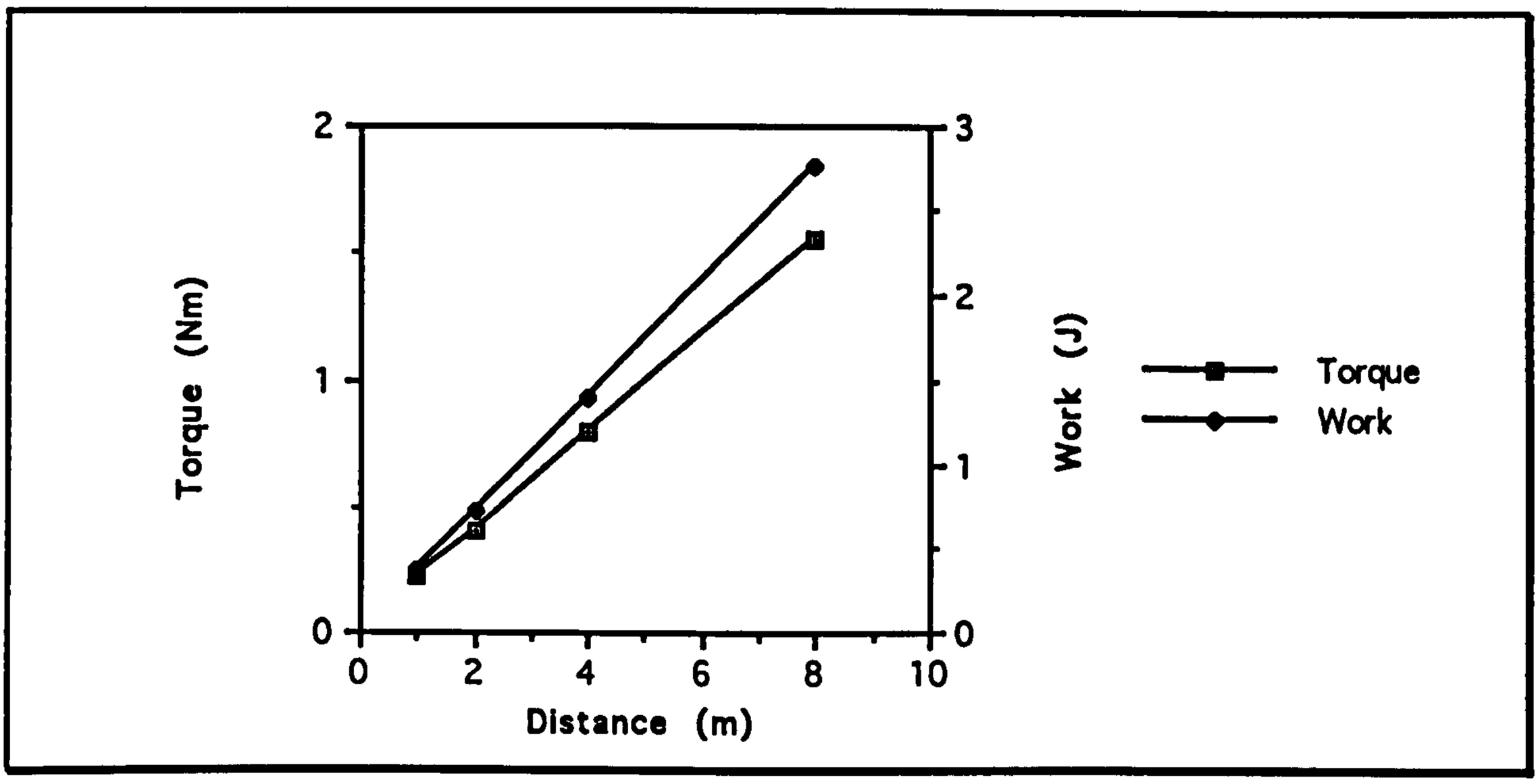
The curves for peak power, peak torque and total work all have different shapes, with both power and torque having minima at different values. If the relative importance of these factors is known, then an optimal extension distance can be calculated. This would require knowledge of the forces and powers produced by the major muscles in the leg - information which is not available at present. A value of 80% for optimal extension distance was chosen largely by inspection. It is the minimum value for peak torque, and both the power and work curves are relatively flat. An equally good argument could doubtless be made for 75% or even 70% extension, but because the curves are relatively flat, this would not greatly affect the rest of the analysis.

The shapes of the peak torque and power curves are very similar for all the animals. This is probably due to the geometrical approach used to obtain the mass properties of the limb segments.

Distance Effects

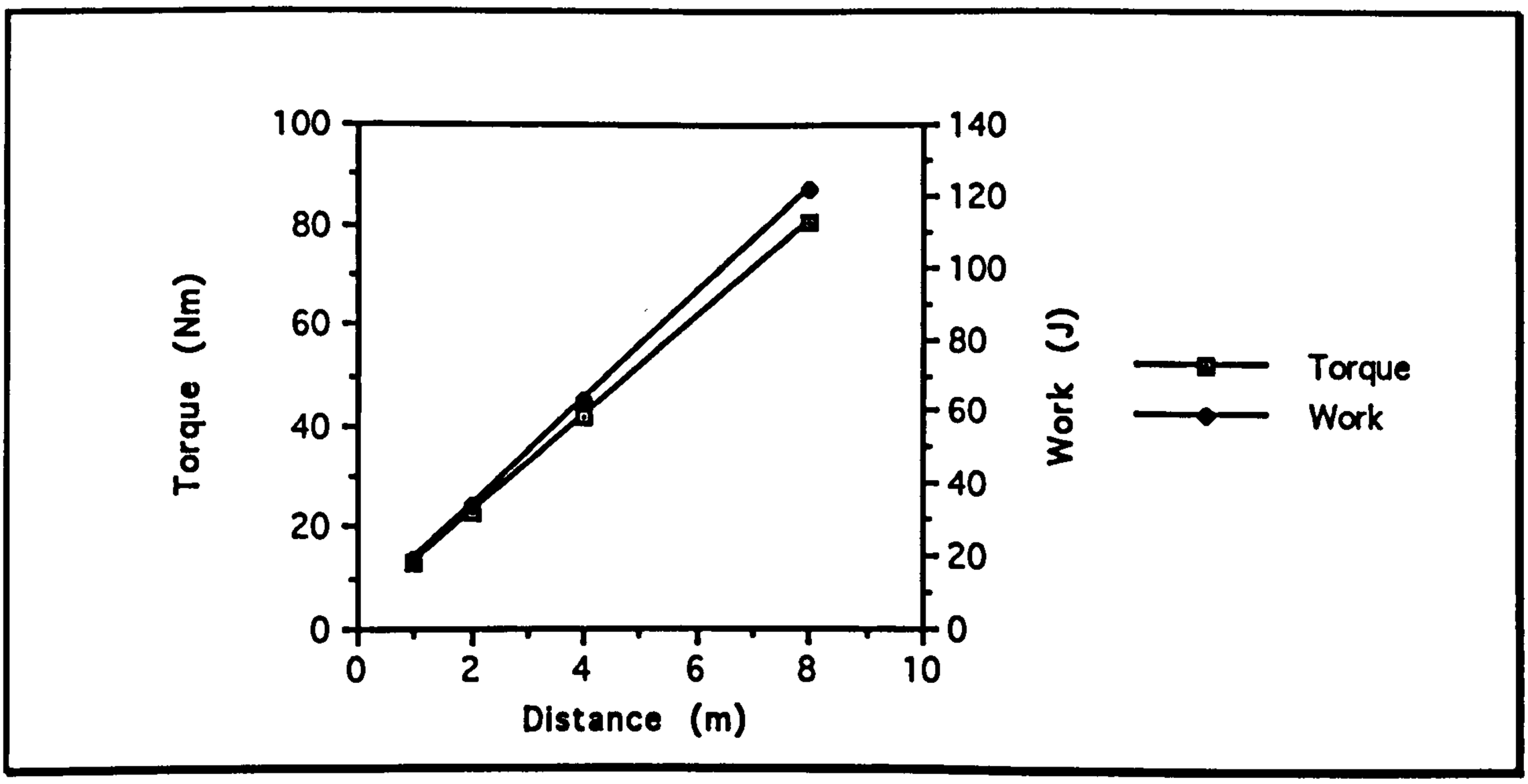
The next set of graphs show the effect of changing the leap distance. As can be seen from the previous graphs, the lowest peak torque occurs when the extension fraction is 80%. Since the aim is to model maximum leaps, where peak torque required is likely to be the limiting factor, this extension fraction has been used for all the subsequent analysis.

Microcebus murinus



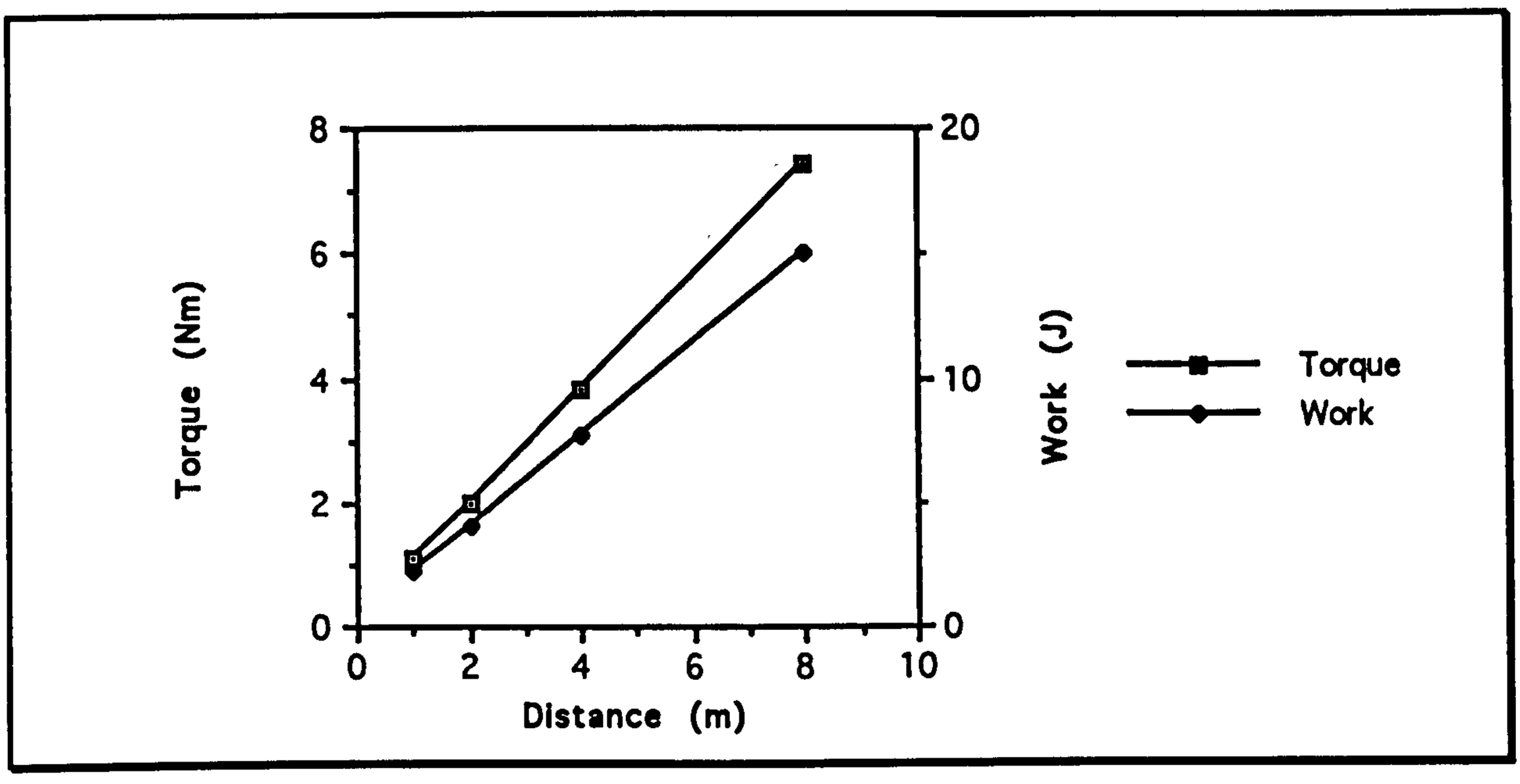
Graph showing the relationship between peak torque about the hip joint, the total work and the leap distance for a simulated leap of *Microcebus murinus* with an 80% hind limb extension and a trajectory of 45°.

Lemur catta



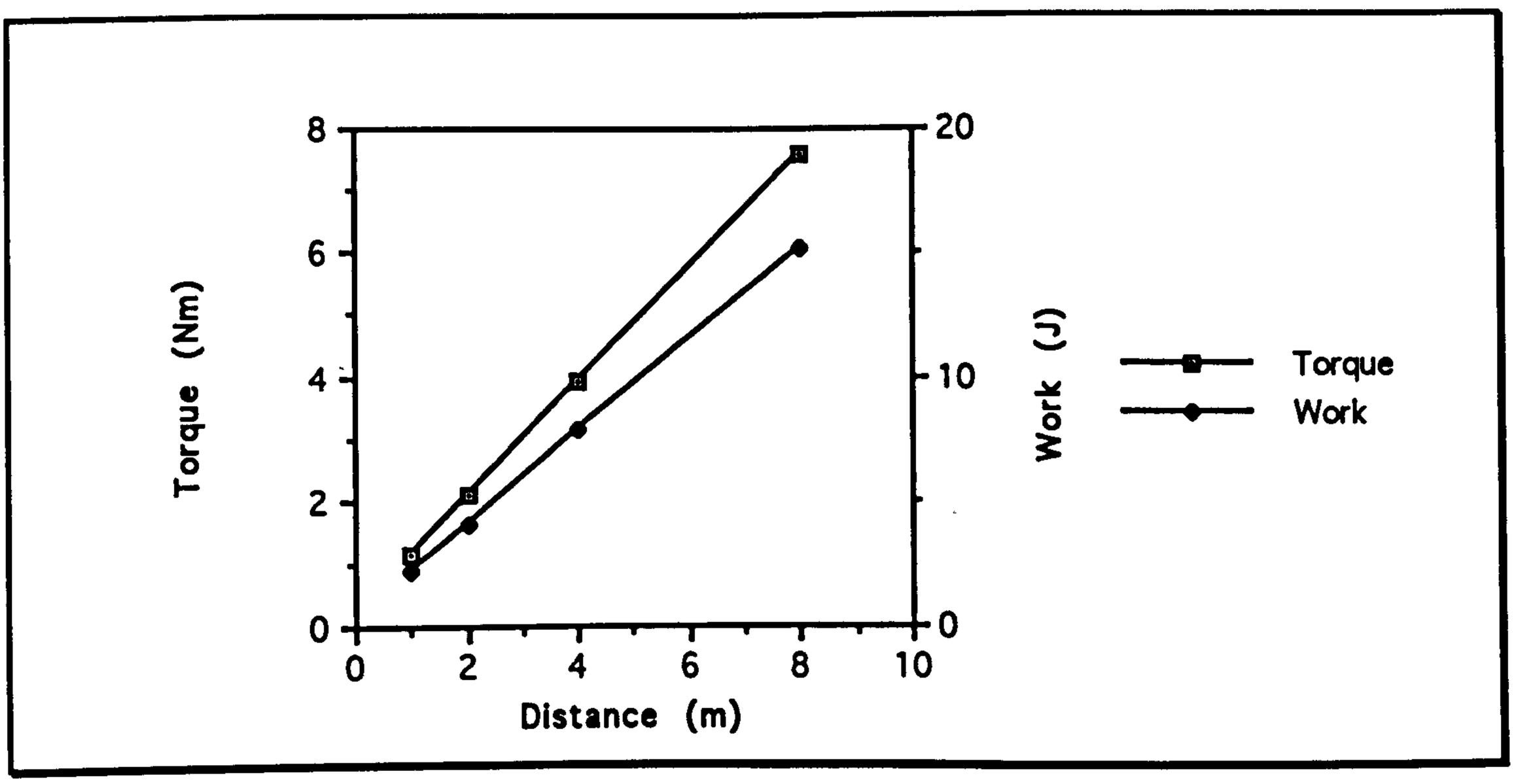
Graph showing the relationship between peak torque about the hip joint, the total work and the leap distance for a simulated leap of *Lemur catta* with an 80% hind limb extension and a trajectory of 45°.

Cheirogaleus major



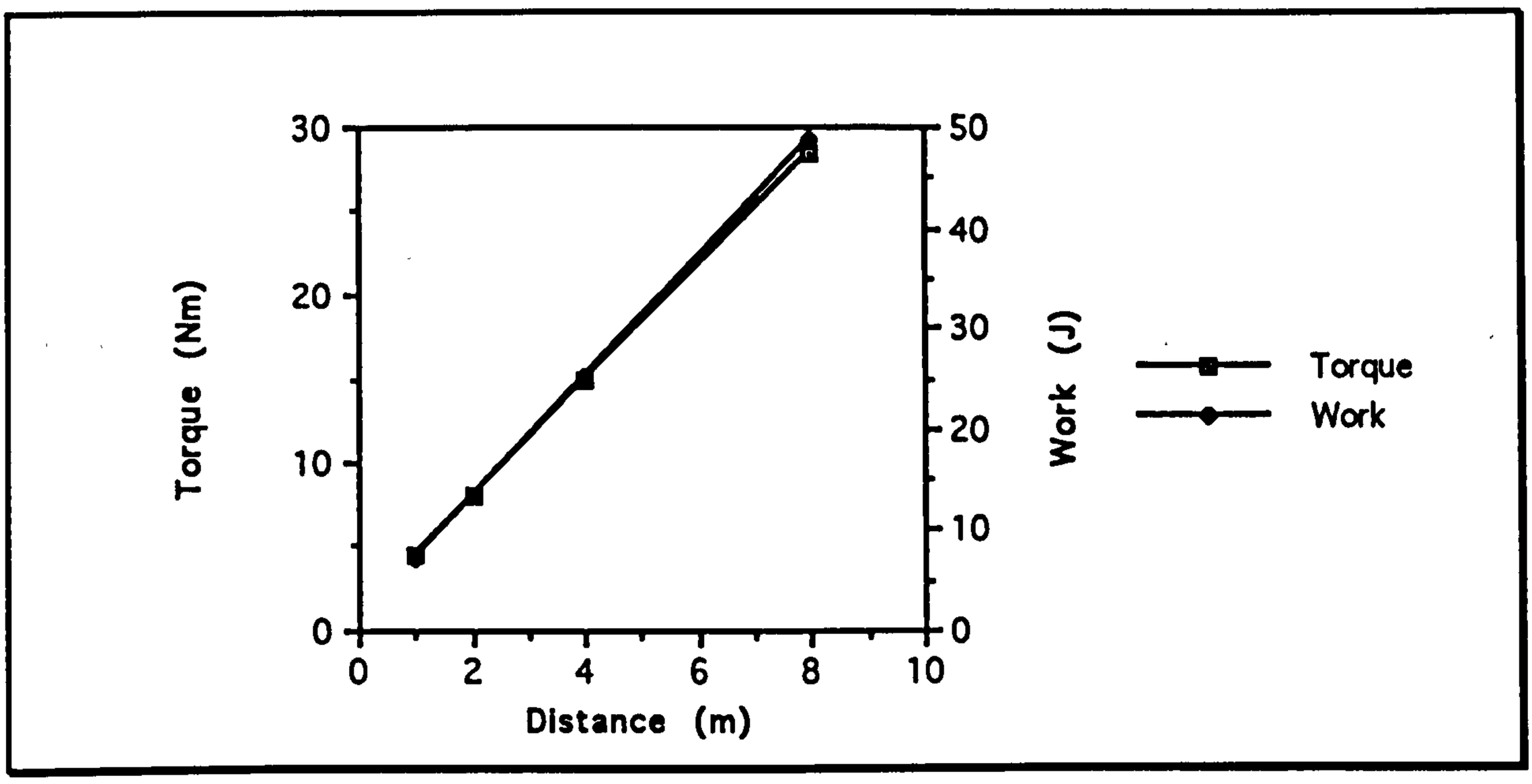
Graph showing the relationship between peak torque about the hip joint, the total work and the leap distance for a simulated leap of *Cheirogaleus major* with an 80% hind limb extension and a trajectory of 45°.

Mirza coquereli



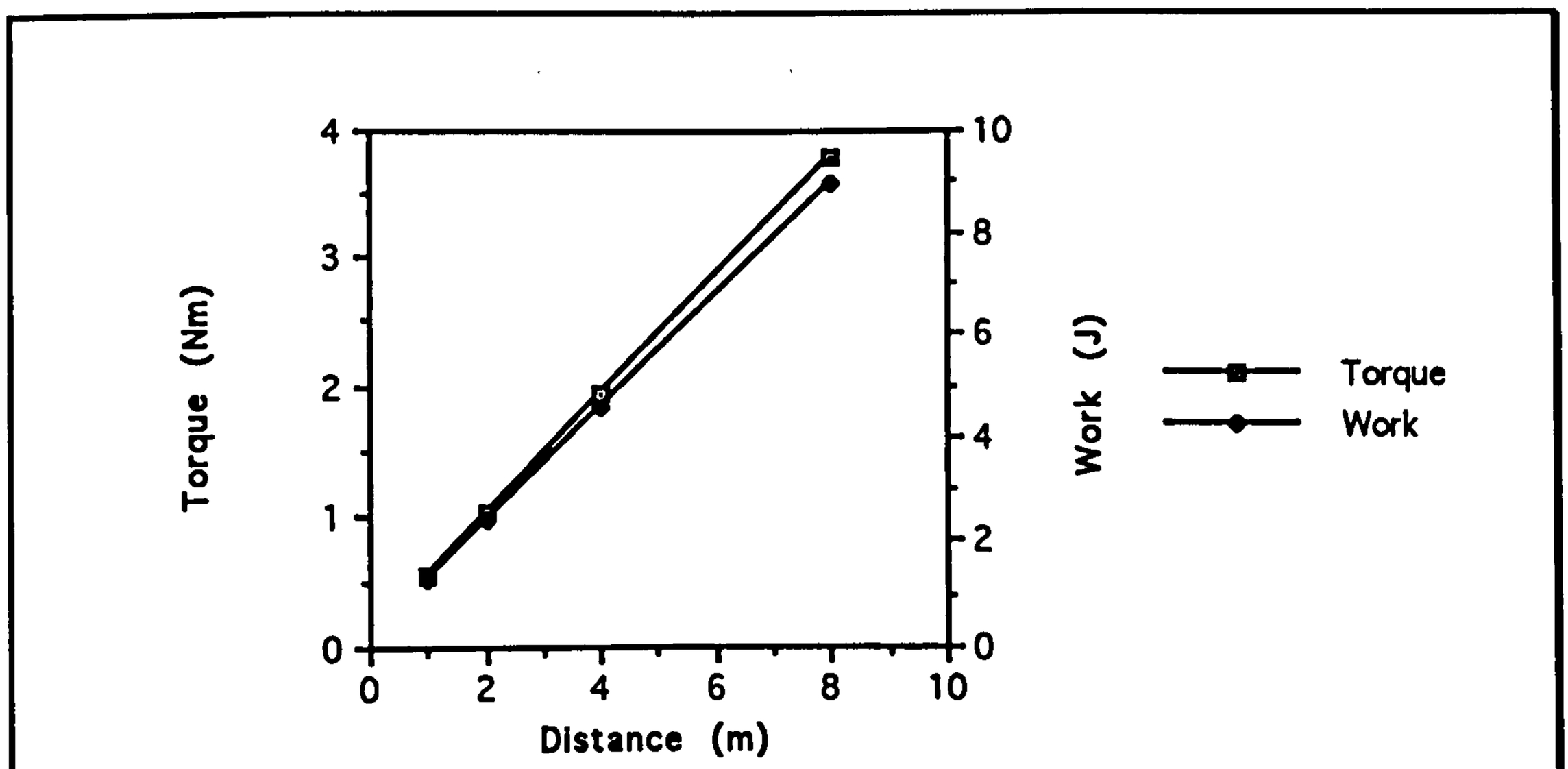
Graph showing the relationship between peak torque about the hip joint, the total work and the leap distance for a simulated leap of *Mirza coquereli* with an 80% hind limb extension and a trajectory of 45°.

Galago gamettii



Graph showing the relationship between peak torque about the hip joint, the total work and the leap distance for a simulated leap of *Galago garnettii* with an 80% hind limb extension and a trajectory of 45°.

Galago moholi



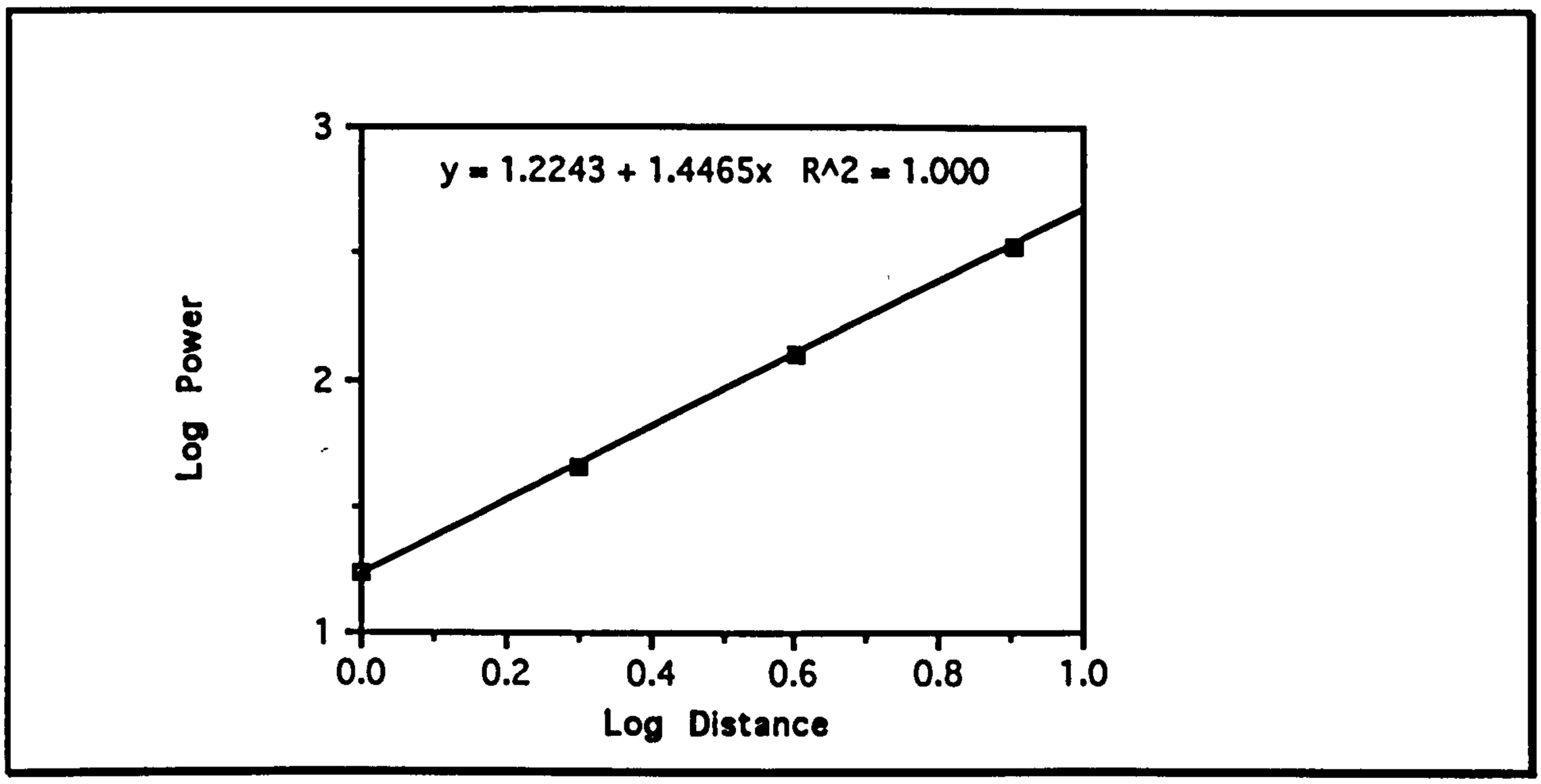
Graph showing the relationship between peak torque about the hip joint, the total work and the leap distance for a simulated leap of *Galago moholi* with an 80% hind limb extension and a trajectory of 45°.

In this model, distance has a simple linear relationship with both torque and total work. In reality, this may not mimic the true effect of distance all that closely, since the importance of the various factors involved in choosing the extension fraction are almost certainly a function of the distance: for short leaps, energetic efficiency is liable to be more important than the peak torque since the peak torque will be well below the maximum the animal can manage, so that a shorter extension fraction with its concommitant lower energy cost is more appropriate.

Torque is only the rotational analogue of force, and in this model, will be proportional to it. As shown before, peak force is linearly related to the distance leapt. The work done depends on the kinetic energy required for the leap. This is proportional to the leap distance, again agreeing with the results of the model.

The following graphs show the effect of leap distance on the peak power required. These have been plotted on logarithmic axes to show clearly that there is a simple mathematical power relationship between peak power and distance leapt.

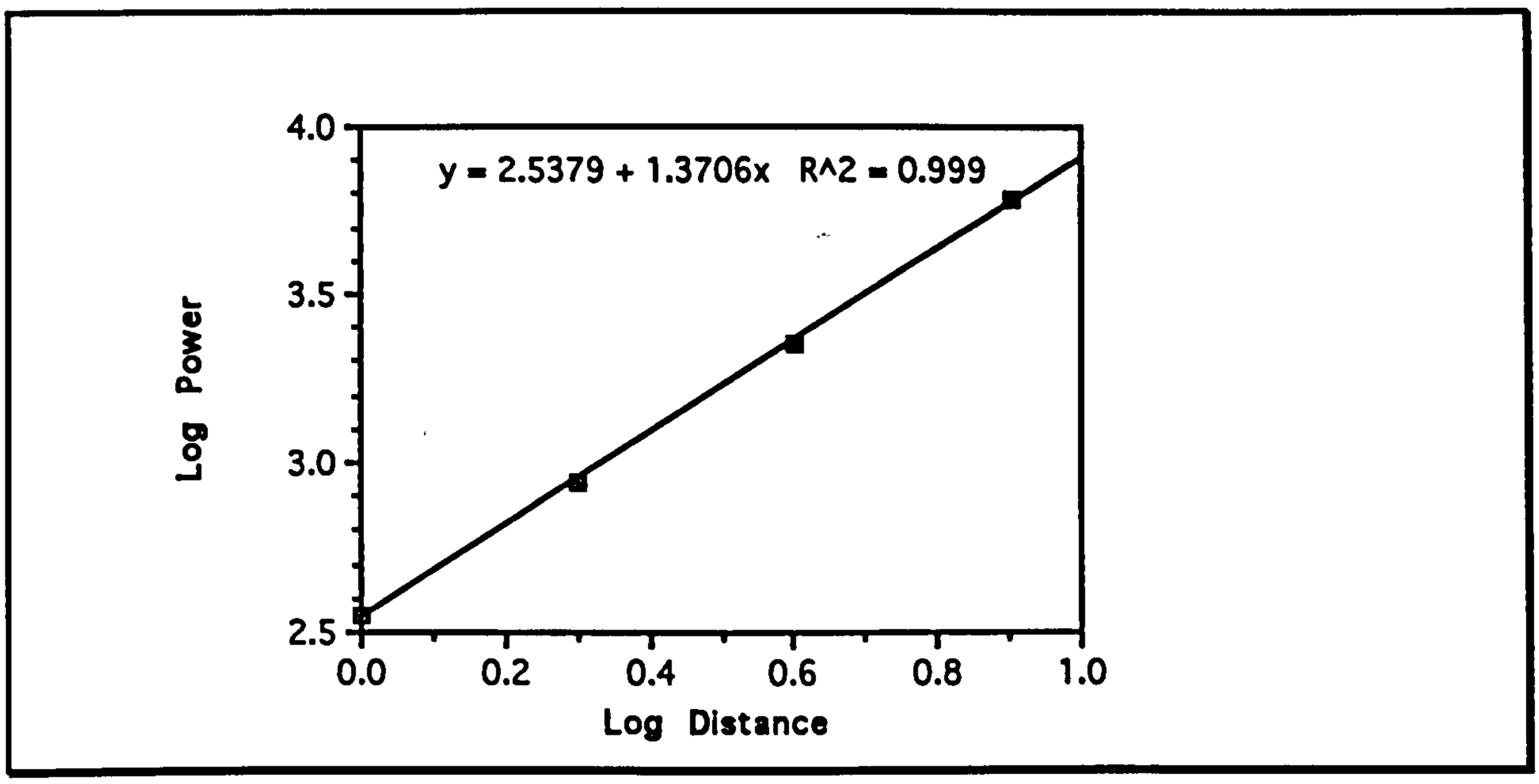
Microcebus murinus



Graph showing the peak power generated about the hip joint of *Microcebus murinus* as a function of distance for a simulated leap at a 45° takeoff trajectory.[36]

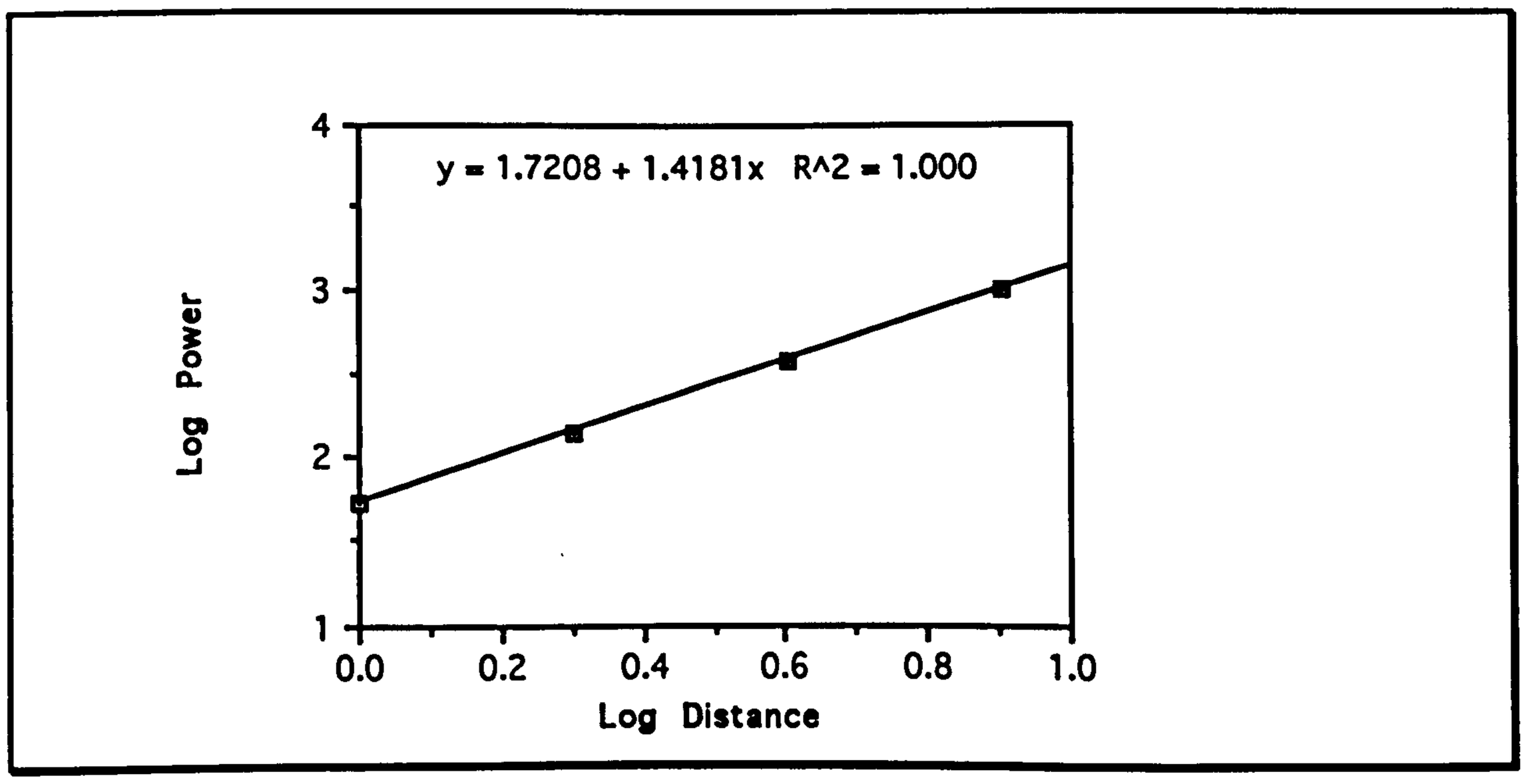
[36] R^2 in the diagram is the r^2 value of the regression line.

Lemur catta



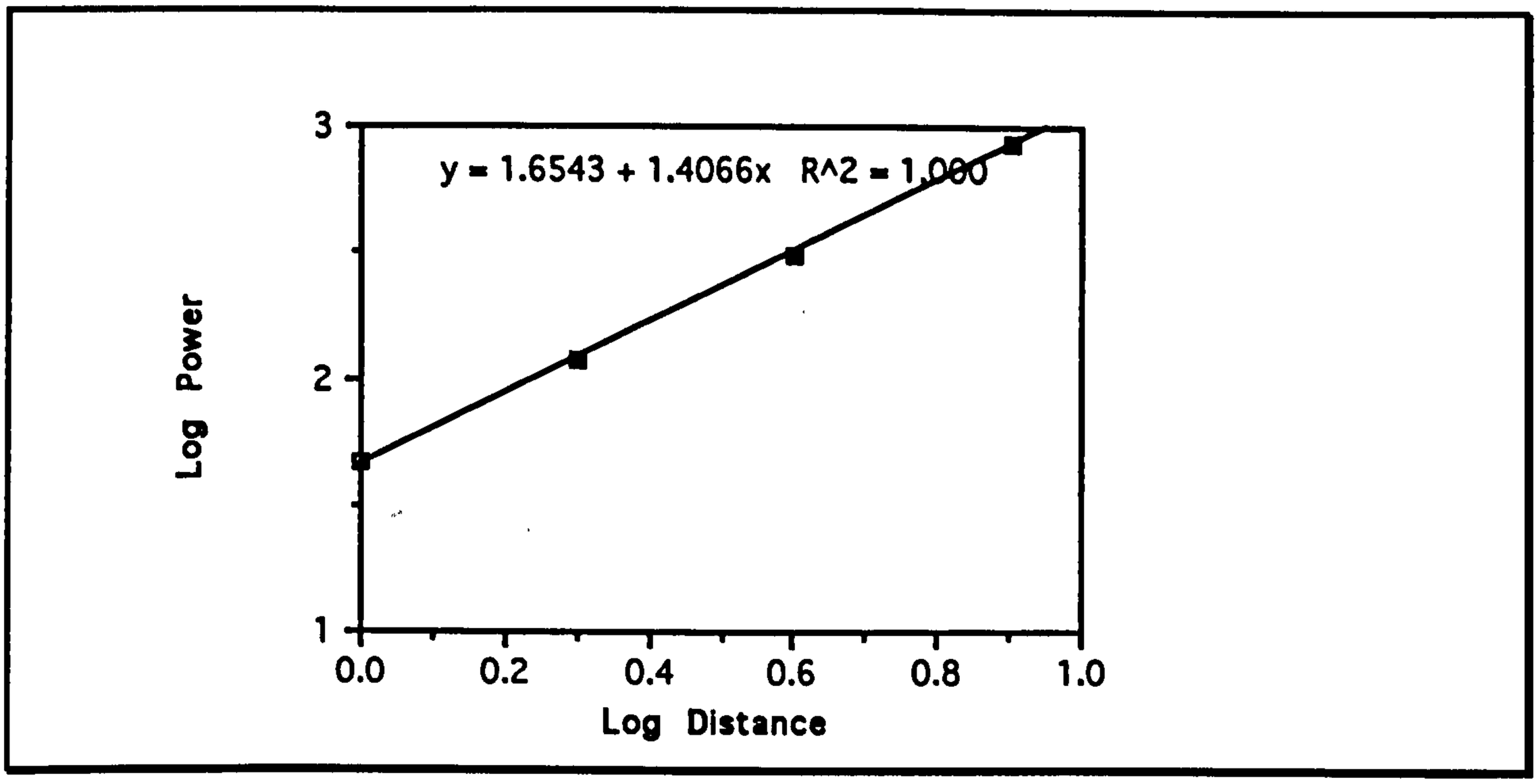
Graph showing the peak power generated about the hip joint of *Lemur catta* as a function of distance for a simulated leap at a 45° takeoff trajectory.

Cheirogaleus major



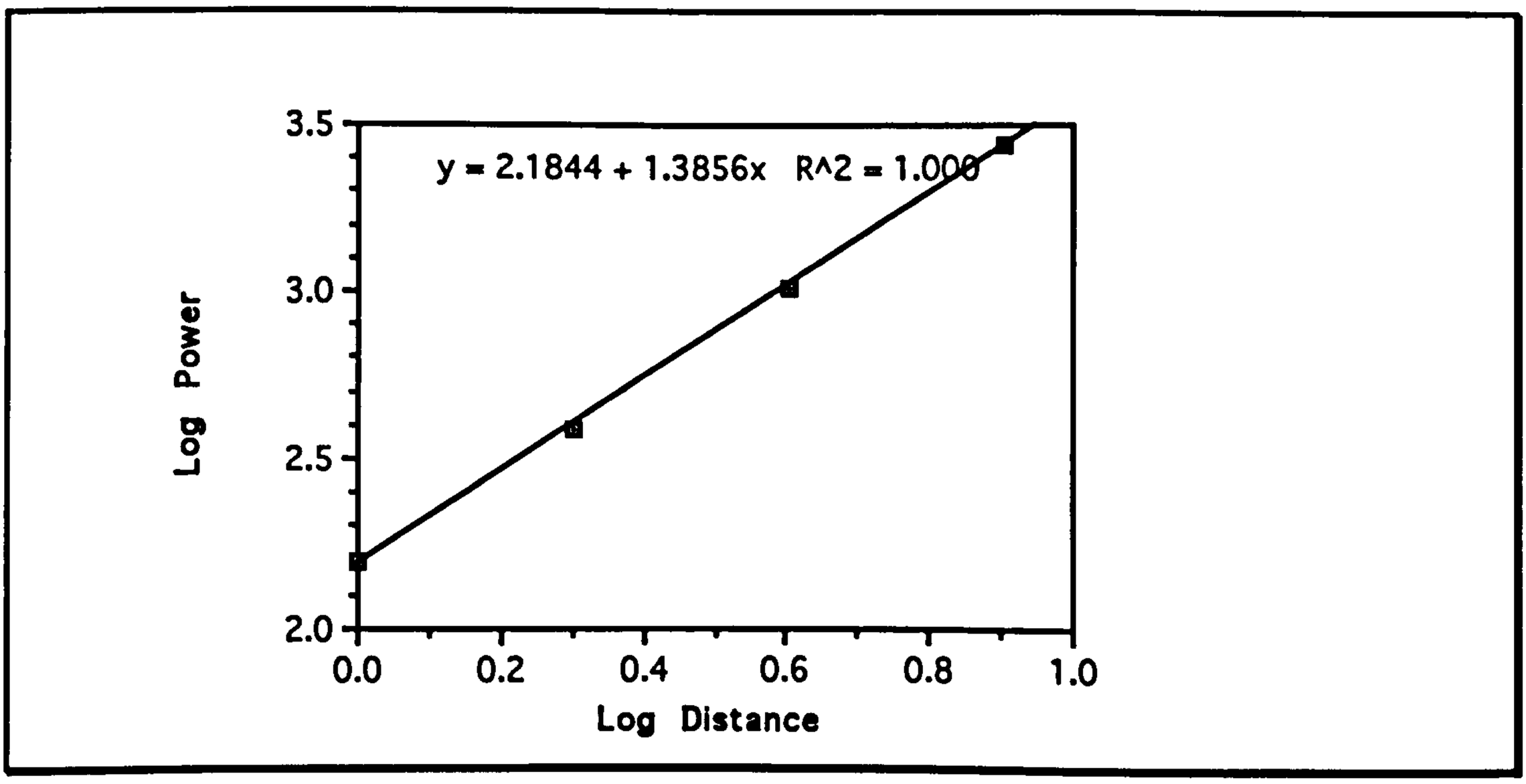
Graph showing the peak power generated about the hip joint of *Cheirogaleus major* as a function of distance for a simulated leap at a 45° takeoff trajectory.

Mirza coquereli



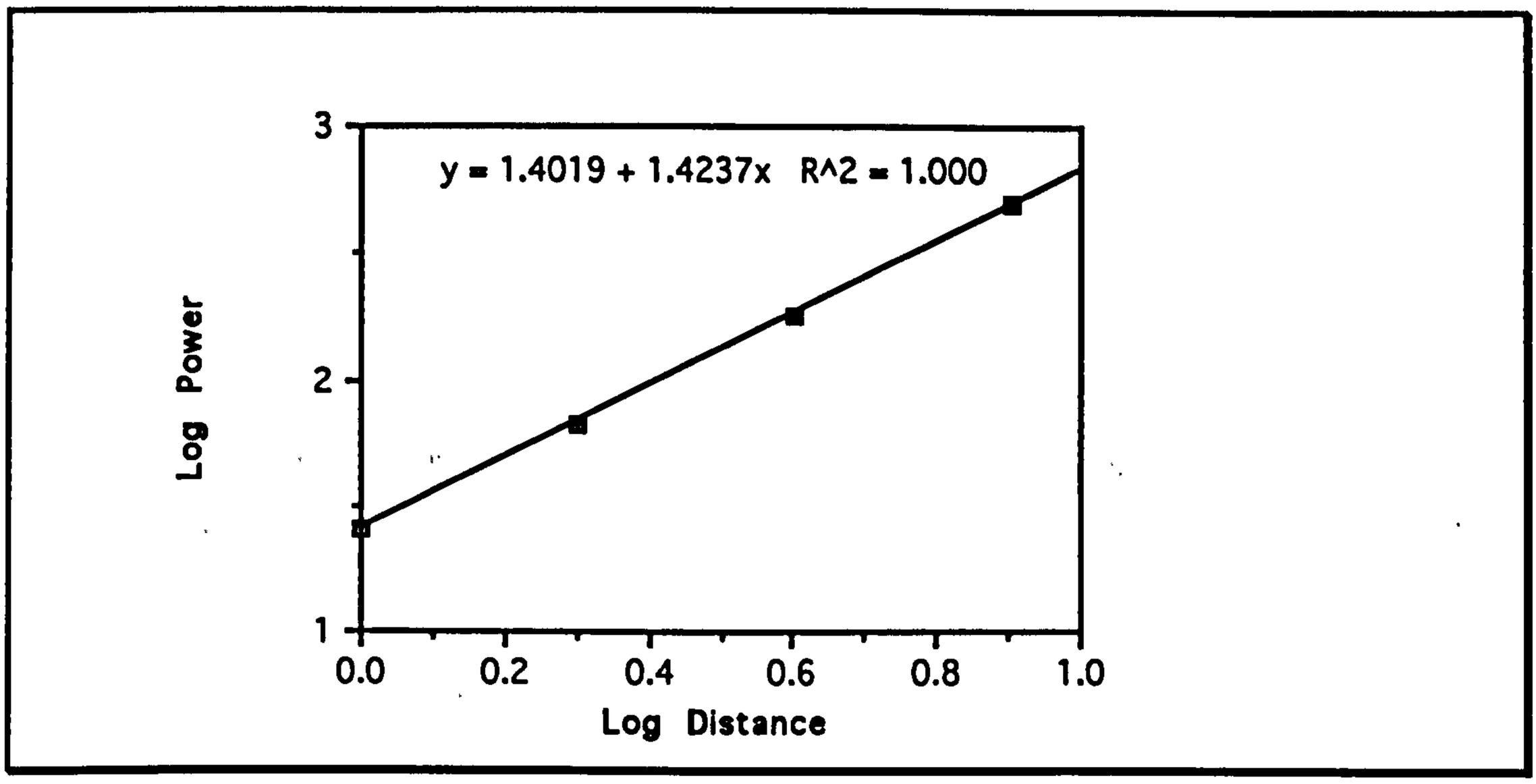
Graph showing the peak power generated about the hip joint of *Mirza coquereli* as a function of distance for a simulated leap at a 45° takeoff trajectory.

Galago garnettii



Graph showing the peak power generated about the hip joint of *Galago garnettii* as a function of distance for a simulated leap at a 45° takeoff trajectory.

Galago moholi



Graph showing the peak power generated about the hip joint of *Galago moholi* as a function of distance for a simulated leap at a 45° takeoff trajectory.

The results for power do not quite agree with those predicted in the general relationships. Power can be defined as follows:

(1) $$P = Fv$$

Where:

P is the power

F is the force

v is the velocity

From the ballistic equations:

(2) $$v_{to} \propto r^{\frac{1}{2}}$$

(3) $$F \propto r$$

Where:

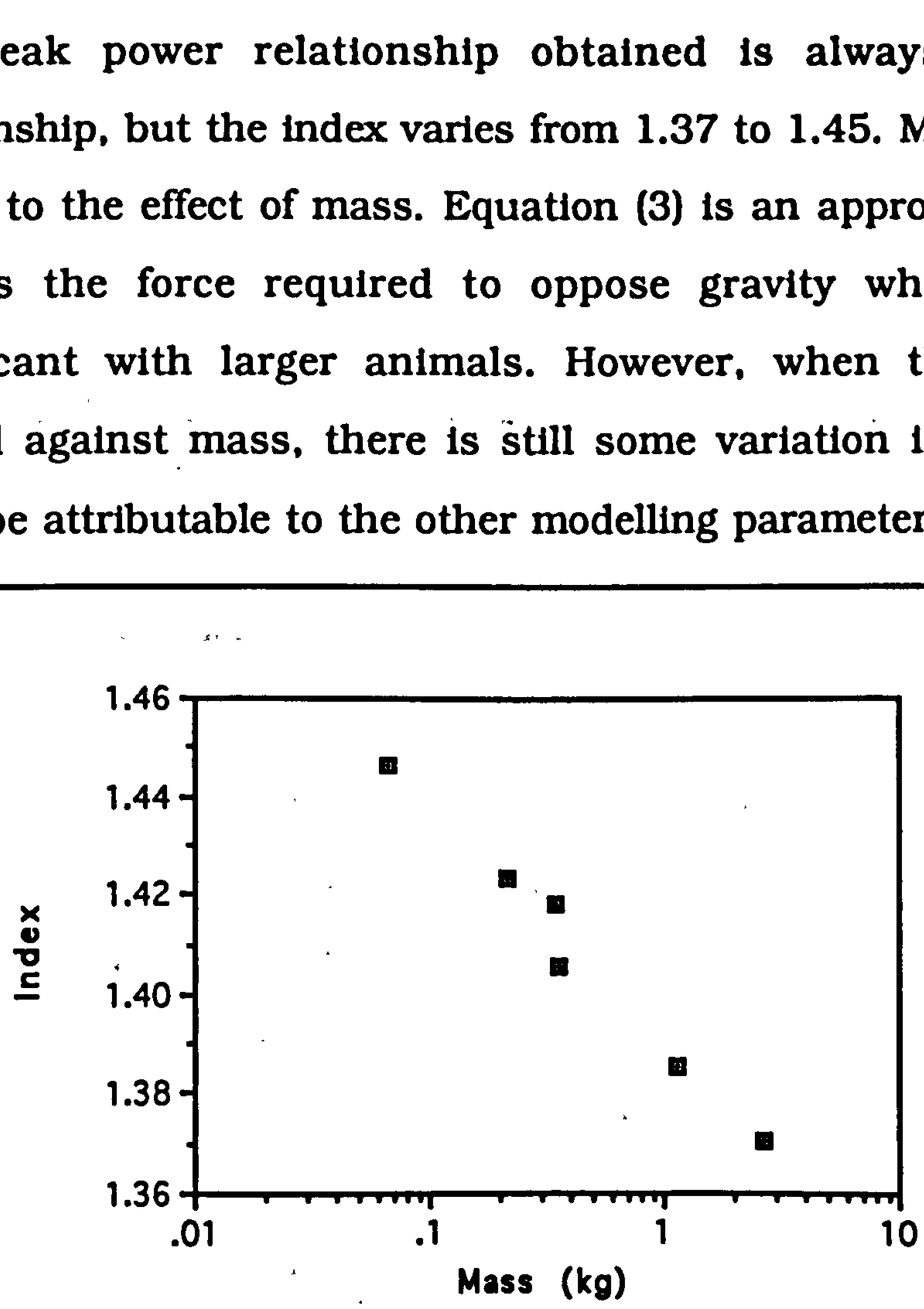
v_{to} is the takeoff velocity

r is the leap distance

Therefore:

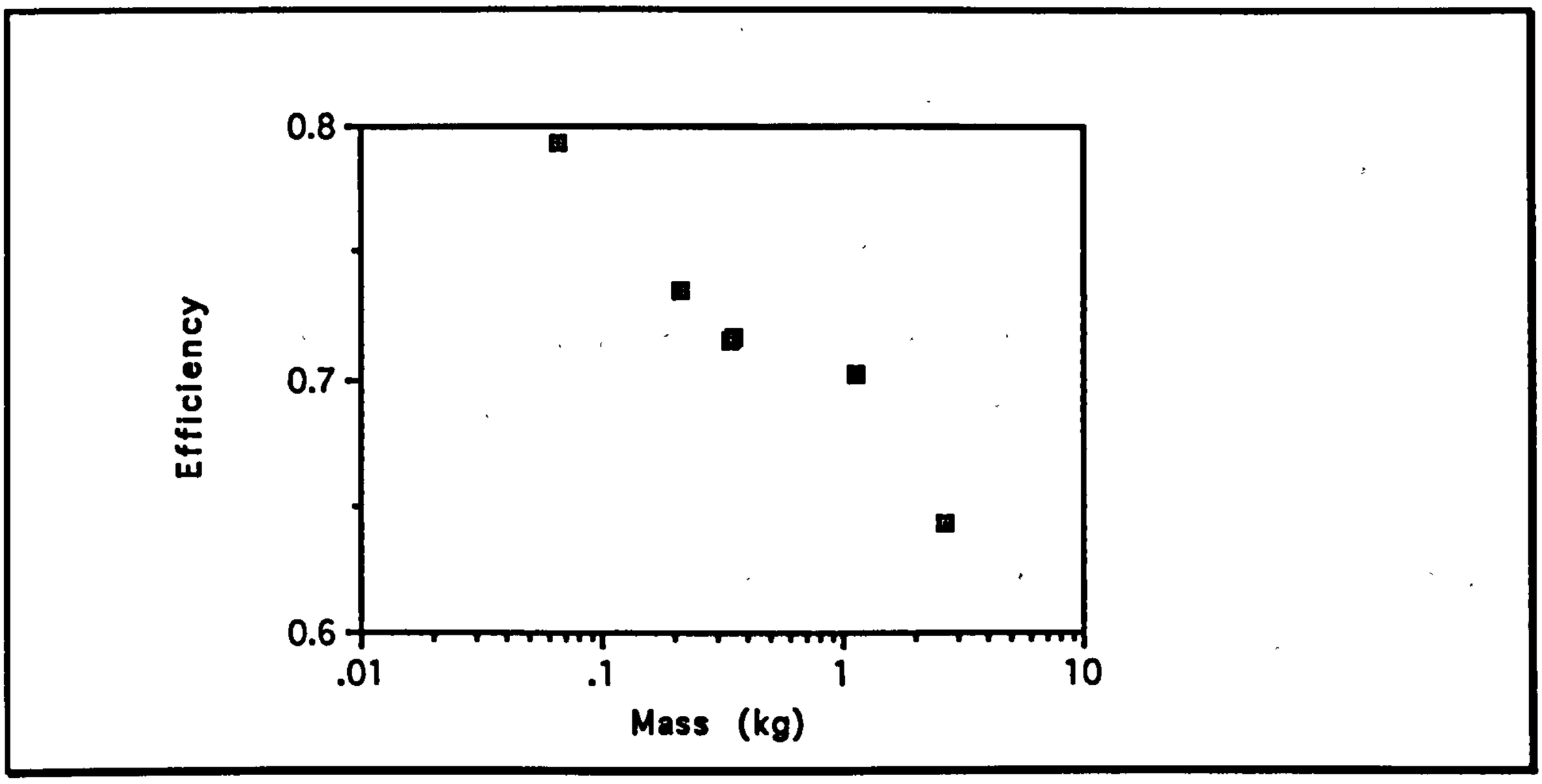
(4) $$P \propto r^2$$

The peak power relationship obtained is always a simple power relationship, but the index varies from 1.37 to 1.45. Most of this variation is due to the effect of mass. Equation (3) is an approximation because it ignores the force required to oppose gravity which becomes more significant with larger animals. However, when the power index is plotted against mass, there is still some variation in the values which must be attributable to the other modelling parameters:



Graph showing the relationship between the mass of the animal and the index of the power relation between the peak power and the distance leapt. No attempt has been made to fit a curve to the points, since any measure of significance would be largely meaningless. However the trend indicated is very clear: that larger animals have a slower increase in peak power demands for longer leaps.

This next graph shows the mechanical efficiency[37] calculated from the model as a function of mass:



Graph showing the relationship between the mass of the animal and the mechanical efficiency of the leap. The mechanical efficiency has been calculated as the ratio of the useful work performed to the total work done by the animal.

This graph indicates that the smaller animals are able to leap more efficiently than the larger ones. The larger animals are less efficient mechanically because most of the energy input that does not do useful work in accelerating the centre of mass of the animal to the required takeoff velocity is used to rotate the body segments of the animal. The internal rotational energy is dependent on the moments of inertia of the segments, and since moment of inertia depends on the 5^{th} power of linear dimensions[38], this cost is going to be higher for larger animals.

---

[37]The mechanical efficiency is the mechanical work done in the model divided by the amount of useful mechanical work obtained.

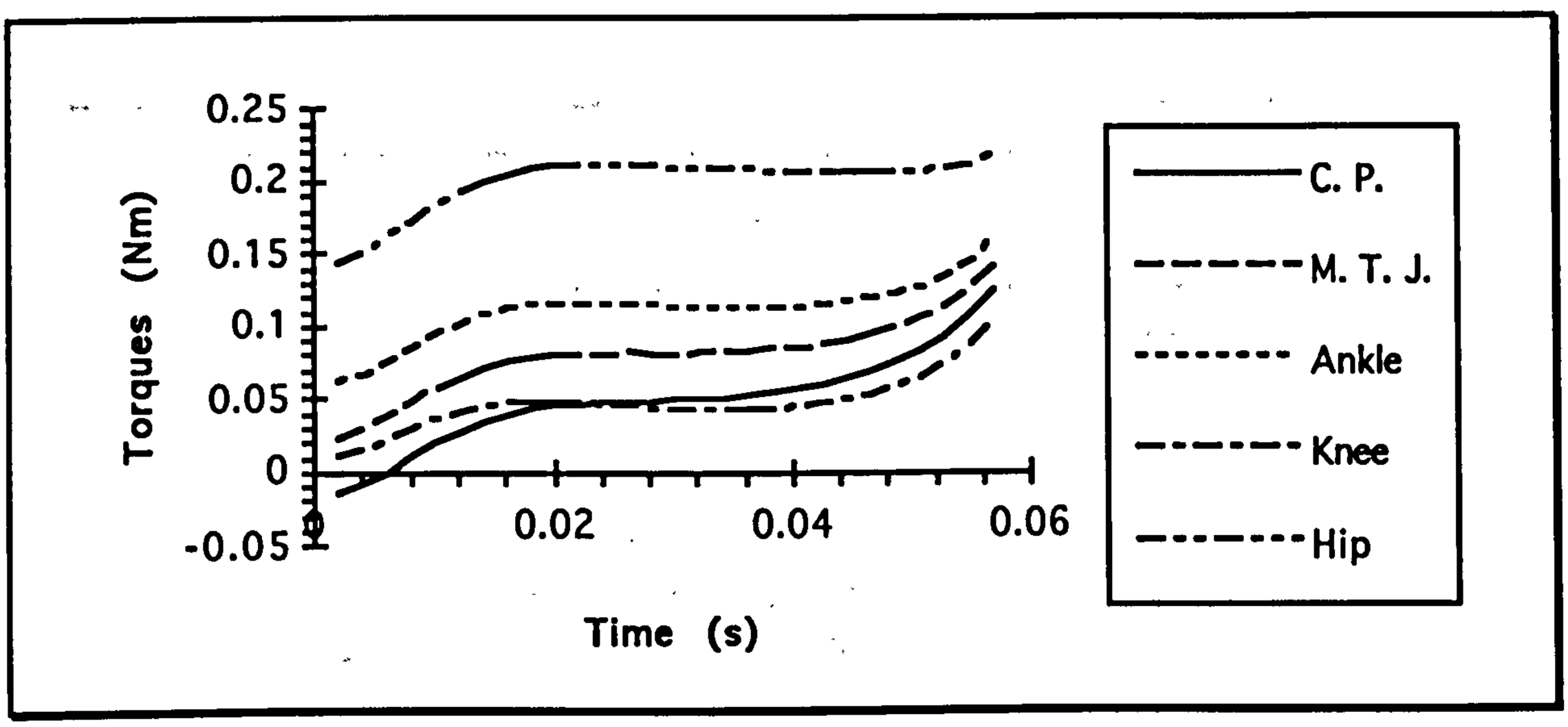
[38]The moment of inertia of a rod is proportional to the mass times the length squared. The mass is proportional to the volume, and the volume depends on the cube of linear dimensions. Therefore, the moment of inertia depends on the linear dimension to the power five. This relationship will only hold precisely for geometrically scaled animals.

Mechanical efficiency, however, provides a very incomplete picture. What concerns the animal more is the metabolic efficiency of the action. This is the metabolic (chemical) energy required to perform an amount of useful work. The efficiency for converting metabolic energy into useful in ideal conditions is about 0.22 (Dickinson 1929) However, he and others have shown that this efficiency depends on a large number of other factors. When looking at animals rather than humans, the efficiency of obtaining energy from food also has to be considered. The real question is the amount of food the animal needs to eat to perform one Joule of useful work. Metabolic efficiency for digestion also depend on various factors including diet, but is generally higher in larger animals (Martin et al. 1985).

Time Dependence

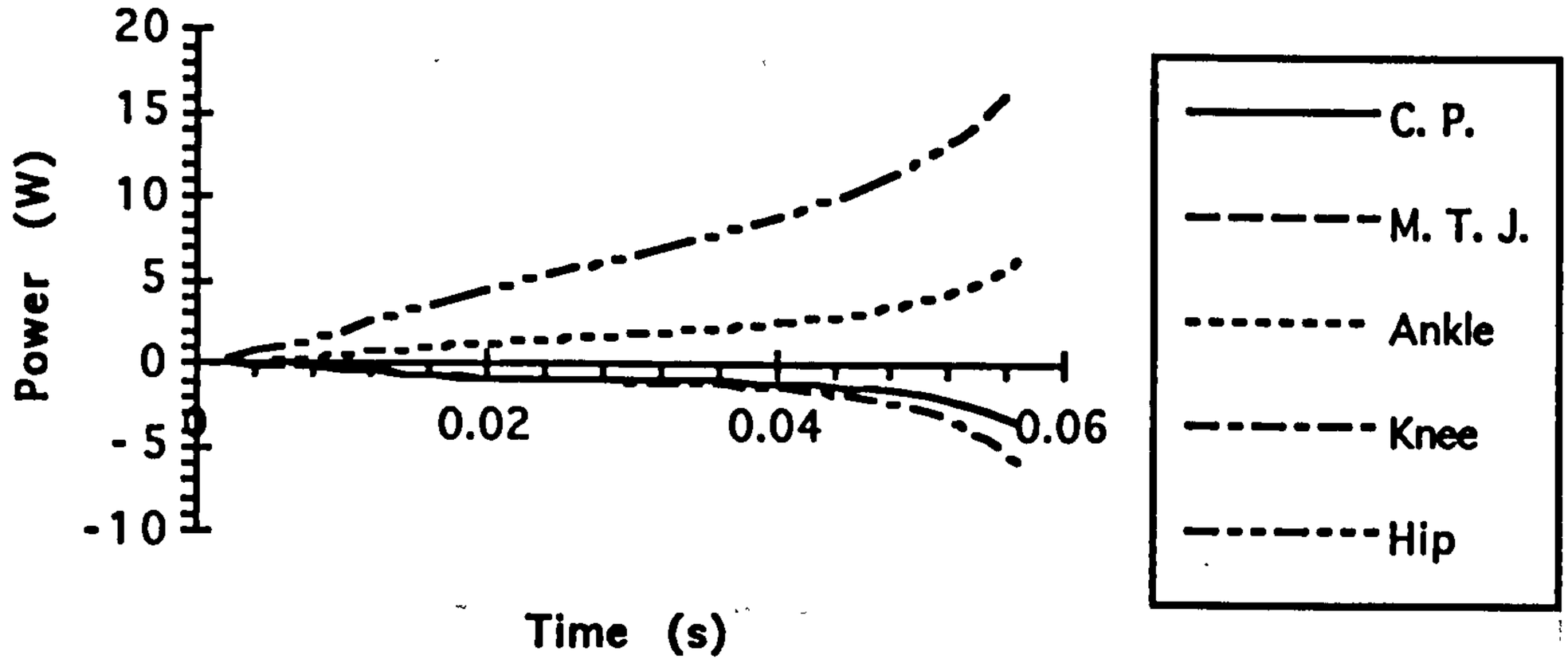
The following graphs show the time dependent data for each joint and segment in the hind-limb. As before, an extension fraction of 80% has been chosen, and the leap distance is 1 m. This latter choice does not effect the shapes of the curves, though the former does.

Microcebus murinus

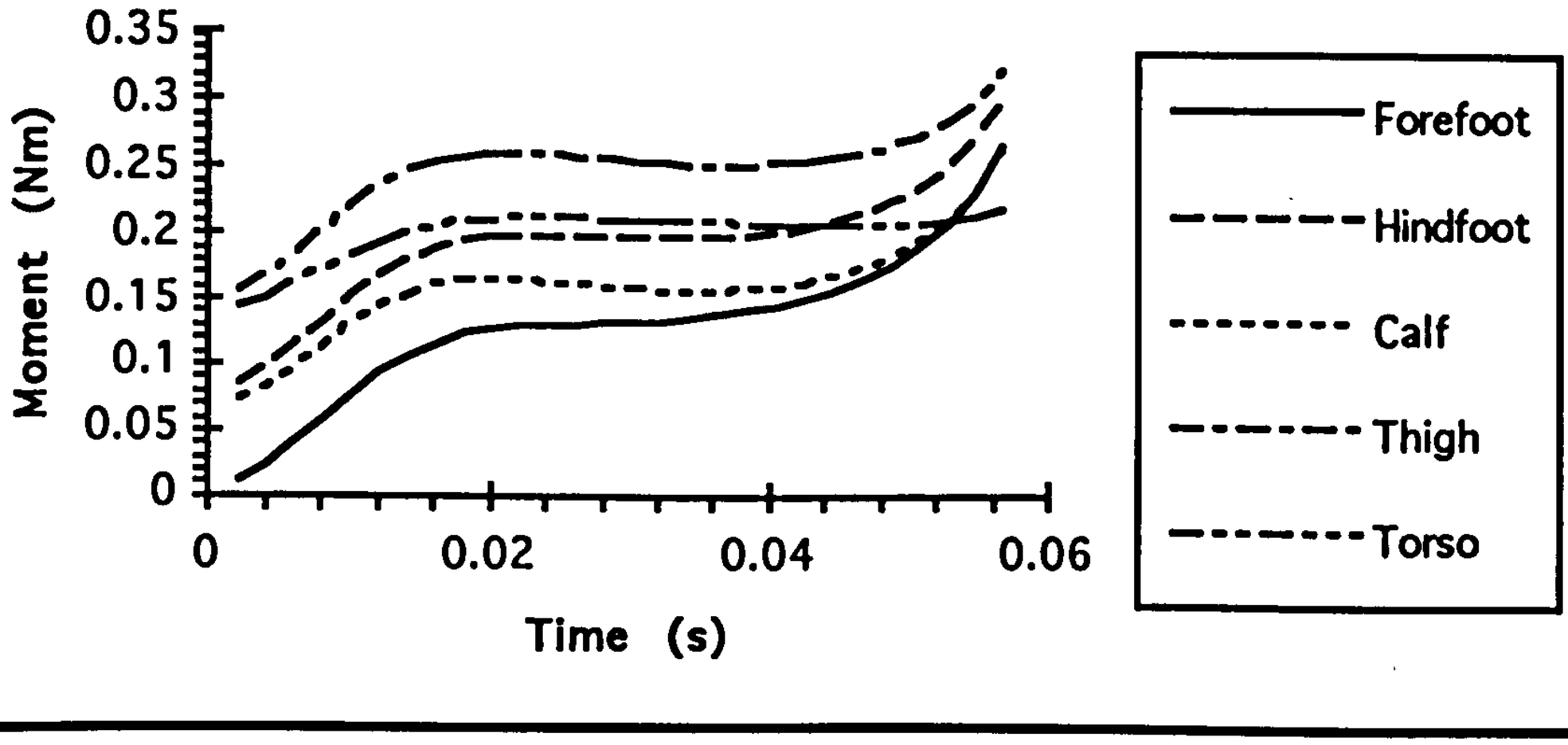


Graph showing the torque about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Microcebus murinus*.[39]

[39]M. T. J. is an abbreviation for mid-tarsal joint. This is a descriptive term for the joint complex halfway along the foot. C. P. is the contact point at the tip of the toe.

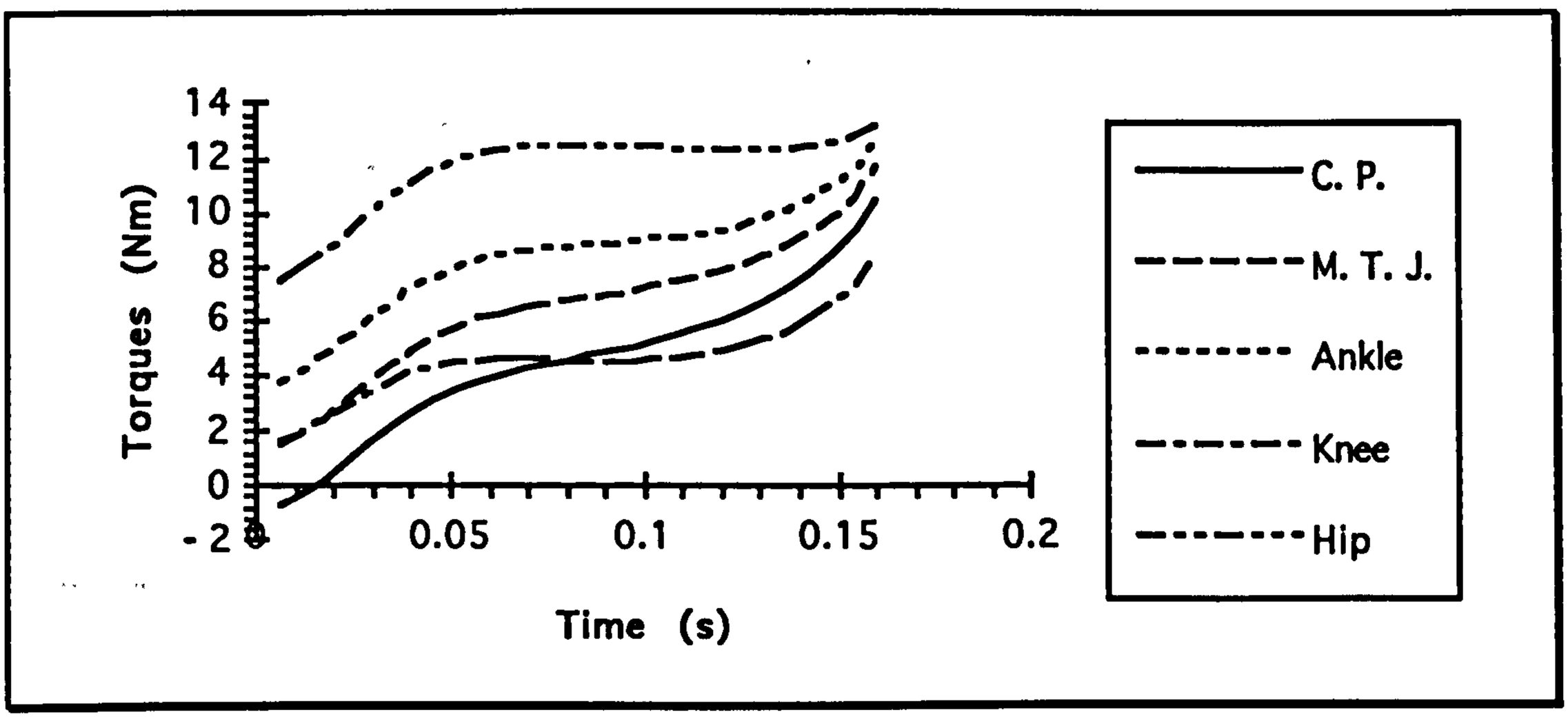


Graph showing the power about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Microcebus murinus*.

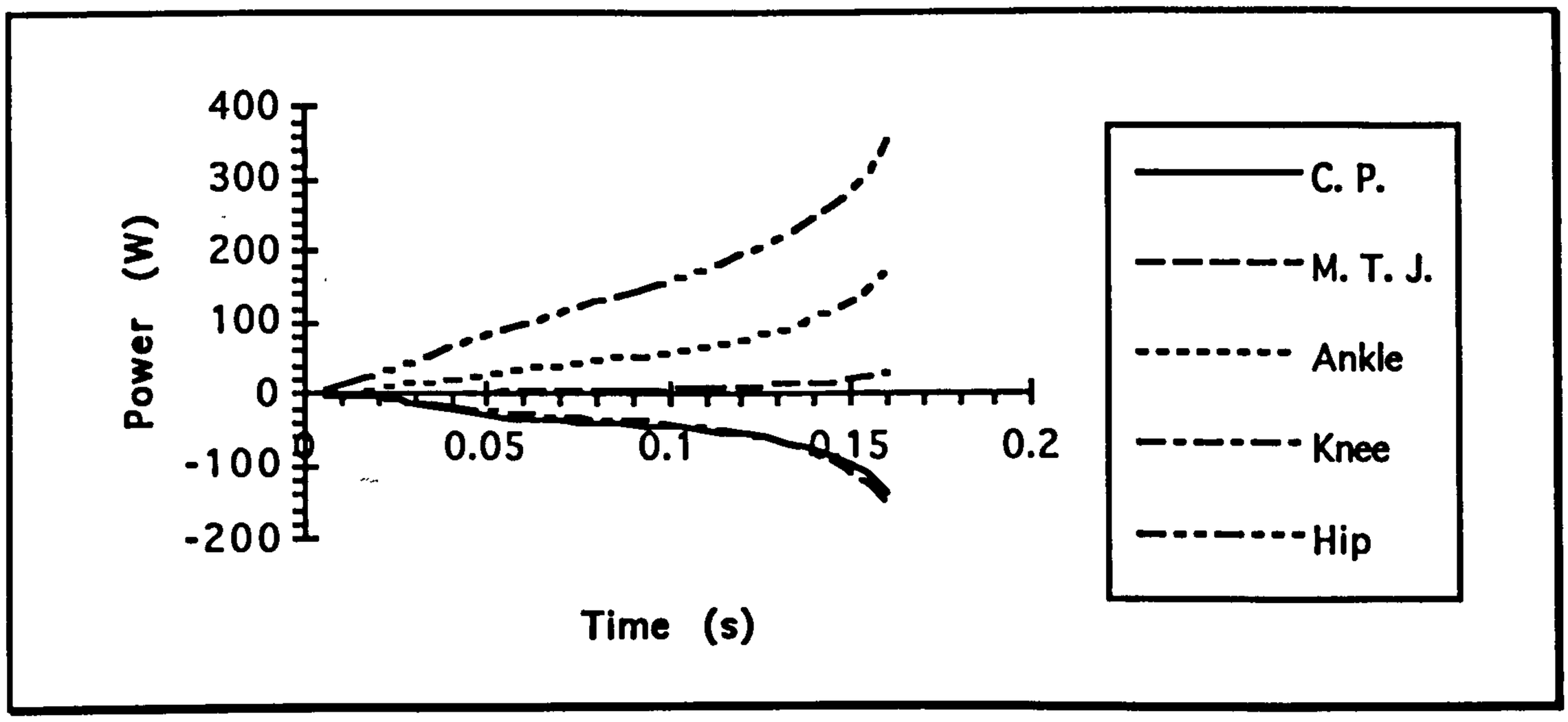


Graph showing the bending moment about the segments in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Microcebus murinus*.

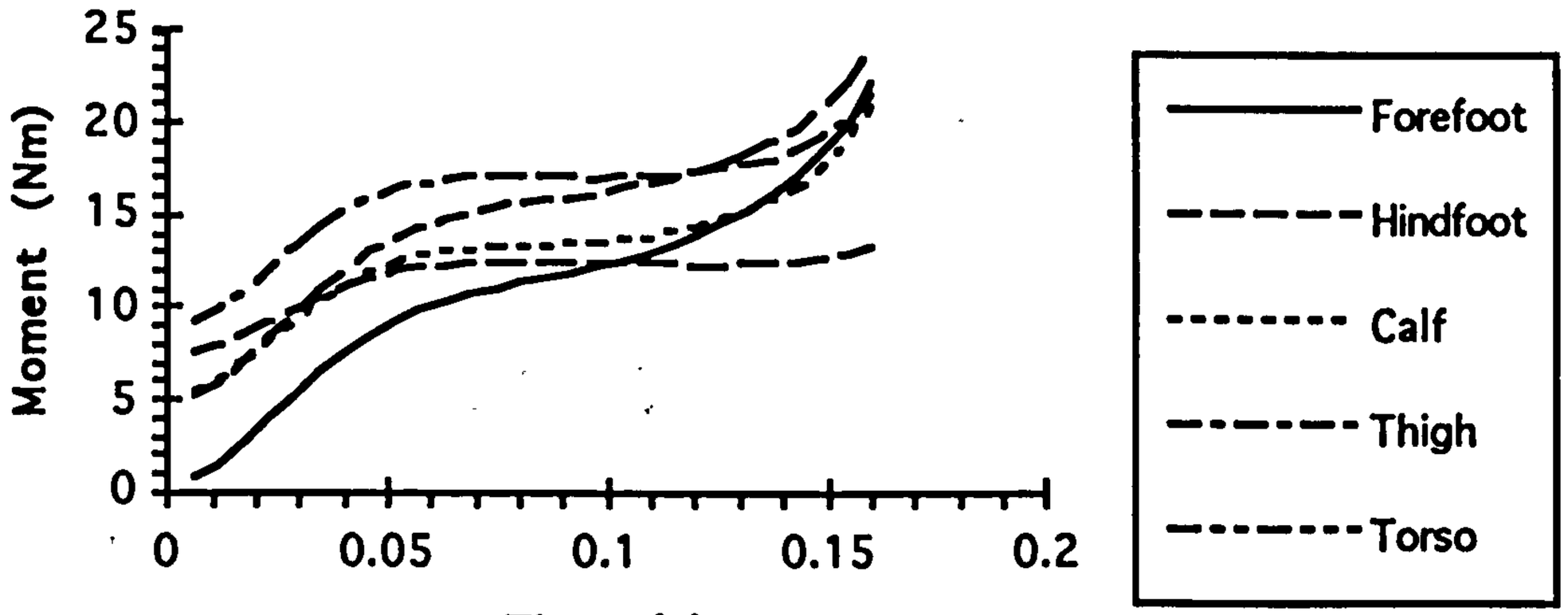
Lemur catta



Graph showing the torque about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Lemur catta*.

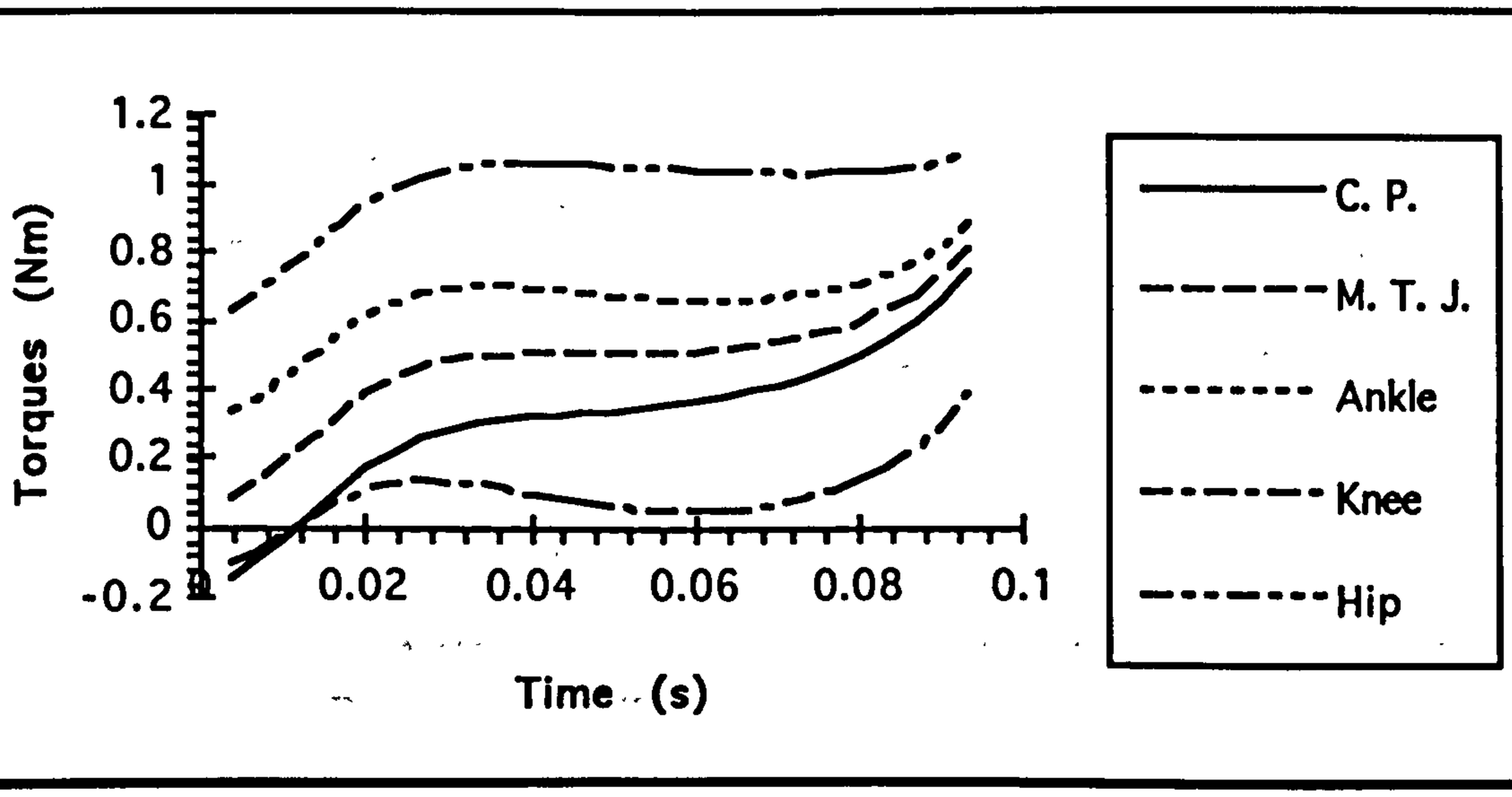


Graph showing the power about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Lemur catta*.

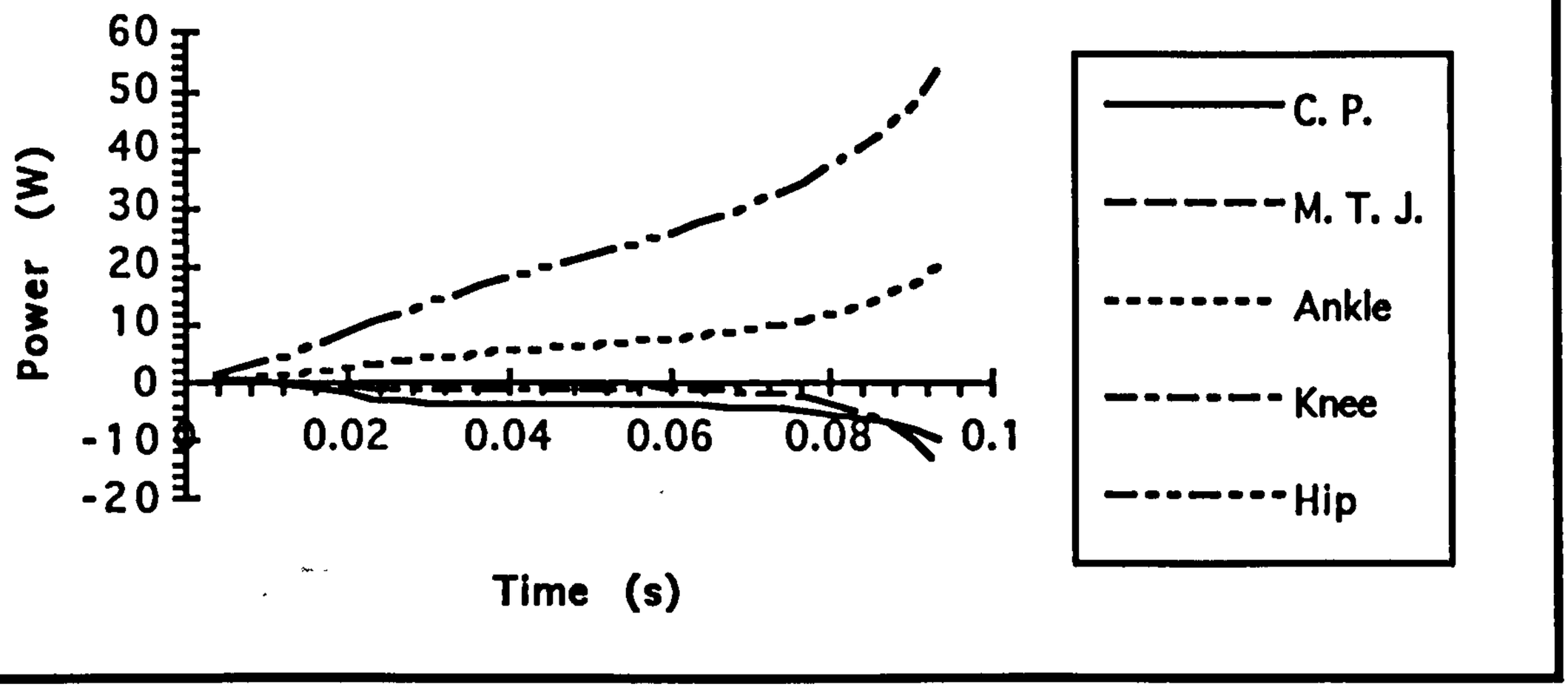


Graph showing the bending moment about the segments in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Lemur catta*.

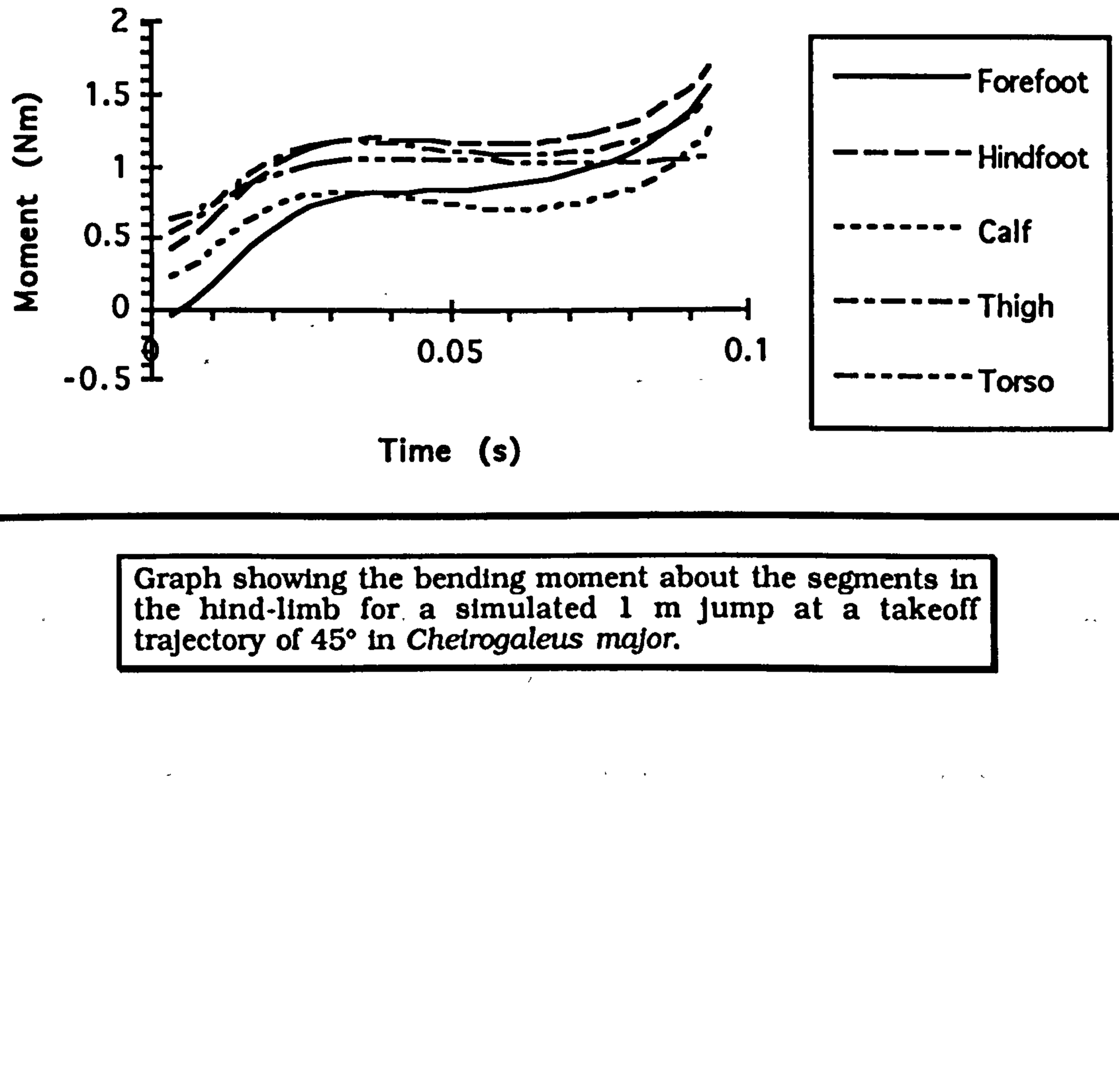
Chelrogaleus major



Graph showing the torque about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Chelrogaleus major*.

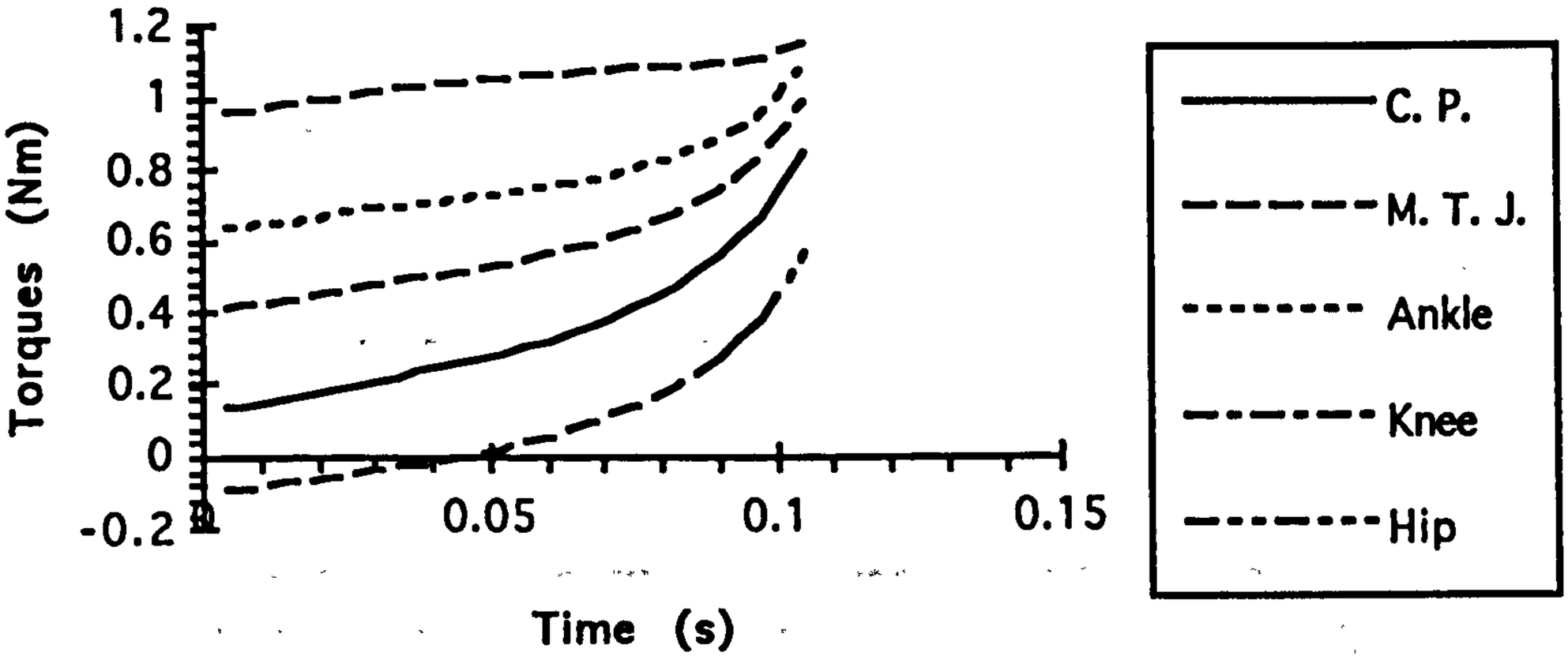


Graph showing the power about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Chetrogaleus major*.

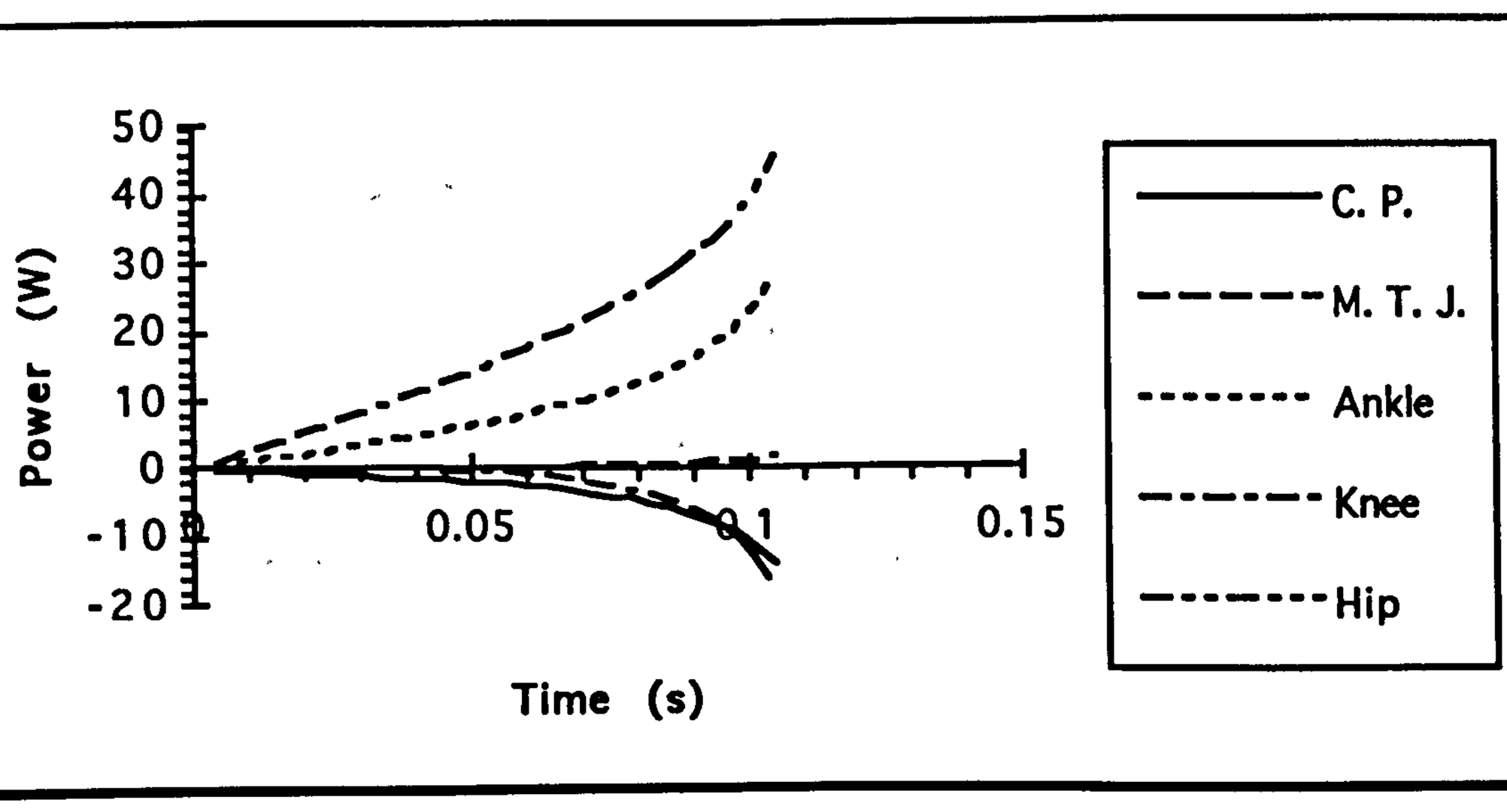


Graph showing the bending moment about the segments in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Chetrogaleus major*.

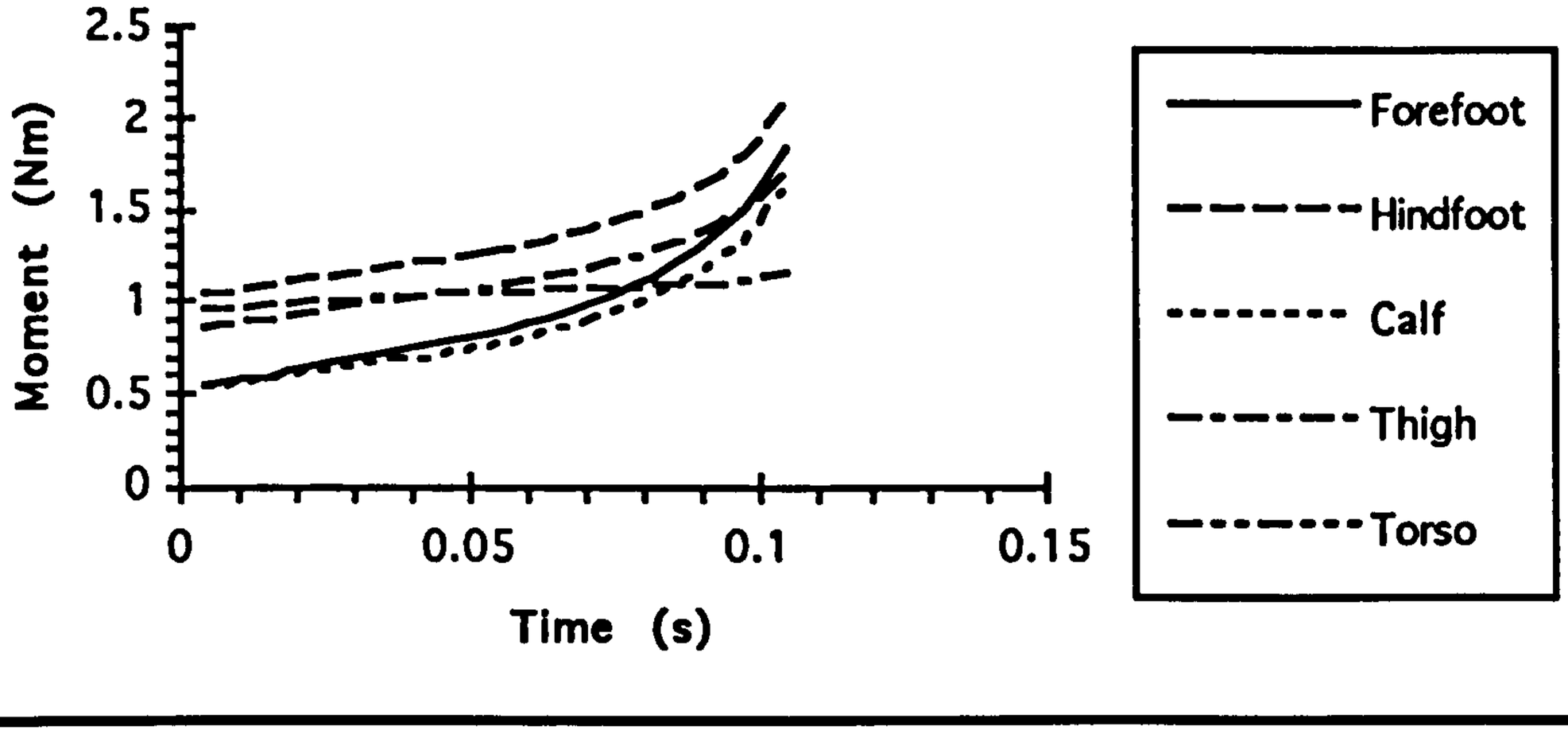
Mirza coquerelli



Graph showing the torque about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Mirza coquereli*.

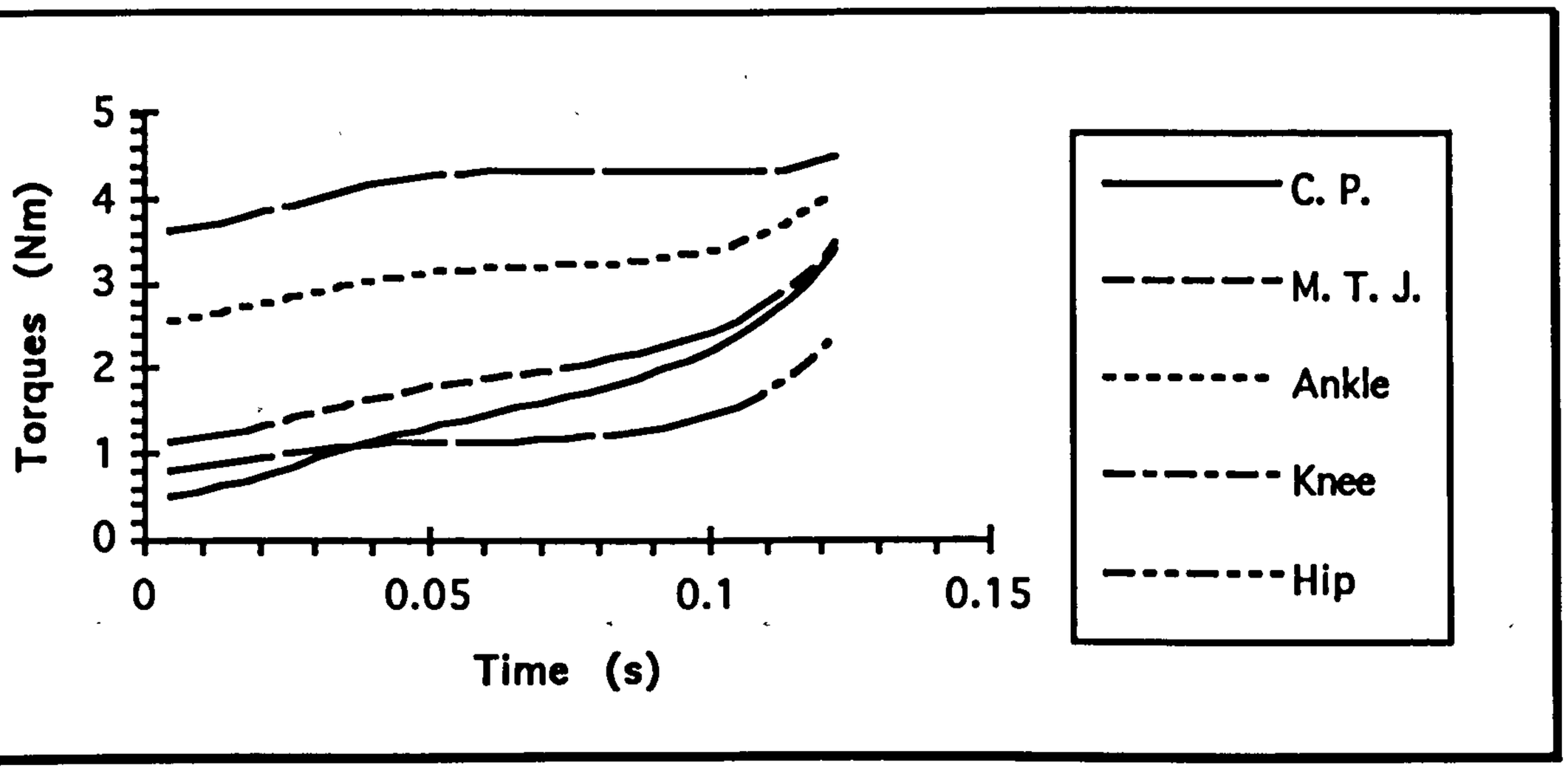


Graph showing the power about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Mirza coquereli*.

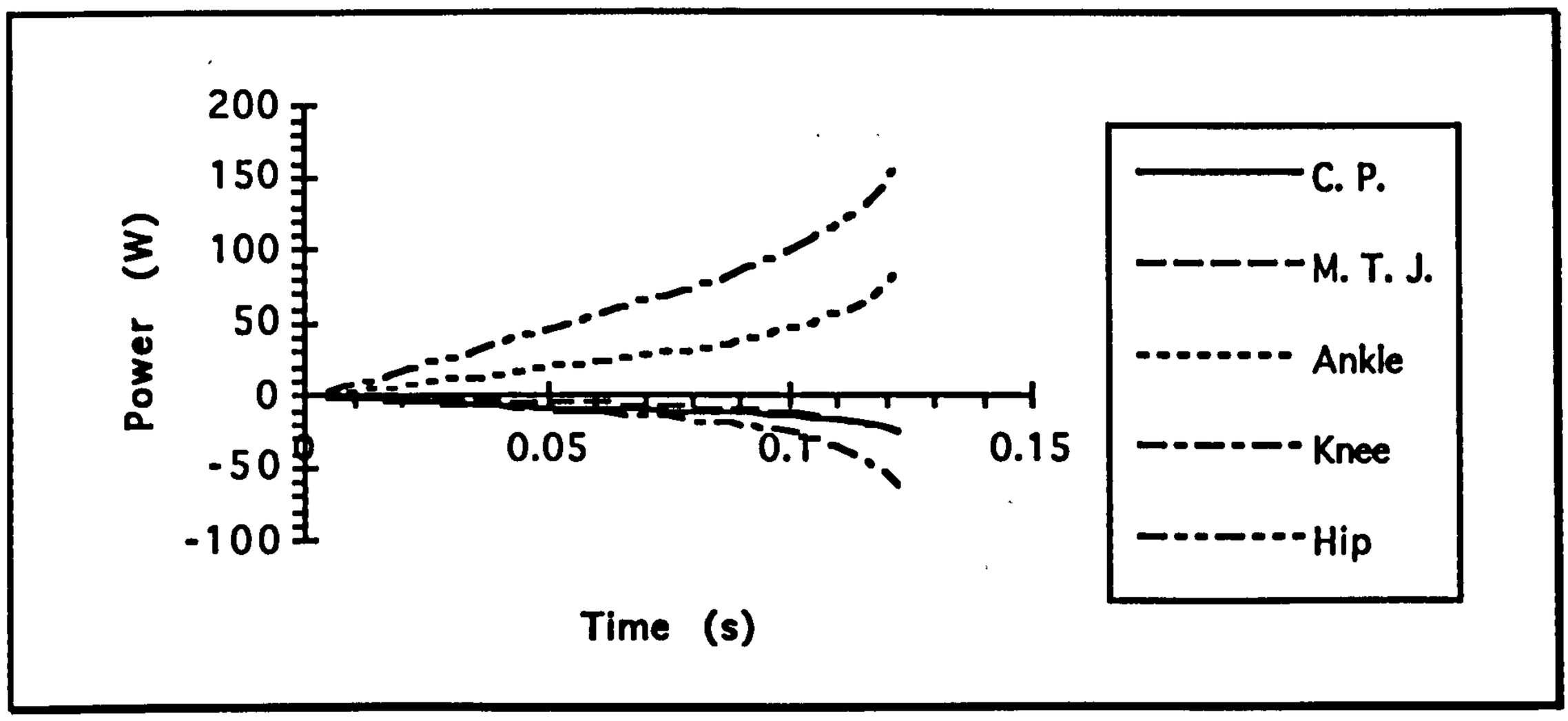


Graph showing the bending moment about the segments in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Mirza coquereli*.

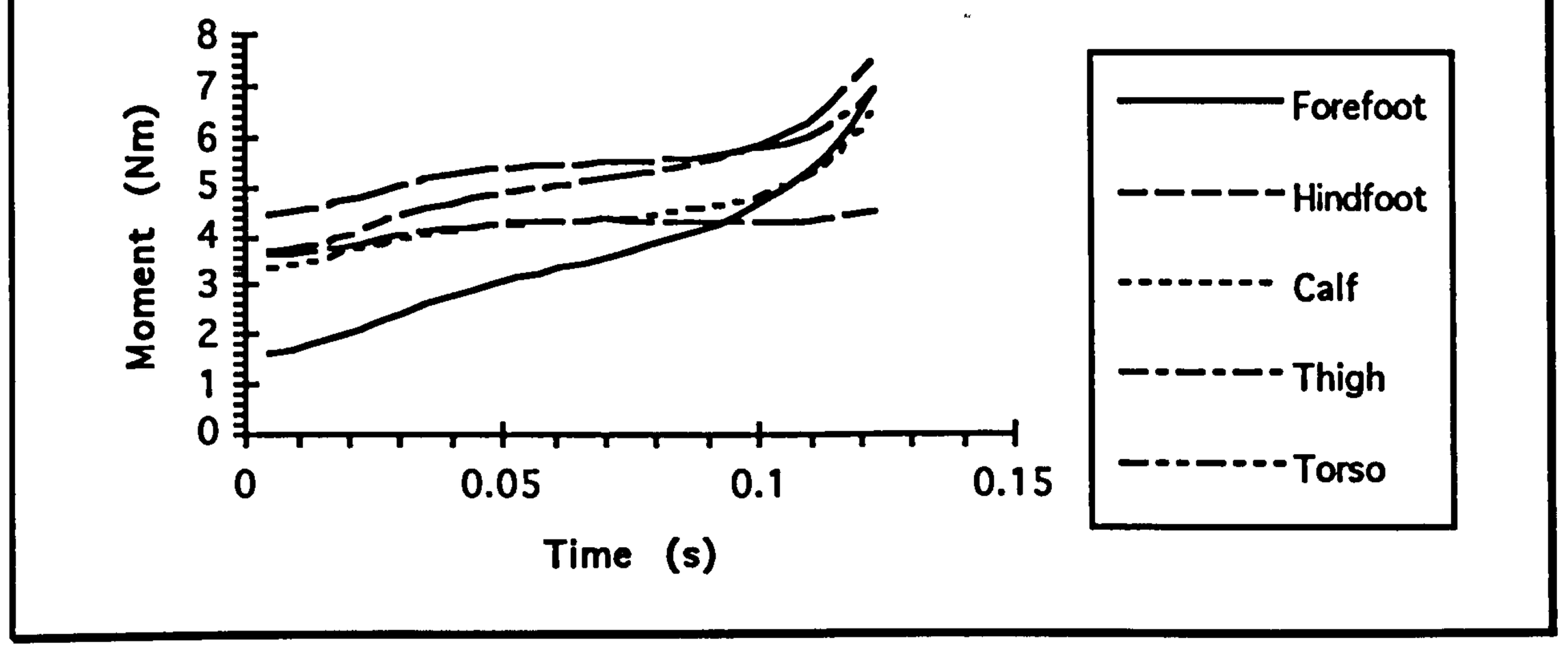
Galago gamettii



Graph showing the torque about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Galago garnettii*.

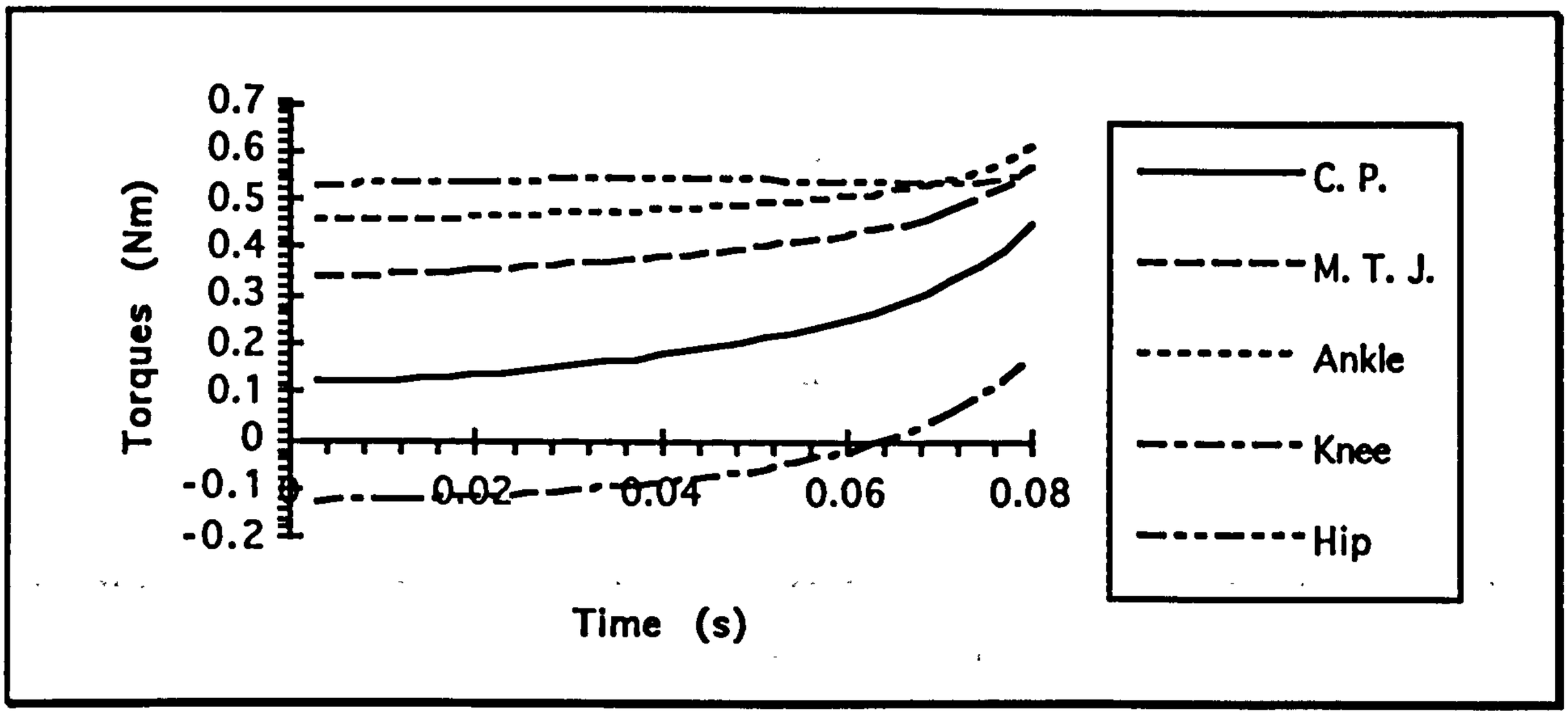


Graph showing the power about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Galago garnettii*.

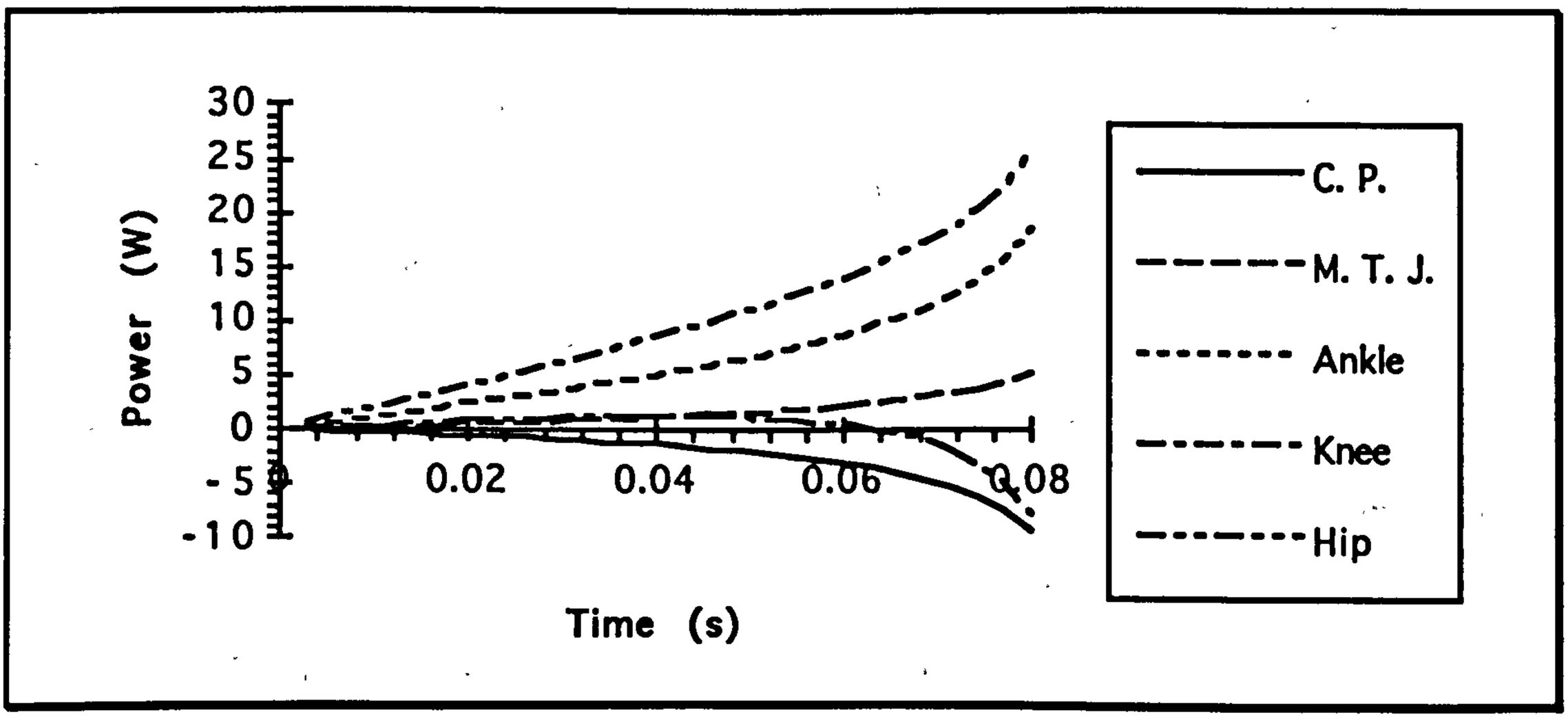


Graph showing the bending moment about the segments in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Galago garnettii*.

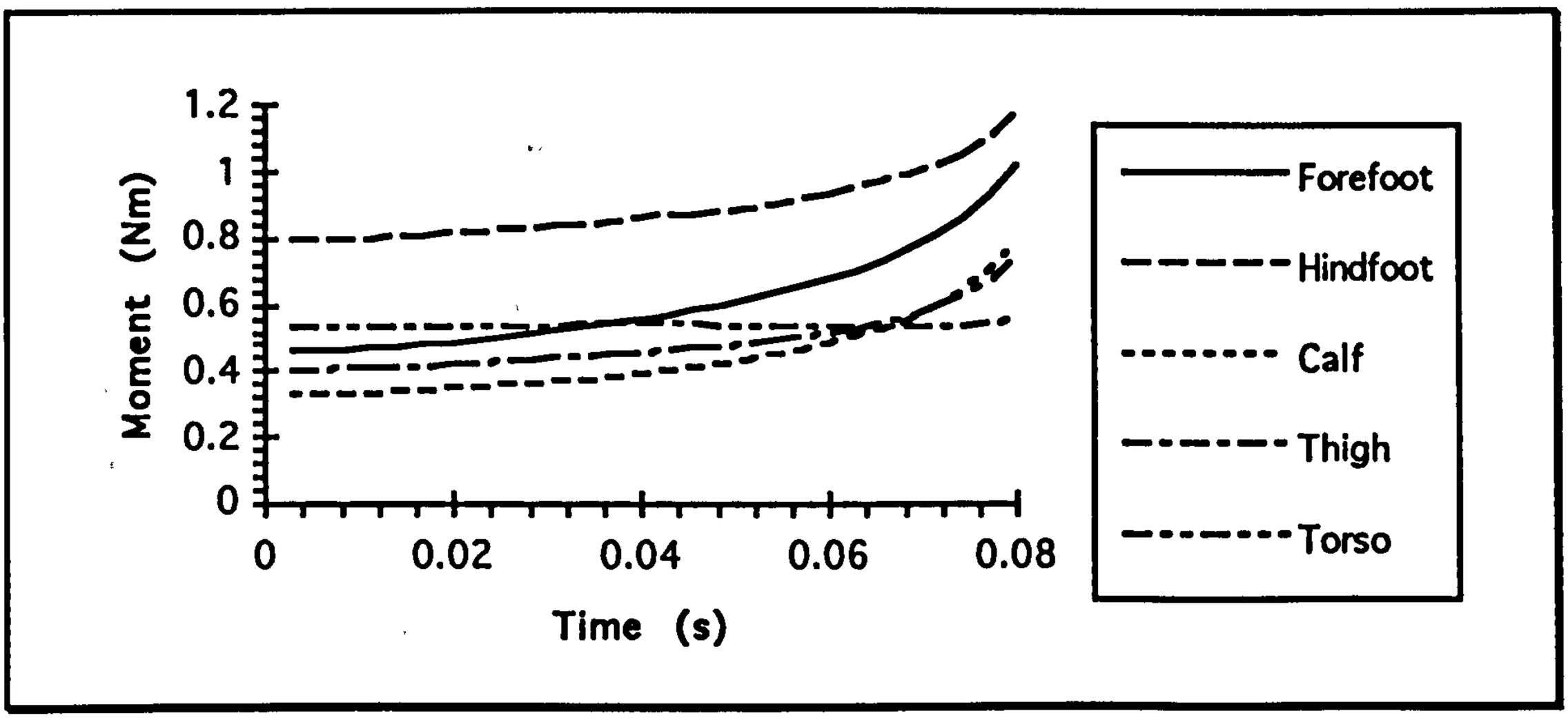
Galago moholi



Graph showing the torque about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Galago moholi*.



Graph showing the power about the joints in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Galago moholi*.



Graph showing the bending moment about the segments in the hind-limb for a simulated 1 m jump at a takeoff trajectory of 45° in *Galago moholi*.

One of the very important assumptions of the model is the smooth extension at each joint in the hind limb. This is certainly an over simplification. The joints do not all extend together, and this will effect the results quite considerably. Nor is the actual force applied constant and this again would alter the shapes of the curves. The torque curves can be visually divided between two groups: *Galago moholi* and *Mirza coquereli* lack the two stage increase of torque with time and do not have the constant torque plateau seen with the other animals. Whether this is indeed a feature of the limb configuration or simply due to the takeoff position chosen is unclear. The power curves are much more convincing, with the power generated about each joint increasing almost linearly with time as the animal's velocity increases and the forces required remain constant. As mentioned before, the upward curve at the end of the takeoff is due to the effect of internal energy to rotate the segments in the hind limb.

More importantly, all the curves indicate the predominance of the hip joint in powering the leap. It generates the highest powers, and, from

the area under the power/time curve, does the most work. This is not unexpected from a design point of view. The hip is the most proximal joint, so that having a large muscle mass around it will have less effect on the total moment of inertia of the limb. However, at least in the case of *Galago moholi*, the *vastus lateralis* muscle is by far the largest muscle in the hind-limb, but is a knee extensor (Günther 1989). This would lead to the conclusion that the prime motive force for a leap in this *Galago* species would be extension around the knee joint which contradicts the findings from the model. However, of all the animals simulated, *G. moholi* is the only one where the knee performs any appreciable amount of positive work. In the other animals, only negative work is performed about the knee joint. In a case where the main motive muscles in the leap were the *gastrocnemius* muscle which extends the ankle and flexes the knee, and the hamstrings, which extend the hip and flex the knee, this would be unsurprising since these two joint muscles would allow the required energy transfer so that the negative work is simply subtracted from the positive work required at the ankle and hip (Wells 1988). Verification of this requires the contraction forces and movements to be measured or modelled in the muscles themselves.

The graphs also show a significant rotational torque about the contact point between the animal and the substrate. The centre of mass of the animal is clearly not directly above the support area and without some form of torque here the animal will fall flat on its face! The only way to reduce this torque would be for the animal to accelerate off in a parabolic path during takeoff with the toe-off point being timed to coincide with the moment when the animal is inclined to the desired trajectory. This will, however, leave the animal with an unwanted angular velocity, and require very much more precise timing and coordination during takeoff.

In practice, most of the required torque will probably be provided by the action of the forelimbs, and a combination of timing and torque about the forefoot used to obtain the correct trajectory and minimize any rotational velocity on takeoff. In addition, a torque can be applied by the hind-feet gripping the substrate. The importance of this torque applied to the substrate is one of the things that cannot be measured with standard force plates, since they can only record the torque about a vertical axis.

Model Performance

To review the results of this exercise: does the predictive leaping model, as described here, provide any useful insight into the mechanism of leaping, and how might it be improved in the future? It does produce sets of movements that look convincing when animated on the computer screen, and it is relatively easy to calculate any of the desired dynamic results from the data produced. The results above were obtained by putting the data into my own gait analysis program, as described earlier, but they could equally have been used as an input dataset for a commercial, mechanical engineering dynamic analysis program such as DADS, or ADAMS. These typically allow much more detailed modelling of elastic and damping elements as well as simple force generators, and bending stresses are handled much more comprehensively, with links into finite elements packages such as ANSYS to cope with irregular shapes. My previous experience using one of these packages (DADS) in an attempt to analyze measured kinematic data was that they are not set up to cope with the kind of levels of uncertainty that are common when dealing with biological phenomena, and hence produce meaningless results. However, given the much higher quality of the data from a predictive model, I am quite sure that they would function perfectly.

The form of the model shown here is necessarily rather simple, but certain extensions could be incorporated which would improve confidence in the results obtained:

Firstly, real force plate data could be used to calculate the position of the centre of mass with time. Combined with a film record of the takeoff posture to get the starting position, this would allow a very much higher temporal resolution than film with very much less work (at least on the part of the experimenter: doing the calculations would involve quite a large amount of computing effort) and the precision of a force plate is very much higher than the accuracy with which joint positions can be measured.

Secondly, the time courses of the extensions of the various joints in the hind-limb can be 'tweaked' to more accurately reflect those observed in a subject animal. These will need to be coded into the model as functions describing in which parts of the takeoff phase they are more or less rapidly extending than normal. They could be weighted to act uniformly, or to unusually fast or slow in the beginning, middle, or end of the takeoff sequence.

And thirdly, the kinematic results produced by the model can be put into a commercial inverse dynamic modelling package which can be set up to use linear force producing engines attached between to the links and acting over pulleys rather than the much simpler rotational torque producing engines that I have used. Passive elements can also be incorporated, and it would also be possible, this way, to model the effects of the elasticity of the substrate.

Better mass distribution data would also be a great help. This is not particularly difficult to measure, but does render the animal body far less

useful for any other anatomical work, and as carcasses of these animals are particularly difficult to obtain, little information of this sort is available.

Discussion

This section discusses the points arising from the chapters of this thesis in the order they were raised.

Kinematographic Measurements

Video film was used exclusively as the source of the raw kinematographic data. This was viewed frame by frame, and points of interest were measured. This approach has a number of associated difficulties:

The major problem with video as a recording medium is poor resolution. Photographic film has an extremely high resolution, and each frame can be easily scanned into a computer at a resolution of 4096 by 4096 or higher. Compare this to a normal video signal where the best that can generally be obtained is 768 by 576; 512 by 512 is very much more common, and in either case this is the combination of two fields which need to be separated for still motion analysis. The images I used had resolutions of 512 by 256. In addition, the range of contrasts that can be recorded is also inferior for video.

The consequence of this is that, to get an image that shows enough detail to be measured, the field of view needs to be kept tightly cropped around the area of interest. This is a problem with leaping: the takeoff phase is relatively compact and the camera needs to be zoomed in so that the subject takes up most of the field. This means that the animal very quickly leaves the field of view after it has taken off. Whatever technique is used, differentiation loses several frames from each end of the sequence, so unless the film includes at least 2 frames after takeoff, no velocity or acceleration data can be obtained for the takeoff point. A number of my sequences were rendered useless because of this.

In addition, the ability to use video cameras in low light conditions is something of a mixed blessing. While the camera will indeed continue to function, the quality of the resultant image degenerates enormously. The increased gain required produces a lot more noise on the image, and because the light level is uniformly low, the contrast of the image falls. Compensating for this further increases the noise level. There is, in fact, no substitute for proper lighting. Even image intensifiers cannot avoid excessive image degeneration with current technology levels, and so for work which requires the level of detail required for accurate still frame measurement, adequate, well placed lighting, is essential.

Filming rate is also an important variable but it needs to be assessed in the light of experience. Slow filming rates will tend to underestimate velocities and accelerations. High rates will lead to greater inaccuracies in velocities and accelerations due to measurement errors occurring at higher effective frequencies. In addition, measurement is more time consuming. The main advantage of using higher filming rates is that it allows for more extensive smoothing, and the precision with which events such as toe-off can be measured is greater.

Leaping Trajectory

The results for leaping trajectory are unequivocal. The animals I studied did not leap at a trajectory of 45° as had been expected. This has important implications. It implies that energetic efficiency is not of prime importance for these animals and that they are prepared to forego the benefit of a reduced energy requirement in favour of some other benefit.

From the present study, there is no way of quantifying the other costs and benefits associated with the choice of leaping trajectory, but there are a number of possibilities:

Firstly, flatter trajectories are quicker. Therefore by leaping at a trajectory less than 45°, a prosimian can move through its habitat faster. This will allow it to spend less time travelling between discrete food patches or from sleeping sites, giving it more time for beneficial activities such as foraging, territory defence and courtship. There is a tradeoff here: the faster the animal moves, the more expensive it becomes to move a unit of distance, and there is only limited benefit to be gained from the time saved. To investigate this further, it would be necessary to derive cost/benefit functions for the various activities that the animal performs with respect to the amount of time it spends doing the activity. For example, an animal's territory size or quality (for example, its food density) may be directly related to the amount of time it has available for territorial defence behaviour. This has been shown to be the case for the golden-winged sunbird (Gill and Wolf 1975). Knowing all these cost functions would allow an optimum travelling speed to be calculated.

An associated factor to increased travelling speed for a flatter trajectory is that the time spent in the air is shorter. In general, raptors are the primary predator of small (<2kg) arboreal primates (Cheney and Wrangham 1986). However, because the presence of an observer tends to reduce the risk of predation, predation tends to be inferred rather than observed. For example, in a two year study of *Galago moholi*, for a population of 75 individuals, 3 predations were observed, 7 were suspected and the predation rate was estimated to be in excess of 10 per year (Bearder and Martin 1980). Thus, one might suppose that the risks of aerial predation would be lower when moving through undergrowth,

but much higher when the animal needs to leap across an exposed discontinuity in the substrate. It may therefore be extremely important to be in flight for as short a period of time as possible to minimize the danger. This predation argument would also go some way to explain the leaping trajectories chosen by *Galago moholi*. This animal was observed to leap at a variety of trajectories, though it was the only animal to average about 45° for all its jumps. This variation could be a mechanism to produce leaps that are more difficult for a predator to predict. If the prey animal always uses the same trajectory, then the raptor, seeing the animal preparing to leap across a gap, could judge the flight path of the animal and intercept it more easily.

In addition, animals that habitually live in the undergrowth may be so used to having to leap in a flat trajectory to avoid other branches that even in situations where there is room they may be insufficiently behaviourally flexible to choose to use a more energetically efficient trajectory. Indeed, they may be so specialized for their chosen environment that they can no longer get into the correct posture for a 45° leap, or again, doing so may be so unnatural to their bodies adapted to flat trajectories that they may not be able to leap efficiently at this angle. While this may be the case for frogs where leaping for height gain is unimportant, it seems unlikely for the animals studied here since all were observed leaping nearly vertically simply to gain height on numerous occasions.

There may be other environmental effects. It is more expensive to leap from a substrate that distorts appreciably from the forces exerted during takeoff due to the energy lost performing the distortion. In the case of an animal leaping from a horizontal branch, this distortion is most likely to be a shear occurring due to the bending moment from the reaction force

of the leap. This bending moment is only from that component of the reaction force that acts perpendicularly to the orientation of the branch. So, for a horizontal branch, the steeper the trajectory for a given leap distance, the more the branch is likely to bend.[40] So, for a given branch flexibility, the optimal leaping angle will be somewhat less than 45°. If the branch is not horizontal, then the picture becomes even more complicated. A branch inclined at 45° will get virtually no bending torque from an animal leaping off at 45°. Practically the only forces will be compressive, and assuming the branch does not buckle, this is likely to have very little effect since the force required to appreciably compress a branch longitudinally is very high indeed. The effects of substrate are likely to be rather more important for larger animals due to the larger forces involved, and may well encourage these animals to use larger supports closer to the main trunk of the tree than to venture out to the periphery when contemplating leaping. This is precisely the sort of behaviour seen in *Indriidae*, the largest and the most habitual leapers among the prosimians (Oxnard et al. 1990).

The other aspect of leaping trajectory concerns performance. All the animals, apart from *Galago moholi* which habitually leapt at 45°, leapt at closer to 45° as the distance they were jumping increased. Given that there are reasons to choose flat trajectories, this increase would be expected. For longer leaps, the animal has no choice but to consider the importance of efficiency because there are limits to the possible amount of energy an animal can put into a leap, and to get as far as possible on that amount of energy requires the animal to use the optimal energy

---

[40]The Y component of the reaction force is actually approximately proportional to $\frac{\sin\theta}{\sin 2\theta}$

efficient trajectory. For all these arboreal primates, it is easy to envisage circumstances when being able to manage an extremely long leap is enormously advantageous. For instance, if there is a large gap between two trees, a single leap is almost certainly a far preferable option than laboriously climbing down one tree trunk, running across the ground, and climbing up the other tree. In general, the ground is an extremely dangerous place for these animals, and the loss of potential energy means that this option is going to be energetically costly.

However, the substrate can also effect maximum performance. If the animal needs to pause before a long leap, which is the case for all the animals I watched when they were at all uncertain about the jump, then it cannot retain kinetic energy from a run up. However, provided the elastic properties of the branch are suitable, it can use it as a springboard, storing energy from a small preparatory bounce to add to the final larger leap (Günther et al. 1991). This is expensive because the branch will absorb an appreciable proportion of the energy from this first jump, but it may allow a greater maximal leap. In addition, the preparatory bounce will store energy in internal elastic structures, but although this has been found to be important in, for example, kangaroo rats (Biewener et al. 1981), its relative importance in prosimians is, as yet, unknown.

Apart from the changes in trajectory, there do not appear to be any other major alterations in the style of leaps with distance. Physics requires an increase in force and a concurrent reduction in the duration of the takeoff phase with increasing leap distance and this is exactly what is seen. There is an increase in the extension distance of the hind-limb, but this is not marked, and can be considered to be fairly constant over the jump distances observed. The fact that the observed parameters agreed reasonably well with the predicted ones is indicative that the

measurements obtained were correct. The values for forces, however, depending as they do on differentiating the measurements twice to calculate accelerations, are rather less certain.

The obtained force curves show an increase in force on the substrate in addition to body weight while the animal is flexing its hind-limbs in preparation for takeoff. This must be due to the animal allowing its centre of mass to fall due to the force of gravity. It can then apply a force to decelerating this fall and subsequently to accelerate the animal upwards in the takeoff phase of the leap. One can only assume that this is a mechanism to pre-tense the leg muscles by getting them to do some negative work before they do useful work in takeoff. This will overcome the problem of the relative slow build-up of tension in muscles: mammalian muscle fibres take somewhere between 10 and 100ms to generate their maximum tension depending on the speed of the muscle fibres (Schmidt-Nielsen 1983). It will also give them the chance to store some energy in elastic structures in the hind-limb without the action of antagonistic muscles to prevent movement. It is quite possible that antagonistic muscle action is also involved, but inverse dynamic analysis can only give information on resultant forces and gives no information as to how they are made up.

Species Differences

Most of the differences observed between the different species can be attributed mainly to their mass difference. However *Galago moholi* stands out from all the others in a number of ways. Firstly, it was seen to leap habitually at the energy efficient angle of 45°. Secondly, the shape of curves produced by the predictive model for this animal was qualitatively different from the others: specifically the minimum in the total work

against extension graph, and the absence of appreciable amounts of negative work about the knee joint. Both these observations fit with the fact that *Galago moholi* is a far more enthusiastic and frequent leaper than the other species (Oxnard et al. 1990). This may mean that energy efficiency is a greater concern. *Cheirogaleus major* is noteworthy in being the least capable leaper of the group. This is indicated by the fact that it needs to change its leaping style to a more energy efficient one at shorter distances than the other animals.

Scaling Models

There are various models used to explain the changes in body proportions with size. Leaping, however, is not a continuous activity like walking or running, so it does not have a characteristic velocity, and dynamic models, such as the 'dynamic similarity hypothesis' (Alexander and Jayes 1983) are not applicable. However the geometric (Hill 1950), elastic and breaking strain (McMahon 1973) similarity models are applicable. None of these make any difference to the predictions about the power relation between force and time and body mass: they all predict the same relationship which the experimental data fit reasonably well. However each does predict a different rate of change change in hind-limb extension distance with mass. There is conflicting evidence about whether the geometric similarity model is a better predictor of limb length than the elastic similarity model. Comparison of limb bone dimensions from animals as different as shrews and elephants has indicated a geometric relationship (Alexander et al. 1979), but analysis of other datasets has supported the elastic similarity model (McMahon 1984). In fact, none of them explain the observed results in this study at all well. It is not an exhaustive sample, but the indications are that the extension distance of the hind-limbs increases more rapidly than would

be predicted by any of these models. These results agree with more general observations on hind-limb length in prosimians (Emerson 1985). which indicate that the larger animals are more highly adapted for maximum leap distance.[41]

In this context, it must be remembered that all the scaling models predict that the maximum leap distance for similarly designed animals is the same irrespective of body mass. By having longer than expected hind-limbs means that larger animals can leap further. This does not seem to be an unreasonable state of affairs, but why would a larger animal need to be a able to leap further? By expending adaptive effort on being a better leaper, it must be suffering in other respects, such as being a less efficient walker, or a less rapid runner, so there must be reasons for this increased capability. Increased limb length is not going to effect the cost of leaping very much[42], but it will increase the length of the maximum leap that can be achieved. If the main role for long leaps is for crossing gaps in the substrate, then we may postulate that the discontinuities for larger animals are themselves larger. The tree spacing will be the same for all sizes of animals, but the size of the gap between trees depends on how far out along branches an animal can get. Smaller animals can get much further before they run the risk of bending or breaking branches, and they need to get less far towards the trunk of the target tree in order to get to a support suitable for landing on. Depending on the diameter

---

[41]There are important limitations in this line of reasoning. There may be no adaptive significance at all in the extra length of the hind-limbs of larger prosimians. The small ancestral leaping form may have obtained its long legs by an increase in their growth rate during infancy. This could lead to an even greater increase in limb length in subsequent larger forms due to their increased duration of infancy. If this extra hind-limb length has no adaptive value (that is, it is neutrally selected) then it will tend to persist.

[42]If it has any effect, increasing the length of the hind-limbs is most likely to decrease the energy efficiency of leaping shorter distances. The main excess cost is due to the internal energy of the rotating limb segments. The internal energy is proportional to the moment of

distribution of the branches of the tree, this may make a very big difference in the effective distance that the animal needs to cross. As mentioned before, the largest leaping prosimians, the *Indriidae*, leap preferentially from larger branches and from the trunk. This may well be because they are too big to reliably leap from branches away from the trunks. This sort of behaviour will appreciably increase the distances involved. Also, the smaller animals in this study can easily move through undergrowth tangles rather than leap across gaps much of the time. This positive allometry in hind-limb length appears to be peculiar to prosimians. Data for other jumping mammals, and even including other jumping vertebrates show a straightforward geometric increase in hind-limb length with body mass (Emerson 1985).

Predictive Modelling

Predictive models, in all their various forms, provide, in my opinion, one of the best way forward for studies of locomotion. Obviously their results need to be compared with those measured by more traditional methods and the models continuously refined to produce as convincing a set of output data as possible. Their advantage is that all the assumptions made have to be clearly set out at an early stage and their effects can be seen directly. Although, with a great deal of care, and extremely precise measurements, it is possible to calculate the torques required around the joints of the hind-limb by using experimentally derived kinematic data and inverse dynamic analysis, this does not necessarily reveal very much about the underlying mechanisms. However, if the same results can only be obtained with a predictive model using particular goal criteria, then

---

inertia of the limb segment and this depends on the square of the length. Longer limbs are likely to have a higher moment of inertia per unit mass because of this.

you have some evidence that the goals chosen might well be those that are have been selected for in the animal.

In addition predictive models are the only way to answer classic "What if" type questions. What if a loris, which is never seen to leap, did actually try to leap a metre? Predictive modelling would allow us to calculate the required torques generated round its hind-limb joints and the bending stresses applied to its skeleton. Then, if the answer to the question is that its tibia would break, or that its hip extensor muscles would not be able to provide the necessary power, the likely reason for not being able to leap would be clearly identified.

Similarly, a sub-fossil prosimian, such as *Megaindri* could be made to leap a variety of distances to see how its maximum power and torque requirements compare with those of other prosimians. This will allow the estimation of its possible maximum leap distance, which is extremely informative about its lifestyle.

Extending the idea further, using more sophisticated models, the effects of mechanical units other than simple links and torque generators could be seen. Does having an elastic element in the Achilles tendon lead to greater efficiency/performance? What feedback mechanisms are required to produce the observed movement in a controlled fashion? The effects of altering the input parameters could be analyzed by using Monte Carlo approaches, where each of the parameters is sampled from a range of possible values and the model is run a large number of times.

Conclusion

In conclusion, the mechanics of leaping in prosimian primates are not as straightforward as might have been thought. The expected optimal energy efficiency model is quite clearly not generally true except in certain special circumstances. The animals are seen to leap in flatter trajectories almost certainly due to the effects of non-mechanical factors. The exception to this is *Galago moholi*, which does appear to leap efficiently with respect to energy consumption.

The detailed internal workings of the limbs during a leap are much closer to those predicted by simple biomechanical requirements. The centre of mass of the animal moves in a straight line during the takeoff phase of the leap. Because of pre-tensioning of the muscles immediately before the takeoff, the force applied to the substrate is relatively constant during takeoff, though it falls off very rapidly as the limb becomes fully extended. Predictive modelling reveals that this is because attempting to get any worthwhile push at the limit of limb extension is unrealistically expensive because of the amount of internal energy required to rotate the segments of the hind-limb. The degree of extension of the hind-limb is not greatly increased with leap distance, but the effect on the takeoff duration and peak force are very much as predicted by simple mathematical analysis.

The extension distance increase with mass is noticeably bigger than would be predicted by any of the popular scaling models. This non-geometric scaling indicates that there is some selective pressure for larger prosimians to have longer hind-limbs. One possible reason for this is that the larger animals have further to leap since they can only leap from large supports that are closer to the trunks of the trees.

The predictive model with its goal oriented approach does appear to give sensible values for leap parameters. It indicates that leaping is mainly hip driven and that for all the animals, except again *Galago moholi*, only negative work is performed around the knee joint. It also shows that torque about the takeoff point is important for a stationary leap. Whilst there are limitations in how far the results obtained from the model can be taken, it could certainly be used to provide answers to a number of postulated questions about behaviours that are not observed and further work will be pursued in this area.

Technical Development

A suite of programs was developed for this project. The main gait analysis was done using a program called **gap** (**g**ait **a**nalysis **p**rogram) running on a Hewlett-Packard Unix workstation. The image grabbing was done using **digit** running on a PC clone. A program called **stretchpic** was used to enlarge the grabbed frames for analysis. This also ran on the workstation. The predictive modelling program, **Leaping Model**, runs on a Macintosh.

This section describes how to use each of these pieces of software, and the following section describes the technical details of the design and implementation of the programs. In addition, it also describes the specialized interface between the computer and the video recorder that was also developed specifically for this project.

User Guide

gap

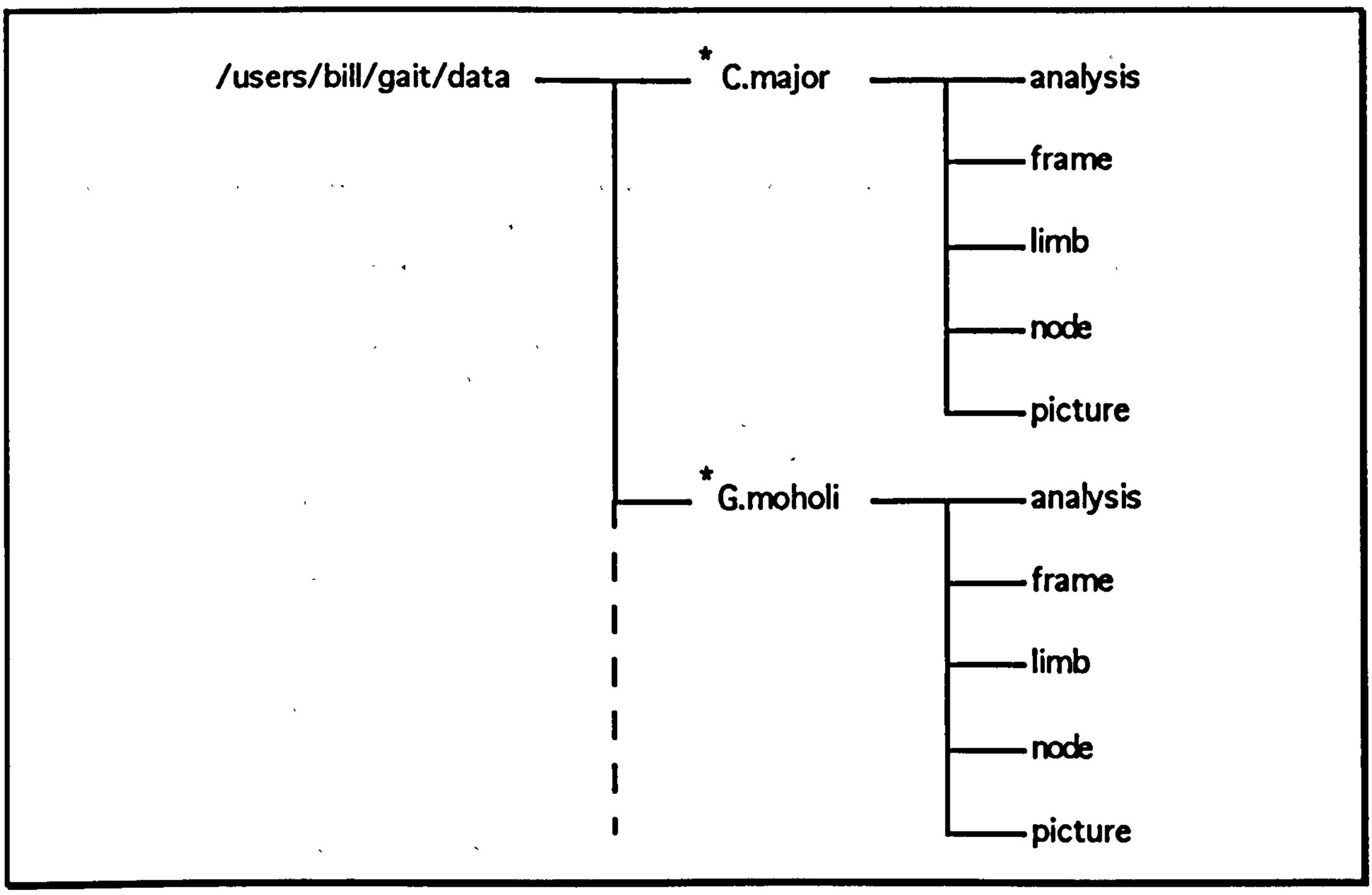
This program is the main kinematic analysis tool. It runs on the HP9000-350-SRX and the HP9000-360-TurboSRX computers fitted with 24 planes of display memory and at least 2 overlay planes. It also requires a knob box and a button box. Installation requires at least some familiarity with the intricacies of the Unix operating system, but using the program requires no knowledge beyond the ability to create directories and move around in the directory structure. The storage requirements for the program are minimal but with each image requiring over 750kbytes, considerable hard disk space is needed for image storage. The program requires the Starbase graphics system and the X window system (version 11) to be installed. In addition, recompilation requires a C compiler, a FORTRAN compiler and the NAG numerical libraries.

Installing the program

The program consists of a single executable file called **gap**. This should be placed in a directory on the user's path.[43] If the program needs to be recompiled, this is done by typing **make** in the directory containing the source code. The file **Makefile** may need to be edited so that it moves the new version of **gap** to the correct destination. If system-wide access to the program is desired, it should be put into the **/usr/bin** directory since this is on each user's default path.

[43]The path is a list of the directories that are searched for an executable file. It is set through the environment variable PATH.

The program must be run from a directory containing the following sub-directories: **analysis, frame, limb, node, picture.** These are where the program expects to be able to put its output files and read its input files. This structure is necessary to keep track of all the files that are involved in an average analysis session. I used a different directory for each animal I was studying, and each of these contained the five required sub-directories.



An example directory structure for **gap**. The program should be run by typing:

gap↵

whilst in one of the directories marked with a *. Alternatively, it can be run from elsewhere, and the working directory can be changed (using the "change working directory" option) to one of the marked directories.

For analysis, the program requires: a model definition file; a calibration image file and a set of image files representing sequential frames in the film to be analyzed. The model definition file should be put in the **limb**

sub-directory and the image files (calibration and sequence) should be put in **frame**. If sequence position data is to be used that has been calculated from elsewhere then it should be put in **node** which is also where **gap** will put any it produces. The directory **analysis** is used for any data export files and **picture** is used for screen dumps of animations and graphs.

In Unix, all user interaction is performed through special device files that are created in the directory **/dev** using the program **mknod**. The program requires the following special files to be set up:

Device File	Description
/dev/crt	Accesses the image planes of the display.
/dev/bbox	Accesses the button box
/dev/knob1	Accesses the bottom row of three knobs
/dev/knob2	Accesses the middle row of three knobs
/dev/knob3	Accesses the top row of three knobs

In addition, X windows needs to be set up to run in the overlay planes of the graphic display. This involves the creation of a **/dev/crto** special file and setting up an **X*screens** file with **/dev/crto** as its first line. X itself requires special files for the mouse and keyboard. These are created automatically, but **/dev/locator**, for the mouse may need to be altered when the knobs and buttons are attached. All the special files needed depend on the exact configuration of the hardware, including the order of devices on the keyboard bus and the presence or absence of the dongle module.[44]

[44]Full details about installation of Unix device drivers, Starbase and X11 can be found in the relevant HP 9000 series 300 manuals (Hewlett-Packard 1988a,b,c,e)

Also, the following Unix environmental variables need to be set: **SB_OUTDEV** to **/dev/crt** and **SB_OUTDRIVER** to **hp98721** or **hp98731** depending on the graphics accelerator being used. These are the device and driver used in the Starbase **gopen** statement and they are usually set in the user's **.profile** file, or they can be set by hand at the start of each session.

For example:

```
export SB_OUTDEV=/dev/crt
export SB_OUTDRIVER=hp98721
```

Setting up a model

Setting up the model requires the creation of a **.limb** file. This is a text file that is used by **gap** to define the nodes and segments in the model, how they are linked together, and the mass properties of the segments. However, before this file is created, the model itself needs some consideration:

First of all, the positions (nodes) on the subject that are to be measured need to be named and numbered. Then the nodes need to be linked up with segments. Nodes that link segments are joints. Each segment is defined by two nodes: one at each end. Measured nodes do not have to be associated with a segment, but each segment must be defined by two nodes. A node can be used as a defining point for any number of segments. The segments also need to be named and numbered. The program calculates position information for nodes and angle information for segments.

The **.limb** file has been designed so that it is easily produced or read by a FORTRAN program where file manipulation in text mode is somewhat limited, but normally, it is produced using a text editor such as **vi** or **emacs**, or alternatively, by any word processor that can save files as unformatted text. The line spacing is not at all important since all the character strings are delimited by single quotes, but the suggested line spacing makes the file relatively easy to read.

Here is an example **.limb** file:

```
'Microcebus murinus limb file (Bitters)'
10
0    'Forelimb tip'
1    'Elbow'
2    'Shoulder'
3    'Toe tip'
4    'Mid-tarsal joint'
5    'Ankle'
6    'Knee'
7    'Hip'
8    'Nose tip'
9    'Tip of tail'
9
0    'Lower arm'    0    1    3.13E-03    5.00E-01    2.35E-07
1    'Upper arm'    1    2    2.88E-03    5.18E-01    1.77E-07
2    'Forefoot'     3    4    8.75E-04    5.73E-01    1.26E-08
3    'Hindfoot'     4    5    8.75E-04    4.78E-01    1.32E-08
4    'Calf'         5    6    3.38E-03    4.01E-01    3.57E-07
5    'Thigh'        6    7    1.03E-02    4.47E-01    1.26E-06
6    'Head'         8    2    6.38E-03    5.00E-01    7.30E-07
7    'Torso'        2    7    3.26E-02    5.00E-01    1.82E-05
8    'Tail'         7    9    2.19E-03    3.78E-01    2.33E-06
```

The first line contains a title that is used to provide more information about the contents of the file. It is delimited by single quotes[45]. The second line is an integer (10 here) that specifies the number of nodes in the model. The nodes are named on the next 10 lines: each of these lines starts with an incremental integral identification number running from 0

[45]The single quote is the ASCII value 39. This should not be confused with the opening and closing single quotes provided on the Macintosh system whose ASCII values are 212 and 213. These are not interchangeable, and do, in fact, look quite different: ' as opposed to ‘ or ’.

to 9, and is followed by the actual name of the node. The names are again delimited by single quotes. The next integer, in this case 9, is the number of segments defined. The 9 subsequent lines contain, first of all, an incremental identification number. This is followed by the name of the segment in single quotes, then the identification numbers of the two joints at the ends of this segment. For the purposes of the program, this segment is considered to extend from the first numbered joint to the second numbered joint. So, in this case, the lower arm, runs from the forelimb-tip (joint 0) to the elbow joint (joint 1). The next number on the line is the mass of this segment in kg, and the next is the position of the centre of mass as a fraction of the distance from the first joint to the second joint. In other words, a value of 0 would indicate that the centre of mass was at the first joint, and a value of 1 would indicate that it was at the second joint. Normally, this value is not too far from 0.5. The last number is the moment of inertia of the segment in $kg.m^2$.

All the values are required though dummy values can be inserted for the mass properties if they are not required. Zero should not be used as a dummy value since it will lead to divide by zero errors in the program, and 0.5 should be used for dummy centre of mass positions. The character strings have no significance within the program except as the names used as prompts to the user and as labels on the output data. Extra nodes and segments are added by changing the integers indicating the total number of nodes or segments and by adding extra node or segment description lines. As mentioned before, a single node can be used to specify more than one segment, or, indeed, no segments at all, and act simply as a position marker.

Only the first half of the file which defines the joints is used for the measurement and reconstruction phase, so if there is an error in the

segment description, this can be changed after measurement as long as the joint definitions themselves are not altered. The other area in which to be careful is in defining the position of the centre of mass. It is very easy to get the direction sense of this fraction wrong. It is the fraction of the distance from the first defined node in a segment to the second. The integers should be typed in as simple numbers (i.e. 1 rather than 1.0) but the floating point numbers can use any standard computer notation (1 1.0 or 1.0e0).

Running the Program

The program is run by typing:

```
gap↵
```

in the directory containing the **analysis**, **frame**, **limb**, **node**, **picture** sub-directories. It needs to be run from an **hpterm** window running in X11 in overlay mode. It uses the system command **xseethru** to open up a seethru window, so this should be available on the path. The window selection control should be set up to use the left mouse button (the default), since the right hand mouse button is used for all selection (both menu and measurement) in the program. Menus pop-up automatically, and windows re-size suitably for a 1280 by 1024 display. To switch to another program, regain control of the mouse pointer by choosing the shell option on the main menu.

Measuring the Film

Before anything else, you need to set up the global options (select "options" from the main menu) for the type of measurements that you are doing. There are 3 types of reconstruction: 2D; 3D with orthogonal cameras; 3D with any camera position. In addition, a fiducial point can be

specified for each frame. This is simply a point that is visible on all frames and can be used to make sure that the registration of each frame is constant. It is measured on each frame and if its apparent position has altered, then the program can shift every point it measures accordingly.

Calibration

Select the "digitize new sequence" option, and you will be prompted to load up an existing calibration file or to perform a new calibration. For the new calibration option, you will be prompted for an image name which will be displayed. Then you will be asked for a number of calibration points (and a fiducial point if appropriate) depending on the reconstruction option, and asked to type in their real world coordinates.

The 3D options assume that both of the required views are contained on a single image as obtained by grabbing a split field view. It does not matter which is chosen as picture 1, as long as it is consistent. The brightness and contrast of an image can be adjusted using the first two knobs.

After calibration, there is an option to save the calibration file.

Measurement

After calibration, you are taken directly to the measurement option. Here, you are prompted for each measurement using the names given in the model definition file. In 3D, the required picture is also requested. Measurement is simply a matter of moving the mouse pointer so that it is pointing to the required position on the image and pressing the right button. The program will acknowledge a point by drawing a marker at the measured position. Occasionally, due to the multi-tasking nature of Unix, it will miss a mouse click because the processor was busy elsewhere. This

can be minimized by having as little going on in background as possible. This should be done in any case to avoid sluggish performance.

There is no option to correct individual mistakes, but at the end of each frame, there is the option to repeat the whole frame, or to go onto the next one. There is also the option to finish the sequence and go back to the main menu. Once here, it is suggested that the sequence should be saved with the appropriate menu option.

Getting Output

Animation

Once a sequence is in the computer's memory, either because it has just been measured, or because it has been loaded up from a file, the animation option can be selected. This allows continuous, movie style display, or frame by frame stepping. There is also the option to fix a particular joint to the centre of the field of view.

When the animation is running various options are available from the knobs and buttons. These functions are displayed on the top corners of the screen. The top right displays each relate to the knob in the geometrically similar position. It is suggested that the user experiment with the use of the knobs to become familiar with their effect on the three dimensional views. In particular, moving the target and the view point positions produce effects that maybe somewhat difficult to get to grips with. The viewpoint can be considered as a camera that is moved around in 3D space (top 3 knobs) with its lens always pointed at the target point (middle 3 knobs). Thus any combination of viewing positions can be achieved. The camera and target positions are limited to a 4 m cube with the origin in the centre.

Analysis

The analysis selection brings up a large menu. These options are all the kinematic and general kinetic measurements that are available. For each one, the use can select the limb segment or joint of interest. Multi-line graphs are possible, but not more than 10 lines should be plotted on each one since this will lead to duplication of symbols.

Once a plot has been obtained on the screen, it can be output to a plotter, saved as screen dump, or the data that is represented can be exported in a format suitable for input into a variety of other programs. In particular the SAS export option is designed to duplicate the displayed graph by producing a SAS program that contains both the data and the required commands to produce the graph.

Analysis can be performed on raw or smoothed data as chosen by the analysis options menu. The user defined menu option contains specific add-ons. At present this consists of the inverse dynamic procedures used in this thesis, but others can be easily added.

Menus

For reference, here is a complete description of the menus available in the current version of **gap**.

Main menu:

Selection	Description
Read limb file	Reads in a **.limb** file from disk.
Read node file	Reads in a **.node** file from disk.
Write node file	Writes out the current position data to a **.node** file on disk.
Digitize new sequence	Start a new digitization sequence.
Digitize additional frames	Add more frames to an existing digitization sequence in memory.
Display frames	Display an animated reconstruction of the data in memory. Goto display menu.
Analyze gait	Perform kinematic and kinetic analysis of the data in memory. Goto analysis menu.
View video frames	View stored images on screen. This does not affect any data in memory.
Set global options	Set program global options. Goto global options menu.
Shell to Unix	Goto the Unix command line prompt. Type: `exit↵` to get back to the menu.
Exit	Quit back to the Unix prompt. Also closes the display window and resizes the command window.

Display menu:

Selection	Description
Start sequence	Start the animation
Select options	Select animation options. Goto display options menu.
Exit	Exit back to the main menu.

Display options menu:

Selection		Description
Colour:	Copper	Select colour and reflectance properties for the surface modeller. This selection is red, metallic copper.
	Rubber	Matt red/orange rubber.
	Plastic	Bright, shiny red plastic.
	Obsidian	Gloss black.
	Pottery	Dull orange/brown.
	Brass	Golden metallic brass.
Style : Hollow		Select fill style. Hollow shows the outlines of the polygons used to construct the animation.
Raw data		Raw or smoothed data for the animation.
Variable limb radius		Not implemented fully, but switches the display to use truncated cones rather than cylinders.
Exit		Exit back to the display menu.

Analysis menu:

Selection	Description
Node position	Display node position data for selected nodes with respect to time.
Node velocity	Display node velocity data for selected nodes with respect to time.
Node acceleration	Display node acceleration data for selected nodes with respect to time.
Segment angle	Display segment angle data for selected segments with respect to time.
Segment angular velocity	Display segment angular velocity data for selected segments with respect to time.
Segment angular acceleration	Display segment angular acceleration data for selected segments with respect to time.
Segment lengths	Write out a file containing mean segment lengths.
Node locus	Display the locus of selected nodes.
Forces	Display the resultant forces acting on selected centres of mass of segments with respect to time.
Torques	Display the resultant torques acting on selected segments with respect to time.
Energies	Display the potential, linear kinetic and rotational kinetic energies of selected segments with respect to time.
Options	Select options for the analysis. At present, only whether to use raw or smoothed data.
User specific analysis	Perform specialized, user written analyses. Goto user specific analysis menu.
Exit	Exit back to the main menu.

User specific analysis menu:

Selection	Description
Simplified Quadrupedal Analysis	Perform inverse dynamic analysis on the simplified quadrupedal link segment model.
Predictive Model Analysis	Perform inverse dynamic analysis on the output of the predictive leaping model.
Toe tip and Body COM output	Write out toe tip and whole body centre of mass positional data to a file.
Exit	Exit back to analysis menu.

Global options menu:

Selection	Description
2d On	Switch between two and three dimensional data and reconstruction. In 2D mode, the Z values of any data are zeroed.
Flexible 3d reconstruction	Switch between orthogonal camera 3D reconstruction and flexible reconstruction using the DLT algorithms.
No fiducial marks	Switch on or off the requirement for a fiducial mark on each frame measured.
Set frame increment	Set the number of frames to be skipped when reading in frames incrementally.
Set filtration cutoff	Set the filtration cutoff frequency as a multiple of the framing rate.
Change working directory	Change the startup directory.
Smoothing	Switch between moving average smoothing and low-pass digital filtration.
Set smoothing number	Set the number of values over which to perform moving average smoothing.
Exit	Exit back to the main menu.

digit.exe

This program is used to grab sequential fields from a video film. It runs on a reasonably fast PC compatible (80286 or better processor) running MSDOS with a serial card (COM1) and a Matrox PIP-1024 board connected to a second monitor. The program requires minimal hard disk space, but each image requires over 100kbytes of storage so that appreciable storage space is required.

Installation

This program requires that the file **digit.exe** be copied into a directory on the path, or the path can be set to include the directory containing the program. **digit.exe** is produced by compiling **digit.c** and linking with the relevant Matrox libraries. These can generally all be kept in the same directory. The program is run by typing:

```
digit↵
```

at the DOS prompt.

The video interface needs to be connected to the COM1 port of the computer and the audio in and audio out sockets need to be connected to their counterparts on the video recorder. When dubbing, the output level from the interface is constant so that any auto level control on the video should be turned off, and the level set manually to the 0dB level. The output level from the video needs to be set experimentally along with the input sensitivity control in the interface to produce reliable results. Since the signal is internally clipped, maximum volume may work best here.

The PIP-1024 card needs to be installed normally and the video output from the recorder connected to channel 2. The board needs to be

installed at the default I/O location and memory address with interrupts disabled. The second monitor is connected to the board as normal.

Running the program

First of all, move to the directory where the image data is going to be saved and start the program. This will produce the following menu (items are selected by keying in their number):

Selection	Description
Write Soundtrack	Writes the timing sound track out to the video recorder.
Read and Display Soundtrack	Read the timing sound track from the video recorder and displays it on the computer screen.
Grab Single Frame	Grabs a single frame at a specific count number on the sound track.
Digitize Frames	Grabs and saves a sequence of fields from the video recorder based on counts from the sound track.
Adjust Brightness and Contrast	Adjust the brightness and contrast settings for the frame grabber card.
Exit	Quit back to DOS.

To digitize a series of fields from the video, the following steps need to be performed:

(1) Record a sound track on the video:

Rewind the video tape containing the sequence. Select the **write sound track** option from the main menu. Press play on the video recorder and switch on sound dubbing and press the space bar on the computer keyboard to start the production of the sound track. The computer counts the video frames and sends out a digitally coded count number to be recorded on the sound track. When the sound track has finished,

press the space bar to get back to the main menu. This process only has to be done once for the whole tape.

(2) Set up the brightness and contrast:

Choose the **adjust brightness and contrast** option and play through the video sequence. Adjust the brightness and contrast using the new menu options until the best picture is obtained on the screen attached to the grabber card, then select exit to get back to the main menu. This will need to be done once per session. It is worth noting down the brightness and contrast settings so that they can be duplicated on subsequent sessions.

(3) Choose the start count for the sequence:

Rewind the video to before the start of the sequence. Select the **read and display sound track** option from the main menu. Press play on the video recorder. After a short delay while the computer gets into sync with the video, the computer screen will display the timing counts. When the required start point is reached, press the space bar on the computer keyboard. The screen attached to the grabber card will show the frame that corresponds to the displayed timing count. To check whether the count before, or perhaps after, is a better start point, the **grab selected frame** option can be selected from the main menu, and the desired timing count keyed in. The video can then be rewound to before the start point and switched to play. The timing count will be displayed, and a frame will be grabbed at the required count number.

(4) Grab and save the desired sequence:

Select the **digitize frames** option from the main menu. You are then prompted for a file name. This should be up to 8 characters long, and is

the name each field is saved as, with an extension starting at **.000** and incrementing by 1 each time. Then key in the start timing count and the number of counts over which to digitize. Each count corresponds to 16 fields (about $^1/_3$ second). Then rewind the video to a point before the start of the sequence and press "play". For each count, the video needs to play through the sequence twice, since only 8 fields are stored on each pass. After these are grabbed, an on-screen prompt appears asking for the tape to be rewound. The tape needs to be rewound each time to a position at least 2 seconds before the start of the sequence since it takes about 2 seconds for the computer to re-sync to the timing track after "play" has been pressed on the video[46]. If, by accident, the tape is not rewound sufficiently, simply rewind the tape again and press "play". This may lead to the synchronization getting confused, but this is easily spotted since the timing counts displayed are nonsensical. If this occurs, just rewind and play until the correct numbers are displayed. This may take two or three attempts.

The end result is a sequence of incrementally numbered files on the hard disk, each containing a single field digitized at 256 grey levels and at a resolution of 512 by 256, with an aspect ratio of 1.333 for the whole image so that each pixel in the stored image is 1 unit wide by 1.5 units high.

stretchpic

The image files produced by **digit** have a resolution of 512 by 256. **gap** can display these files, but they will appear very small on the display and

---

[46]The easiest way to perform this repetitive rewind and play cycle is to use the shuttle and play controls on an edit controller. Alternatively, an in and out point can be set, and the required sequence of film can be cycled through using the review button.

will be horizontally stretched because the pixels on the workstation are square, and those in the image are rectangular. **stretchpic**, as its name suggests, stretches the image to 1024 by 768 pixels. Of course, this increases their storage requirement 6 fold, so for archive storage, images should be left in their unstretched form. The program currently runs under Unix, though, since it is a very simple program it could be recompiled to run under virtually any operating system.

Installation

The executable file **stretchpic** can be installed anywhere on the path. As with **gap**, for system-wide access, **/usr/bin** is a good choice. Recompilation simply requires running the C compiler on the source file **stretchpic.c**.

Running the program

The program is run by typing:

```
stretchpic file1 file2 … ↵
```

where *file1 file2* etc. are the names of the original image files. This will produce stretched image files which will replace the originals. Wildcards such as * and ? can be used to specify a group of files in the standard Unix fashion.

stretchpic is quite slow, so the suggested action is to copy a session's worth of image file from archive storage into a directory and to run **stretchpic** on all of them at once before the session starts. This may take an hour or so to complete, but will allow the measurement session to take place without interruption.

Leaping Model

This is the predictive leaping model program. It is given a set of parameters describing the required leap and the physical properties of the leaping animal, and it produces a set of positional data which have the correct kinematics to perform such a jump. This output data set is in the correct format to be read into **gap** for display and kinetic analysis. It has been tested on a Macintosh Plus running system 6.7 under multifinder, but should run with no problems on more recent versions of the system and with newer machines. The program fully supports desk accessories and MultiFinder, but it does not run in the background.

Installation

Copy the application file **Leaping Model** anywhere on the hard disk or floppies. If recompilation is required, use the the Symantec Think C project file **Leaping Model.π**. The resource file and the C source files are all in the same folder.

Running the program

The program can be started by either double clicking on the program icon or on one of the files produced by the program.

Program Icon

Model Icon

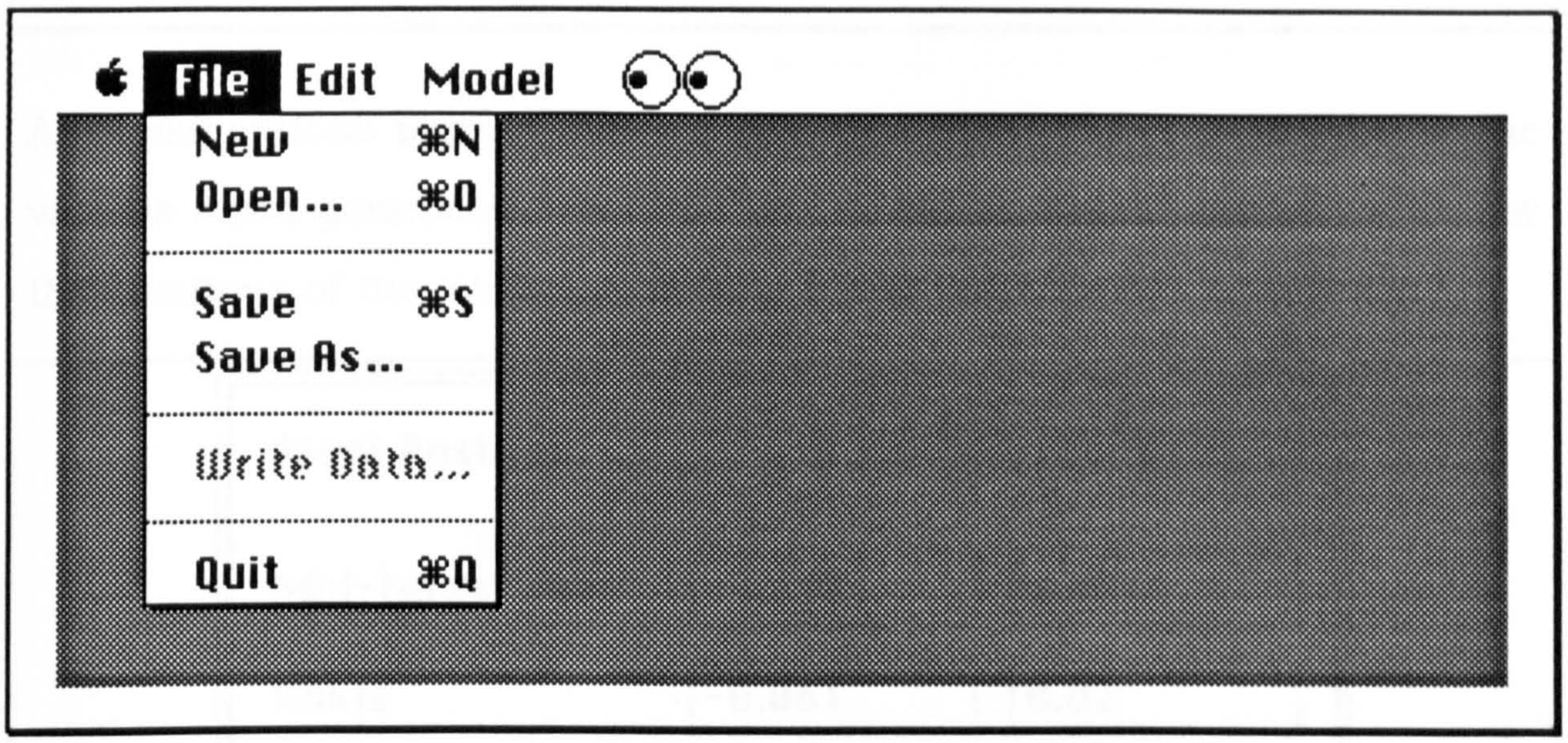
Results Icon

Macintosh icons for the predictive modelling program. Double clicking on the model icon causes that model to be loaded up immediately.

There are four main menu groups: , **File**, **Edit** and **Model**. The menu accesses the desktop accessories and other programs running under

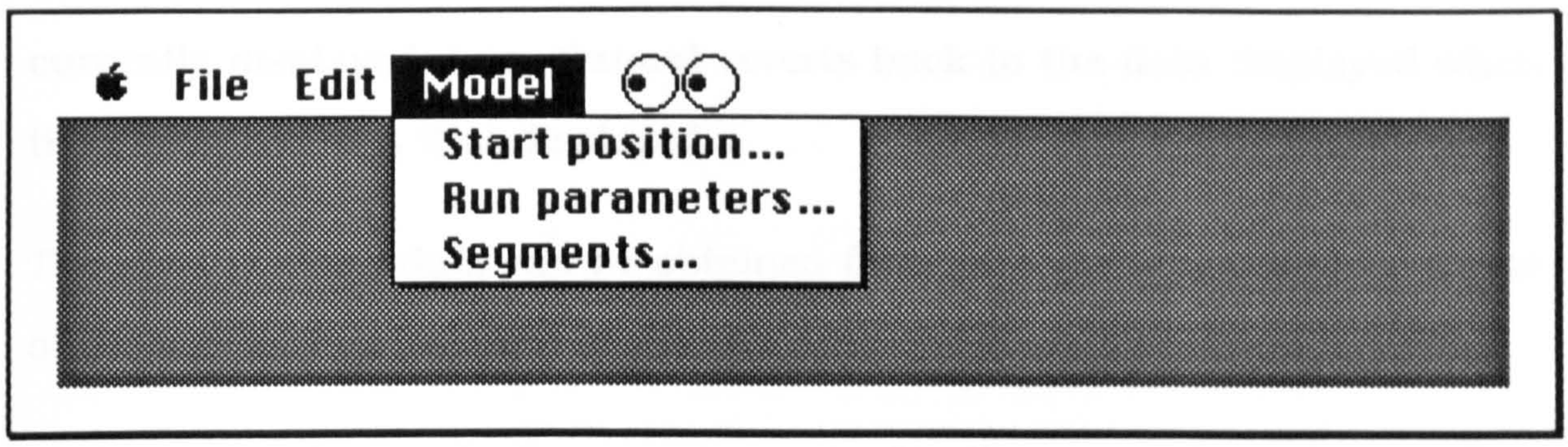
MultiFinder as well as the "about box" for **Leaping Model**. The **Edit** menu is only used for desk accessories and for number entry within fields in the dialog boxes.

The **File** menu has the following options:

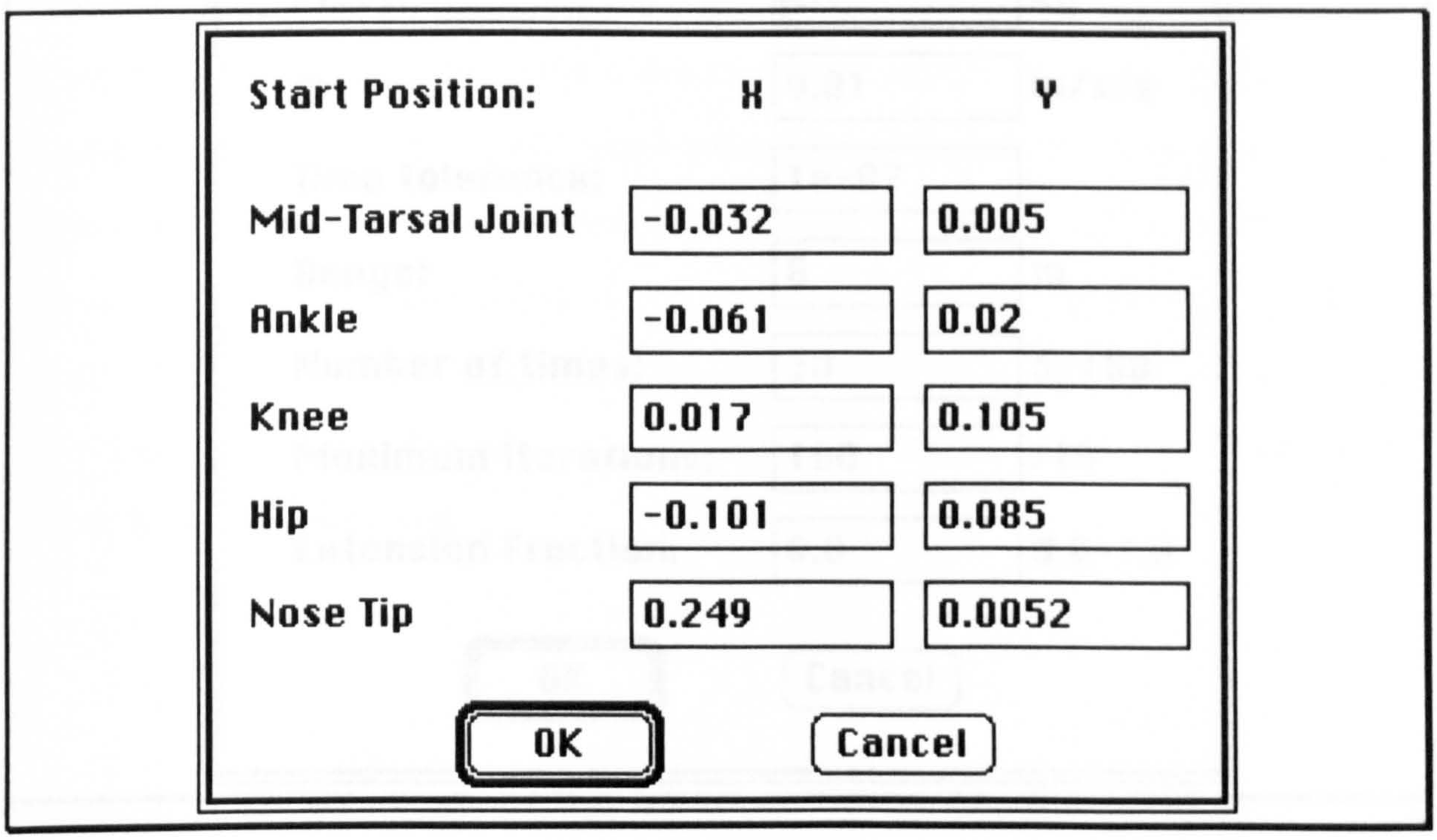


New deletes the current modelling data, and restores the values to their defaults. **Open** reads in an existing modelling data file. This is the file that contains the input data for the program. Only modelling data files will be displayed in the standard file dialog box. **Save** saves the current modelling data file with its current name if it has one. **Save as** saves the current modelling data file under a different name. **Write data** calculates the kinematic data from the current modelling data and writes it out to a file. It is greyed out, as in this example, until a modelling data file is opened, or new modelling data is entered. **Quit** exits the program. All standard Macintosh data traps are implemented so that any action that might cause the loss of data is queried.

The **Model** menu has the following options:



All these options pop-up dialog boxes that allow the user to type in the various input parameters required by the model. **Start position** is the for the positions of the joints in the fully flexed position:[47]



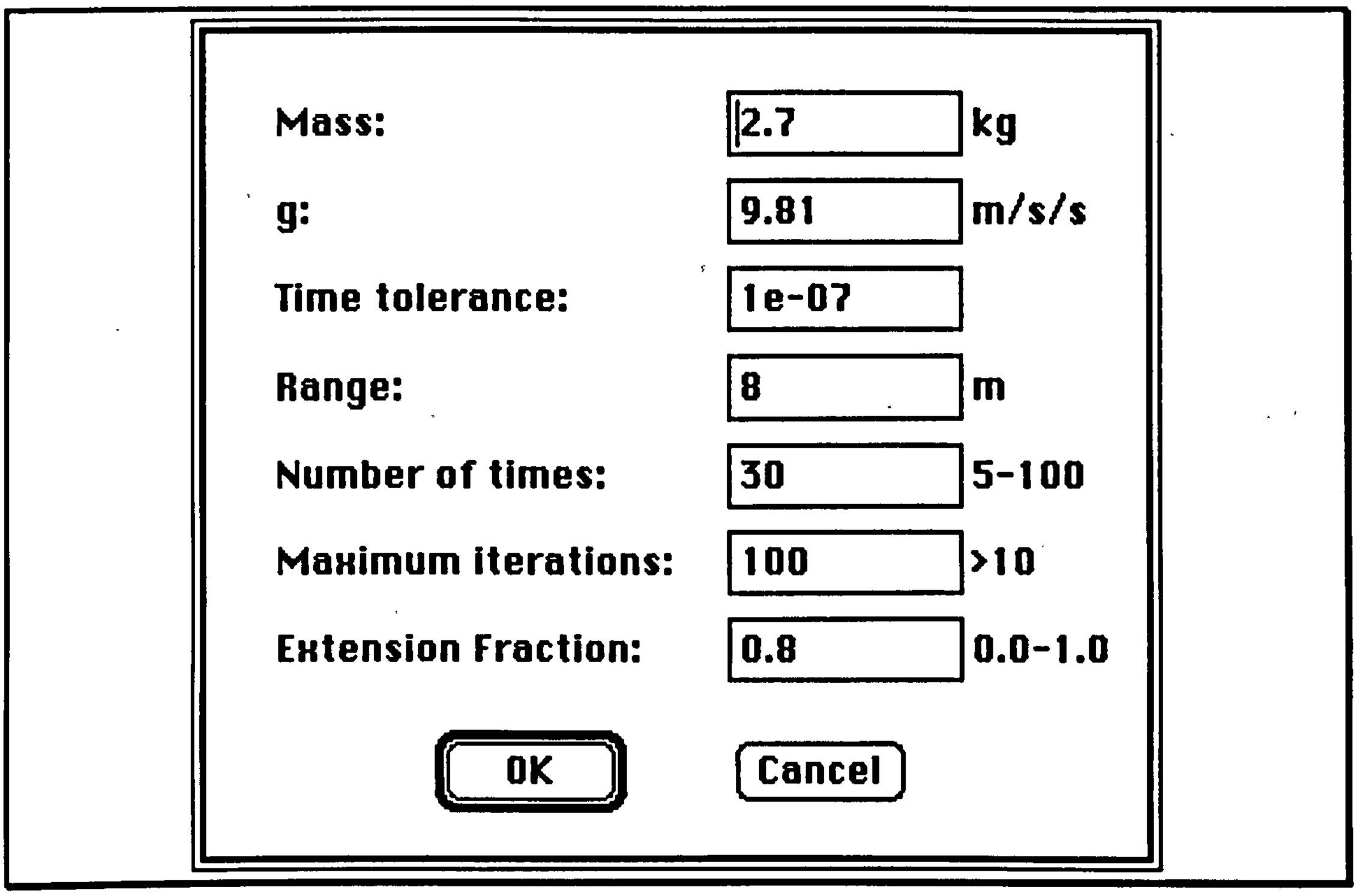
The toe tip needs to be at the origin, but otherwise, any coordinate system can be used provided that the units are in metres. The model is rotated so that the centre of mass is at a 45° inclination before calculation starts. As in all the dialog boxes, values can be entered in any of the standard numerical formats (1 1.0 1.0e0). The **OK** button accepts the

[47]The values in the dialog boxes shown are for the *Lemur catta* leaping model.

currently displayed data. **Cancel** reverts back to the data displayed when the dialog box was first displayed.

The start position data can be obtained from measuring a still video frame of the animal just before it starts to leap.

Run parameters sets the requirements for the computer program that are independent of the animal being modelled (generally):

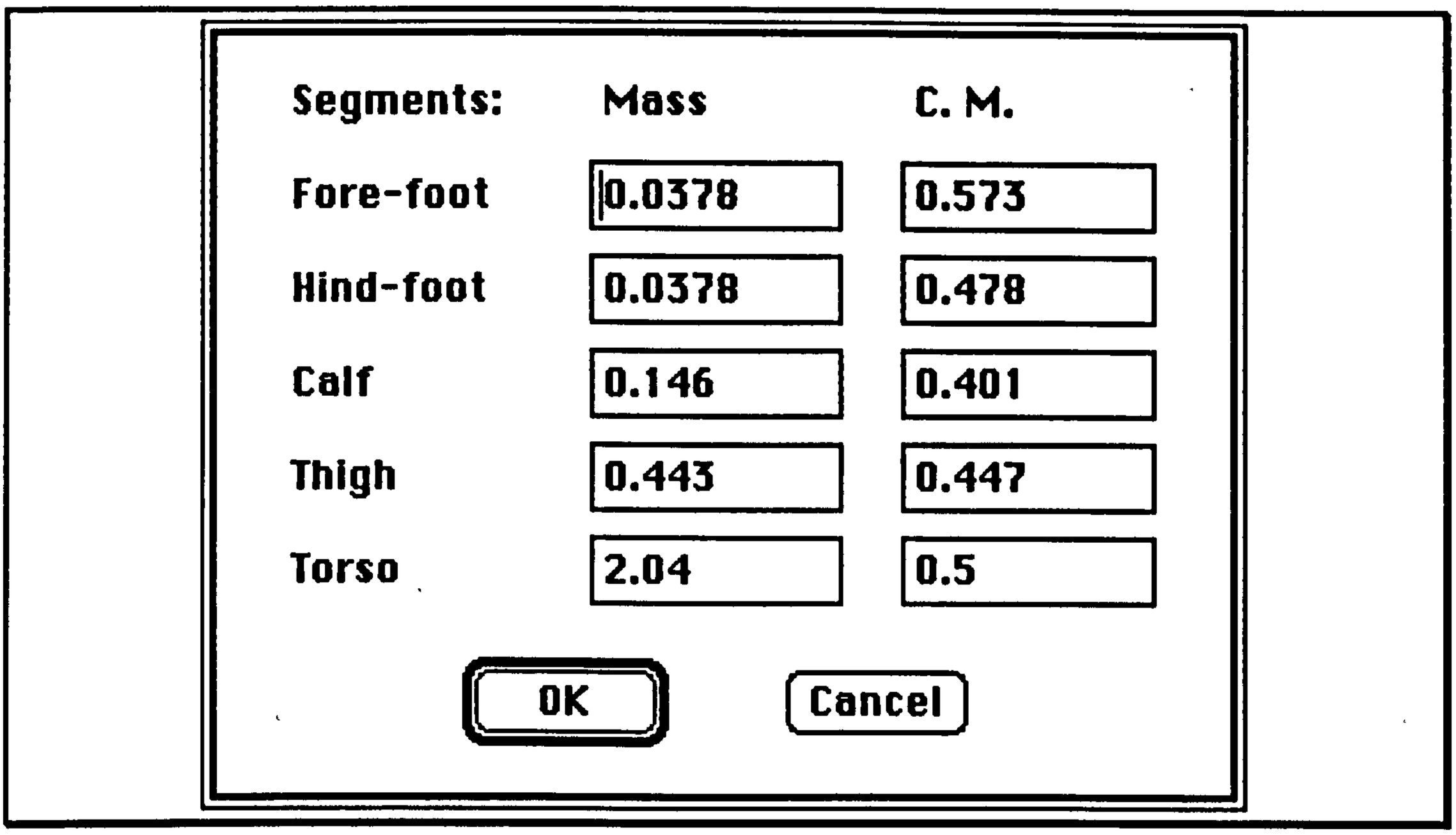


Mass is the overall mass of the animal in kg. **g** is the acceleration due to gravity in ms^{-2}. **Time tolerance** is a measure of the precision used to decide whether the iteration has finished. It is actually a fraction of the time interval but there is little reason to alter it from its default value given here. **Range** is the horizontal distance of the leap in metres. **Number of times** is the number of times required in the output data. **Maximum iterations** is the maximum number of iterations that the program will use to attempt to achieve the desired time tolerance. A

warning is issued if the number of iterations is exceeded, and the output data contains the best approximation currently available. **Extension fraction** is the fraction of the maximum extension distance of the hind-limb to be reached by takeoff. Sensible defaults are provided for all the values here except **Mass** which needs to be set for each animal.

It is not necessary, or even desirable, to set the tolerance to a much lower value than the default, or to set the number of times to calculate to a value greater than 100. The current tolerance is close to the floating point accuracy of the computer, and is easily accurate enough. The program is currently set with a maximum number of times of 100. This could easily be extended by recompiling, but, experience has shown that the errors from second order differentiation simply due to rounding error become very noticeable if the sample frequency is set much higher than this value. The default maximum number of iterations value of 100 has been sufficient for all cases tried so far unless there has been some error in the input data.

Segments is for the mass distribution properties of the animal:



Mass is the mass of the segment. These values can either be fractional masses, or actual masses in any units. The actual mass of each segment is calculated from the total mass of the animal. This allows a generic set of segment masses to be used here with only the total mass changing. **C. M.** is the fractional position of the centre of mass moving proximally and cranially.

The output from **Leaping Model** is a text file with a **.node** suffix in the correct format to be read into **gap** for further analysis. This file is a list of the positions of the nodes of the model for the number of times requested. Since each position is calculated by an iterative method rather than analytically, it is fairly slow, and would benefit greatly from being run on a Macintosh with a maths co-processor chip installed.

gap requires an appropriate **.limb** file to understand the data produced by **Leaping Model.** Here is a suitable example:

```
'Microcebus murinus limb file (Bitters)'
6
0     'Toe tip'
1     'Mid-tarsal joint'
2     'Ankle'
3     'Knee'
4     'Hip'
5     'Nose tip'
5
0     'Forefoot'     0     1     8.75E-04     5.73E-01     1.26E-08
1     'Hind-foot'    1     2     8.75E-04     4.78E-01     1.32E-08
2     'Calf'         2     3     3.38E-03     4.01E-01     3.57E-07
3     'Thigh'        3     4     1.03E-02     4.47E-01     1.26E-06
4     'Torso'        4     5     4.71E-02     5.00E-01     1.82E-05
```

Only the mass properties and the title line need to be changed for each animal to match those used in **Leaping Model.** The rest of the values describe how the segments in the model are joined together, and this cannot be altered.

Technical Description

This section describes the programming techniques used for each of the computer programs, and describes how they might be customized and extended. None of this information is necessary to use the software. In addition, a full listing of the source code is given for each program.

gap

gap has been written in C and currently runs on Hewlett-Packard 9000 series 300 hardware under HP-UX, a System V version of Unix. It makes extensive use of the HP X11 tool-kit and widget library and the HP 3D graphics library Starbase so that currently it could not be simply re-compiled to run on a different machine. It also uses the NAG numerical analysis libraries for some of the calculations and for plotting the results graphically. However, parts of the program have already been ported to run on Macintosh hardware, and it is expected that the whole program will eventually run on any hardware that supports X11.

The HP9000-350SRX that is used to develop the program has some features not commonly found on computing systems: a 3D hardware graphics accelerator which allows real-time animation of solid rendered images; knob and button boxes in addition to the normal mouse for extended analogue control. These features are all supported by the code.

Features

User Interface

The program has been designed from the start to be as easy to use as possible. It is entirely menu driven, and primarily mouse controlled. The main compromise here is between ease of use and generality. The more

things are tied to menu choices, the less absolute flexibility is available. In an attempt to overcome some of these problems, many of the menus are constructed dynamically from information in the model description.

The main flaw is that the program has not adopted the *event driven* format that is now gaining in popularity. This is where the actions of the program are governed as much as possible by events generated by interaction with the program's user. The main advantage is the avoidance of *modes* where the user is required to perform operations sequentially: first do A; then do B; and so on... This may not seem to be a problem to hardened computer users, and is indeed the classical programming model used, but it is not, unfortunately, the way that people choose to operate. In general, A and B should be performed when the user decides/remembers that it should be done, and the software can sort out the required ordering internally. As can be imagined, there is an appreciable amount of extra work involved for the programmer.

In addition, more use could have been made of the X11 tool-kit to provide dialog and message boxes instead of using the terminal emulator window. Again this an expedient demanded by lack of time. X11, unlike other graphical interfaces such as the Macintosh does not provide interactive design tools for its widgets and panels so that providing this level of interaction requires a very much larger amount of work.

Generality

Generality was a major functional aim of the program. Whilst the primary goal for this project was to look at leaping in prosimian primates, it was also envisaged that the same software could be extremely useful in looking at the mechanics of a tennis serve, or a golf swing. Thus the number of measurement points, their names and their connection

pattern are completely defined by the user. The only limitation is that each segment is only defined by two points and these need to be the joints so that, for instance, there is no facility for picking a point on an animal that is easy to measure and can be considered to have some fixed relationship to a particular joint that is, itself, very difficult to define. However, since the program produces a data file containing raw position information, the different parts of the program: data acquisition and data analysis can be used in isolation.

The source code is written for clarity rather than, necessarily, efficiency where there was any conflict between the two. In particular, considerable effort was made to keep the machine specific parts of the program isolated so that it could be moved onto other platforms without too much difficulty. In addition, there is specific support for user supplied extensions in the analysis subroutine since it is quite impossible to provide all possible options that could be requested. In addition, the inverse dynamic module is specific for a particular limb segment configuration and although it is relatively easy to change the given source code for a different model, it is extremely difficult to provide general code for this.

Scope

The program allows the user to call up and measure a series of stored images. It has three calibration models: 2D with correction for rotation and scale; 3D from 2 orthogonally mounted cameras correcting for rotation and scale; 3D from 2 randomly placed cameras calibrated from at least 6 known points. It has interactive image enhancement of contrast and brightness, and can use pseudo-colours to further improve detail separation. Current images are 512 by 256 (stretched to 1024 by 768) by

256 grey levels, but this can easily be expanded depending on the available video display and capture equipment. All measurement is performed by mouse movement which is extremely time efficient and allows each measured position to be prompted for on-screen and to be marked on the image.

Calibration data can be stored for subsequent re-use. Measurement data is stored unsmoothed, but is optionally smoothed by a digital filter or by a user selectable moving average when it is re-read.

The program is able to present animated sequences depending on the linkage map set up in the original model description file. When in 3D mode, this is a fully Phong rendered stick figure with smoothed octagonal prisms representing the limb segments (Hewlett-Packard 1988d). The colour and surface properties of the display can be changed and the user is able to move the viewpoint around the model, zoom and pan in real-time using the knob box. In addition the animation can be single stepped, or allowed to run through only a user defined part of the whole sequence. The 3D effect is extremely convincing and can be an important visualization tool.

Full kinematic analysis facilities in 2 or 3D are available within the program with the results output to the screen, a plotter or to data file in 123, Excel or SAS formats. The program will calculate positions, velocities and accelerations of all measured points and the centres of mass of each segment and the overall model. It will calculate the angles, angular velocities and angular accelerations of each segment. Using the mass and moment of inertia data, it can then calculate the net force and torques acting on each segment, and also its potential and kinetic energy - both linear and rotational.

In the user specified analysis section it is currently set up to perform inverse dynamic calculations on the measured 2D model (8 segments) and the predictive model (5 segments). These both calculate the reaction forces, the torques and the work per frame at each joint, and the torque induced bending moment on each segment.

Structure

Initialization

The basic structure of the program can be considered to be a tree rooted at the main menu level. Outside this are just the routines to initialize the graphics, windows and user interaction hardware at the beginning and the routines to disconnect them all at the end of the program. The function **open_dev** performs initialization and **close_dev** tidies up when the program has finished.

The window arrangement is set so that text interaction occurs in the terminal window from which the program is called, and a so called *seethru* window is set up to allow the graphics routines to run in the image planes of the display. The windowing is all done in the overlay planes.[48] This allows the user to move and resize the graphics window whilst still allowing use of the high-speed hardware 3D rendering under Starbase.

[48]The picture displayed on the video screen is obtained from a large block of video memory in the graphic accelerator. For each pixel, there are 24 bits (planes) of image information to allow 16 million colours and 4 bits of overlay information to allow 15 colours plus transparent so that the underlying image can be masked.

Data Input

Video Display

Image measurement is done from digitally stored images. This can be from any source, though the format needs to be appropriate for this program. The current size limits are 1024 by 768 and only 256 grey levels are supported. This limitation is because the image is treated simply as a block of data that is written over to the display hardware in an untranslated form. This proved to be necessary to get the images displayed from disc at a usably fast speed. On the current hardware it takes about 2 seconds.

Once displayed, the user can modify the contrast and brightness using the knob box. This is done by switching the display mode to use a lookup table instead of the true colours used for 3D. The display look-up table can be re-mapped very quickly[49] to change either the effective brightness range or the offset of the whole screen. In addition, there are facilities to allow the display of the image in pseudo-colours[50], again by re-mapping the look-up table.

On-Screen Measurement

Data measurement is performed in the function **digrd** by entering a loop where the right mouse button status is polled. X11, by default, only uses

---

[49]Each pixel of the image is stored as a number from 0 to 255. This number is used as the index to a look-up table of red, green and blue values so that, although only 256 colours can be displayed simultaneously, these can be chosen from a palette of approximately 16 million. To rapidly change the appearance of colours on the display, the entries in this table can be altered and it will immediately effect the whole of the display.

[50]Pseudo-colours are where artificial colours are applied to an image instead of grey levels. By using colours, the effective contrast is enhanced and certain areas of the image can be made to stand out very clearly.

the left mouse button, so by using the right button, the normal X11 window switching functions can still be supported. Within this loop, the knob positions are also monitored to allow real-time contrast adjustment whilst measuring is being undertaken.

When a right button press is detected, the loop is ended and a marker drawn on the screen to provide feedback to the user that a particular location has now been measured. Use has shown that this is particularly reassuring, and eliminates measurement errors associated with measuring the same point twice. The marker, and all subsequent calculation is done in the Starbase coordinate system rather than the X11 coordinate system. Starbase uses floating point coordinates with the origin in the lower left, whereas X11 uses integers with the origin at the upper left. There is also some discrepancy between the perceived position of the cursor and the centre of the marker and this few pixel difference also has to be compensated for.

2D/3D Reconstruction

The reconstruction is handled by having two separate menu selections: one for starting a new series of measurements; the other for continuing with an existing set of measurements. The effects of both are controlled by the global options specifying 2D or 3D and, for 3D, orthogonal or DLT reconstruction. The initial calibration is performed through the **initrd** function, and subsequent measurements via the **read2d** and **read3d** functions.

When starting a new series of measurements, the option is given to load up an existing calibration file or to produce a new calibration, which can then be saved as required. Calibration is achieved by finding a geometrical transformation that maps real world coordinates onto screen coordinates.

To find real world coordinates from screen coordinates is then simply a matter of reversing this transformation. With 2D, this is a relatively trivial exercise since both the display device and the real world are considered as parallel planes, so that only uniform scale and translation are required.

For 3D measurements, the additional dimension means that information is required from another frame of reference: that is another camera. The complexity then depends on the restrictions placed on the camera positions. By having cameras placed orthogonally, one can be considered to be measuring the X and Y coordinates exclusively, and the other the Y and Z. These can then be treated as two independent 2D problems for X and Z and a mean can be used for the Y value. The only *caveat* here is that the cameras should be sufficiently far from the subject that this distance should swamp the variation in the depth of the subject so that parallax can be ignored. This, in itself, can be difficult to arrange.

Completely flexible 3D reconstruction, allowing for free choice of camera positions, and yet calibrating from known positions in the subject volume, requires a more thorough understanding of the optic train. Effectively, each position measured on the image can be considered as defining a straight line in space from the point defined on the film plane of the camera, through the optical centre of the cameras lens, to the actual position of the point being measured. If this is done for two cameras, then two lines are defined in space and the position of the target point is where these two lines cross. Unfortunately, due to errors in measurement, these two lines will almost certainly never actually touch, so some sort of nearest estimate approximation must be obtained. In addition, the optics of currently available lenses are such that the optical centre will change depending on the position on the film plane in some uncertain fashion (unless the camera has been specially calibrated as in

the case of specialist mapping stereo-photogrammetric systems), so that an approximation method will almost certainly give better results than trying to solve the problem analytically. This is where DLT reconstruction is useful: (Shapiro 1978, Miller et al. 1980)

DLT Reconstruction

The basic DLT equations derive from the standard photogrammetric equation, and are an approximation that lends itself easily to calculation and are relatively stable.

(1) $$q = \frac{L_1X + L_2Y + L_3Z + L_4}{L_9X + L_{10}Y + L_{11}Z + 1}$$

(2) $$r = \frac{L_5X + L_6Y + L_7Z + L_8}{L_9X + L_{10}Y + L_{11}Z + 1}$$

Where:

p,q	Camera (and hence screen) x and y coordinates
L_1–L_{11}	DLT parameters
X,Y,Z	World coordinates

For reconstruction use, the parameters, L_1 to L_{11} need to be calculated for each camera, by measuring the screen coordinates, p and q, for at least six sets of known world coordinates X, Y and Z.

Rearranging as follows:

(3) $$-L_1X - L_2Y - L_3Z - L_4 + L_9Xq + L_{10}Yq + L_{11}Zq = -q$$

(4) $$-L_5X - L_6Y - L_7Z - L_8 + L_9Xr + L_{10}Yr + L_{11}Zr = -r$$

For $q_{(1-n)}$, $r_{(1-n)}$, $X_{(1-n)}$, $Y_{(1-n)}$, $Z_{(1-n)}$, the solution matrix becomes:

(5)

$$\begin{pmatrix} -X_1 & -Y_1 & -Z_1 & -1 & 0 & 0 & 0 & 0 & q_1X_1 & q_1Y_1 & q_1Z_1 \\ 0 & 0 & 0 & 0 & -X_1 & -Y_1 & -Z_1 & -1 & r_1X_1 & r_1Y_1 & r_1Z_1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ -X_n & -Y_n & -Z_n & -1 & 0 & 0 & 0 & 0 & q_nX_n & q_nY_n & q_nZ_n \\ 0 & 0 & 0 & 0 & -X_n & -Y_n & -Z_n & -1 & r_nX_n & r_nY_n & r_nZ_n \end{pmatrix} \begin{pmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \\ L_5 \\ L_6 \\ L_7 \\ L_8 \\ L_9 \\ L_{10} \\ L_{11} \end{pmatrix} = \begin{pmatrix} -q_1 \\ -r_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ -q_n \\ -r_n \end{pmatrix}$$

Which can be solved for L_1 to L_{11} by standard over-defined linear equation techniques.

Once all the DLT parameters are known for each camera, then the unknown world coordinates of points can be calculated by measuring their screen coordinates in each camera where they are visible. They must be visible in at least two cameras.

Rearranging the standard equations:

(6) $$(qL_9 - L_1)X + (qL_{10} - L_2)Y + (qL_{11} - L_3)Z = L_4 - q$$

(7) $$(rL_9 - L_5)X + (rL_{10} - L_6)Y + (rL_{11} - L_7)Z = L_8 - r$$

For a series of cameras a, b, c..., for $q_{(a, b...)}$, $r_{(a, b...)}$ and $L_{(1\text{-}11)(a, b...)}$ the solution matrix becomes:

(8)

$$\begin{pmatrix} q_aL_{9a}-L_{1a} & q_aL_{10a}-L_{2a} & q_aL_{11a}-L_{3a} \\ r_aL_{9a}-L_{5a} & r_aL_{10a}-L_{6a} & r_aL_{11a}-L_{7a} \\ q_bL_{9b}-L_{1b} & q_bL_{10b}-L_{2b} & q_bL_{11b}-L_{3b} \\ r_bL_{9b}-L_{5b} & r_bL_{10b}-L_{6b} & r_bL_{11b}-L_{7b} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} L_{4a}-q_a \\ L_{8a}-r_a \\ L_{4b}-q_b \\ L_{8b}-r_b \\ \cdot \\ \cdot \end{pmatrix}$$

Which can be solved as before for X, Y and Z.

The NAG routine E02GCF is used to find the MINIMAX solution of the sets of linear equations. It is written in FORTRAN rather than C, so a certain amount of jiggery-pokery is required to get the two languages to talk to each other, and this is done via the small FORTRAN routines **dlt_parameters** and **dlt_recon.**

Initial tests in the laboratory showed this to be a perfectly acceptable implementation of the DLT algorithm, but subsequent field trials revealed it to be completely unuseable with my experimental setup. 3D reconstruction is much more sensitive to measurement error than the much simpler 2D approach. Visualize shining two narrow light beams from torches, trying to get the position where the two beams intersect, and to hold the beams still enough so that this intersection position does not move. The point measured in the two images must be the same point in space. In my experiments, the operator has to estimatie the position of the joint, and thus getting the same position estimate from two different views is extremely difficult. While this error is relatively small as far as 2D is concerned, it equates to a much larger error in 3D. Thus, whereas markers are optional for 2D, they are much more important if the DLT method of 3D measurement is to be attempted.

Kinematic Modelling

The kinematic modelling is achieved by treating the model segments as a set of vectors running from one defined joint to another as described in the limb model provided by the user. Each joint has an X, Y and Z position,and each segment has an angle in the X=0, Y=0 and Z=0 plane. For the 2D model, Z is always zero, so all rotation is in the Z=0 plane. The positions of the centres of mass of the segments are found by sub-dividing the segment vector by the centre of mass position fraction given

in the model description file. The reconstruction data gives a series of values of position or angle at a series of discrete times. Kinematic analysis requires these values to be converted into velocities and accelerations: a process that requires differentiation with respect to time. This program uses a simple straight line fitting approach with or without data smoothing. However, the differentiation routine is isolated so that other approaches can be easily implemented. Differentiation is available in the NAG libraries for example, and in is certainly relatively easy to fit a smooth curve to the data and differentiate this analytically. Various polynomial splines are ideal in this respect because they are guaranteed to be continuous up to the second derivative. However the experience of others indicate that the values obtained by these methods reflect more the fitting method than the actual data; this is certainly so when used to find second rather than first derivatives (Pezzack et al. 1977).

The smoothing is performed directly from the main menu loop whenever new data is obtained. The form is controlled by a global option and both the raw and smoothed data are stored internally. The choice of whether to use smoothed or raw data is then made from within the analysis and the animation sections.

Two digital smoothing algorithms are available in the program: 4th order low pass Butterworth digital filtration at a variety of cutoff frequencies and moving average smoothing over a variable number of steps and moving average smoothing over a user-defined number of steps. The advantage of the first method is that it has a real physical meaning: frequencies above a certain value are reduced by 12dB per octave, so that if a cyclic movement is being studied, where the characteristic frequencies are known by examining the frequency spectrum (obtained by Fourier analysis, for example), a cutoff point can be set above which

there are no frequencies of interest. Its main disadvantage is that it takes an appreciable number of samples at each end of the sequence for the filter to stabilize, so that a relatively large number of measurements are lost. In my experience, the filter appeared to behave properly after about 4 samples (in addition to the two lost due to the digital filter itself), which, again would not be a problem when looking at a cyclic behaviour, but because of the speed the animal left the field of view after takeoff, meant that this method was unsuitable for my experiments. Moving average smoothing has a less well defined physical description: there is more smoothing when the average is performed over a greater number of samples, and a greater number of samples is lost at each end. However, there is no settling time problem, so the number of samples lost is just the smoothing interval divided by two and rounded down. The minimum smoothing, over three values, only causes the loss of one data point at each end and provides a reasonable degree of smoothing. This was the technique that I habitually used.

The 4th order low pass Butterworth was implemented as described by Winter (Winter 1991). This is achieved by running the data through a second order filter twice, reversing the direction for the second run to correct for any phase distortion that has occurred.

The equation for the filtration function is as follows:

(1) $$F_n = a_0X_n + a_1X_{n-1} + a_2X_{n-2} + b_1F_{n-1} + b_2F_{n-2}$$

Where:

F_n	is the n^{th} filtered output coordinate
X_n	is the n^{th} raw input coordinate
a_n b_n	are the filtration coefficients

The filtration coefficients used are as follows: (Winter 1991)

f_s/f_c	a_0	a_1	a_2	b_1	b_2
4.0	0.2929	0.5858	0.2929	0.0000	-0.1716
5.0	0.2066	0.4132	0.2066	0.3695	-0.1959
6.0	0.15505	0.3101	0.15505	0.6202	-0.2404
7.0	0.1212	0.2424	0.1212	0.8030	-0.2878
8.5	0.0884	0.1768	0.0884	1.0011	-0.3547
10.0	0.06745	0.1349	0.06745	1.1430	-0.4128
12.0	0.0495	0.0990	0.0495	1.2796	-0.4776
14.0	0.0379	0.0758	0.0379	1.3789	-0.5305
16.0	0.02995	0.0599	0.02995	1.4542	-0.5740
18.0	0.0243	0.0486	0.0243	1.5134	-0.6106
20.0	0.0201	0.0402	0.0201	1.5610	-0.6414

Where:

f_c is the cutoff frequency

f_s is the sampling frequency

Filtration is performed in the routine **filter** so this can be altered independently if a different filtration algorithm is required.

The moving average smoothing is performed in the routine **smooth**. The formula used is as follows:

$$S_n = \frac{1}{2m+1} \sum_{i=-m}^{m} X_{n+i} \qquad (2)$$

And:

$$m = \frac{N-1}{2} \qquad (3)$$

Where:

S_n	is the n^{th} smoothed value
N	is the number of values smoothed over (must be odd)

A certain amount of experimentation is required to get the degree of smoothing right, but it is closely linked to the framing rate chosen, and if this is fixed (for example to 50Hz for a video camera) then any smoothing is quite likely to remove some of the high frequency detail.

Animation

The animation section of the program provides the facility to display the measured data as a movie sequence. This is visually very effective, and by removing the extraneous detail from the original recording, is often much more revealing. In the 3D mode, by allowing rotation of the model, the movement can be viewed from novel positions which can give a very clear picture of the important movements. Also, it immediately reveals if any joints have been mis-measured in any of the frames. Often, a joint is measured out of sequence, even with the on-screen prompting, but the animated display will quickly reveal, for instance, if a hip position has been entered as the tail-tip. It is very much harder to pick this out when just looking at graphs of joint positions.

The animation technique uses double buffering to produce completely smooth screen updating. The original 24 planes of image memory are divided up into two sets of 12 planes, only one of which is displayed. The next frame to be displayed is drawn up on the non-visible set of planes, a and when the drawing has been finished, the visible plane sets are switched. This switching is very much quicker than the drawing (even with the accelerated video hardware) and can be synchronized to the

vertical fly-back period of the cathode ray tube[51], and hence never causes the flickering associated with screen update while the electron beam of the CRT is involved in displaying an image.

The routine that draws up the 3D figure on the screen needs some explanation. Each segment is drawn as an octagonal prism extending from the proximal to the distal joint positions. To take advantage of the accelerator hardware, as much calculation as possible needs to be done in the accelerator, and as little as possible locally. On the SRX systems, the 3D calculations rely on a matrix transformation stack. Any transformations that are to use the accelerator are *pushed* onto the stack (including the 3D to 2D conversion) and all subsequent drawing operation will use the combined stack transformation. When no longer required, transformations can be *popped* off in the reverse of the order in which they were applied. So, when the octagonal prism drawing routine, **draw_limb** is called, it actually sends the commands to draw an appropriately sized prism running from the origin along the X axis, a simple scaling exercise, and then calculates the transformation that would be required to put this prism in the correct place as a combination of a Y and Z axis rotation and a translation and puts these transformation matrices onto the accelerator stack. The graphics hardware then draws the limb in the correct place. The prism is built up from a series of planar polygons drawn in an anti-clockwise fashion when viewed from outside. Phong shading, selected in **displa**, makes the octagonal prism

---

[51]The vertical flyback period is when the beam of the CRT is being from the bottom of one frame back to the top of the screen ready to display the next frame. If all the display updating can be done during this period when nothing is being draw to the VDU screen, then the update will appear completely seamless, and the animation will appear very much smoother.

appear as a smooth cylinder. Colours and reflectance properties, along with light positions, are also added to improve the illusion of depth.[52]

If desired, there are options to show the model as a series of lines, to show how it is actually built up. Hidden line removal is performed by using a Z buffer. Unfortunately, this is only 16 bits deep and there are problems with breakthrough of what should be hidden faces at the joints because of the relatively poor Z resolution[53]. For 2D, the model is displayed as a series of linked rectangles.

Graphical Data Display

The graphical data display is all passed to the routine **d_graph** as a series of linked X and Y coordinates and labels. It then calculates the optimal ranges internally and lets the actual plotting be handled by the NAG graphical routines working on top of Starbase. This again requires the interface between C and FORTRAN, and some ancillary conversion code is required. This is all contained in the file **general.f**. The main problem here is that in C, two dimensional arrays increase their second dimension fastest. In FORTRAN, the first dimension is increased first. This means that the order of dimension specifiers has to be reversed. In addition, character expressions in FORTRAN are not zero delimited as they are in C, so that anything dependent on the length of a string of characters is unlikely to work well.

---

[52]For more details, see the Starbase manual (Hewlett-Packard 1988d).

[53]Hidden line/face removal is a difficult problem. One approach is to store a depth associated with each pixel colour. The colour will only be overwritten by another colour if the depth value associated with the new colour is higher, meaning that this new colour is closer to the observer than the old one. The problem comes when two pixels are very close in depth. Because of the relatively low resolution of the Z buffer, the one that will get displayed may very well be dependent solely on rounding errors in the depth calculation and random breakthrough can occur.

For plots of the locus of a point, each individual position is numbered to allow more timing information to be displayed. This function is not provided in the libraries so has had to be coded explicitly using the NAG graphics primitives. A key is also drawn up, identifying the individual lines in multi-line graphs.

Data Export

The data export section of the program, **save_an**, produces files that can be read directly into some popular data manipulation and analysis packages. It is an option after a set of data has been plotted on the screen. The ones chosen are the text formats used by Microsoft Excel and Lotus 123. These are sometimes loosely known as *tab delimited format* and *comma delimited format*. In addition, GAP is capable of producing a program file for SAS that contains the data embedded in it, and has the required statements to produce a very similar graph using SAS graph. These programs allow for analysis that is not supported within GAP, and can, for instance, be used to produce graphs of data combinations that are not supported internally.

User Routines/Dynamic Modelling

The routine **user_specific_analysis** is simply a convenient point where all the data produced by the program is available. It is probably easier to add any extra analysis here rather than work from exported data because all the data is predefined and so no more data input routines need to be written.

The inverse dynamic sections of the program are accessed from here because, unlike the other data manipulations, they are not general, and require a specific connection arrangement of limbs. The two provided:

simplified_quadrupedal and **predictive_model** are specific to the 2D measurements I used and my predictive model respectively. They are both straightforward examples of implementations of the inverse dynamic equations as described in the inverse dynamics chapter. The only difficulty is keeping track of the relatively large number of individual values associated with each segment, and getting the signs right across joints: when more than one segment meets.

The **predictive_model** routine is the more straightforward simply because it has fewer segments and they are only connected in a simple linear fashion with no branching. The calculation proceeds from the cranial end of the torso, since there are no forces or torques acting there, and calculates the forces and torques that must be acting on the caudal end to produce the observed angular and linear acceleration given its mass and moment of inertia. These forces must also apply, in an equal and opposite fashion in accordance with Newton, on the proximal end of the thigh. Knowing these forces and torques, in a similar fashion, the forces and torques on the distal end of the thigh are calculated. These are equal and opposite to the ones on the proximal end of the calf and so the distal calf values can be found. This process is continued until the distal end of the fore-foot, giving values for all the joints in the model.

The only important difference in **simplified_quadrupedal** is that there are 3 free ends: the hand; the nose tip; the tip of the tail. These are all separately treated as previously with the caudal end of the torso, and have no forces acting on them. In addition, the neck and hip joints have three segments each meeting at them. The approach here is to make sure that the total force and torque from the ends of the three meeting segments is zero. Only one segment end in each case is unknown.

File Formats

.limb File

This file has been designed so that it is easily produced or read by a FORTRAN program where file manipulation in text mode is somewhat limited, but normally, it is produced using a text editor such as **vi** or **emacs**, or alternatively, by any word processor that can save files as unformatted text. The line spacing is not at all important since all the character strings are delimited by single quotes, but the suggested line spacing makes the file relatively easy to read.

Here is an example **.limb** file:

```
'Microcebus murinus limb file (Bitters)'
10
0    'Forelimb tip'
1    'Elbow'
2    'Shoulder'
3    'Toe tip'
4    'Mid-tarsal joint'
5    'Ankle'
6    'Knee'
7    'Hip'
8    'Nose tip'
9    'Tip of tail'
9
0    'Lower arm'    0    1    3.13E-03    5.00E-01    2.35E-07
1    'Upper arm'    1    2    2.88E-03    5.18E-01    1.77E-07
2    'Forefoot'     3    4    8.75E-04    5.73E-01    1.26E-08
3    'Hindfoot'     4    5    8.75E-04    4.78E-01    1.32E-08
4    'Calf'         5    6    3.38E-03    4.01E-01    3.57E-07
5    'Thigh'        6    7    1.03E-02    4.47E-01    1.26E-06
6    'Head'         8    2    6.38E-03    5.00E-01    7.30E-07
7    'Torso'        2    7    3.26E-02    5.00E-01    1.82E-05
8    'Tail'         7    9    2.19E-03    3.78E-01    2.33E-06
```

The first line contains a title that is used to provide more information about the contents of the file. It is delimited by single quotes[54]. The second line is an integer (10 here) that specifies the number of nodes in the model. The nodes are named on the next 10 lines: each of these lines starts with an incremental integral identification number running from 0 to 9, and is followed by the actual name of the node. The names are again delimited by single quotes. The next integer, in this case 9, is the number of segments defined. The 9 subsequent lines contain, first of all, an incremental identification number. This is followed by the name of the segment in single quotes, then the identification numbers of the two joints at the ends of this segment. For the purposes of the program, this segment is considered to extend from the first numbered joint to the second numbered joint. So, in this case, the lower arm, runs from the forelimb-tip (joint 0) to the elbow joint (joint 1). The next number on the line is the mass of this segment in kg, and the next is the position of the centre of mass as a fraction of the distance from the first joint to the second joint. In other words, a value of 0 would indicate that the centre of mass was at the first joint, and a value of 1 would indicate that it was at the second joint. Normally, this value is not too far from 0.5. The last number is the moment of inertia of the segment in $kg.m^2$.

All the values are required though dummy values can be inserted for the mass properties if they are not required. Zero should not be used as a dummy value since it will lead to divide by zero errors in the program, and 0.5 should be used for dummy centre of mass positions. The character strings have no significance within the program except as the

---

[54]The single quote is the ASCII value 39. This should not be confused with the opening and closing single quotes provided on the Macintosh system whose ASCII values are 212 and 213. These are not interchangeable, and do, in fact, look quite different: ' as opposed to ‘ or’.

names used as prompts to the user and as labels on the output data. Extra nodes and segments are added by changing the integers indicating the total number of nodes or segments and by adding extra node or segment description lines. As mentioned before, a single node can be used to specify more than one segment, or, indeed, no segments at all, and act simply as a position marker.

Only the first half of the file which defines the joints is used for the measurement and reconstruction phase, so if there is an error in the segment description, this can be changed after measurement as long as the joint definitions themselves are not altered. The other area to be careful in is in defining the position of the centre of mass. It is very easy to get the direction sense of this fraction wrong. It is the fraction of the distance from the first defined node in a segment to the second. The integers should be typed in as simple numbers (i.e. 1 rather than 1.0) but the floating point numbers can use any standard computer notation (1 1.0 or 1.0e0).

This is the code segment used to read this data (NPS is defined as 2):

```
/* read the data */

fscanf(unit,"'%[^']'",title);
fscanf(unit,"%d",nnodes);
for (inodes=0;inodes<*nnodes;inodes++)
{
    fscanf(unit,"%d '%[^']'",&inode,nodes[inodes]);
    if (inode!=inodes)
    {
        printf("Node number mismatch during file read error\n");
        return;
    }
}

fscanf(unit,"%d",nsegs);
for (isegs=0;isegs<*nsegs;isegs++)
{
    fscanf(unit,"%d '%[^']'",&iseg,segs[isegs]);
    if (iseg!=isegs)
    {
        printf("Segment number mismatch during file read error\n");
        return;
    }
    for (i=0;i<NPS;i++)
    {
        fscanf(unit,"%d",&nodes_per_seg[i][isegs]);
    }
    fscanf(unit,"%f",&seg_mass[isegs]);
    fscanf(unit,"%f",&seg_com[isegs]);
    fscanf(unit,"%f",&seg_moi[isegs]);
}
```

.node File

This is the file containing the measurements produced by GAP. It is also produced as the output from **leaping model**. It is not expected to be produced by hand, but it is in an ASCII form to allow portability across different computer platforms, and to allow other computer programs to produce suitable raw data for input into the analysis sections of GAP.

Since these files are generally quite large, and are not designed to be easily read by people, I will instead give the code segments both for writing and reading the file.

Writing a .node file:

```
/* write data */

fprintf(unit,"%s\n",title);
fprintf(unit,"%e\n",fspeed);
fprintf(unit,"%d\n",nframe);
for (iframes=0;iframes<nframe;iframes++)
{
    fprintf(unit,"%d\n",iframes);
    fprintf(unit,"%d\n",nnodes);
    for (inodes=0;inodes<nnodes;inodes++)
    {
        fprintf(unit,"%d %e %e %e\n",inodes,xpos[inodes][iframes],
            ypos[inodes][iframes],zpos[inodes][iframes]);
        fprintf(unit,"%s\n",nodes[inodes]);
    }
}
```

Reading a .node file:

```
/* read data */

fscanf(unit,"%[^\n]\n",title);
fscanf(unit,"%f\n",fspeed);
fscanf(unit,"%d\n",nframe);
for (iframes=0;iframes<*nframe;iframes++)
{
    fscanf(unit,"%d\n",&iframe);
    if (iframe!=iframes)
    {
        printf("Frame number mismatch in node data file\n");
        return;
    }
    fscanf(unit,"%d\n",nnodes);
    for (inodes=0;inodes<*nnodes;inodes++)
    {
        fscanf(unit,"%d%f%f%f\n",&inode,&xpos[inodes][iframes],
            &ypos[inodes][iframes],&zpos[inodes][iframes]);
        fscanf(unit,"%[^\n]\n",nodes[inodes]);
        if (inode!=inodes)
        {
            printf("Node number mismatch in node data file\n");
            return;
        }
        if (flag_2d) zpos[inodes][iframes]=0.0;
    }
}
```

Where:

fspeed	is the interval between frames in seconds
nframe	is the number of frames
nnodes	is the number of nodes

xpos is the X position of the nodes for each frame in metres

ypos is the Y position

zpos is the Z position

.2d .3d1 .3d2 Files

These are the calibration files produced by the various different reconstruction models used in the program. They are almost certainly of no use whatsoever to any other program but their details will be given for the sake of completeness.

Program segment to read **.2d** file:

```
fscanf(unit,"%e%e",&fiducial_x,&fiducial_y);

fscanf(unit,"%e",&x_offset);
fscanf(unit,"%e",&y_offset);
fscanf(unit,"%e",&scale_factor);
fscanf(unit,"%e%e",&rotation[0][0],&rotation[0][1]);
fscanf(unit,"%e%e",&rotation[1][0],&rotation[1][1]);
```

Program segment to read **.3d1** file:

```
for (i=0;i<11;i++) fscanf(unit,"%e",&l1[i]);
for (i=0;i<11;i++) fscanf(unit,"%e",&l2[i]);

fscanf(unit,"%e%e",&fiducial_x,&fiducial_y);
```

Program segment to read **.3d2** file:

```
fscanf(unit,"%e%e",&fiducial_x,&fiducial_y);

fscanf(unit,"%e",&x_offset_1);
fscanf(unit,"%e",&y_offset_1);
fscanf(unit,"%e",&scale_factor_1);
fscanf(unit,"%e%e",&rotation_1[0][0],&rotation_1[0][1]);
fscanf(unit,"%e%e",&rotation_1[1][0],&rotation_1[1][1]);
fscanf(unit,"%d",&x_mirror_1);

fscanf(unit,"%e",&x_offset_2);
fscanf(unit,"%e",&y_offset_2);
fscanf(unit,"%e",&scale_factor_2);
fscanf(unit,"%e%e",&rotation_2[0][0],&rotation_2[0][1]);
fscanf(unit,"%e%e",&rotation_2[1][0],&rotation_2[1][1]);
fscanf(unit,"%d",&x_mirror_2);

fscanf(unit,"%d",&x_axis_source);
fscanf(unit,"%d",&y_axis_source);
fscanf(unit,"%d",&z_axis_source);
```

.00n File

This is the format of the image file used. If the file extension is a 3 digit number padded with zeros (.000, .001, .002 etc.) then the auto-increment feature of the program can be used which allows image files to be read in sequentially by supplying the name of the file without the extension, and the number of the first file to be read in.

Currently two formats are supported which are switchable at compile time. One is used by the Visilog image analysis package, and the other is specific to this program. There is also the option of allowing the image to be re-scaled as it is read in. This is not recommended unless a faster computer is used: I preferred to re-scale all the images that I was going to use in a session beforehand using **stretchpic** so that individual images were loaded into the computer as quickly as possible. Only 256 grey level images are supported.

Here is the code segment for reading in a Visilog format image:

```
struct
{
    long int magicNumber;
    long int pixelsPerLine;
    long int numberOfLines;
    long int res1;
    long int res2;
    long int res3;
    long int gridType;
    long int res4;
    long int arithmaticType;
    long int bitsPerPixel;
    long int res5;
    long int xOrigin;
    long int yOrigin;
    long int res6;
    long int res7;
    long int visilogHeaderSize;
    long int userHeaderSize;
    long int res8;
    long int totalHeaderSize;
} imageHeader;

fread(&imageHeader,sizeof(imageHeader),1,unit);
xrange=imageHeader.pixelsPerLine;
yrange=imageHeader.numberOfLines;
nbytes=fread(buffer,xrange*yrange,1,unit);
```

And for the other option:

```
xrange=getc(unit);
xrange=xrange+256*getc(unit);
yrange=getc(unit);
yrange=yrange+256*getc(unit);
nbytes=fread(buffer,xrange*yrange,1,unit);
```

.txt .prn .sas Files

These files are written out by the function **save_an** and are designed to be read in by other programs. **.txt** is for Microsoft Excel, **.prn** is for Lotus 123 and **.sas** is for SAS. Sensible column labels are defined for each program from the names of the lines on the corresponding plot and the units used.

Code for Excel (tab delimited):

```
/* write out data in ASCII form suitable for EXCEL import */

fprintf(unit,"%s\r",title);
for (iline=0;iline<nline;iline++)
    fprintf(unit,"%s\011\011",key[iline]);
fprintf(unit,"\r");
for (iline=0;iline<nline;iline++)
    fprintf(unit,"%s\011%s\011",x_label,y_label);
fprintf(unit,"\r");
for (ipoint=0;ipoint<npoint;ipoint++)
{
    for (iline=0;iline<nline;iline++)
        fprintf(unit,"%12.5e\011%12.5e\011",
            x_point[iline][ipoint],
            y_point[iline][ipoint]);
    fprintf(unit,"\r");
}
```

Code for 123 (comma delimited):

```
/* write out data in ASCII form suitable for LOTUS 123 import */

fprintf(unit,"\"%s\"\n",title);
for (iline=0;iline<nline;iline++)
    fprintf(unit,"\"%s\" \" \" ",key[iline]);
fprintf(unit,"\n");
for (iline=0;iline<nline;iline++)
    fprintf(unit,"\"%s\" \"%s\" ",x_label,y_label);
fprintf(unit,"\n");
for (ipoint=0;ipoint<npoint;ipoint++)
{
    for (iline=0;iline<nline;iline++)
        fprintf(unit,"%12.5e %12.5e ",x_point[iline][ipoint],
            y_point[iline][ipoint]);
    fprintf(unit,"\n");
}
break;
```

Code for SAS:

```
/* write out SAS program in ASCII file */

fprintf(unit,"/* %s */\n\n",title);
for (iline=0;iline<nline;iline++)
    fprintf(unit,"/* col%02d - %s */\n",iline,key[iline]);
fprintf(unit,"\n");
fprintf(unit,"/* x label - %s */\n/* y label - %s */\n\n",
    x_label,y_label);
fprintf(unit,"data gait;\ninput\n");
for (iline=0;iline<nline;iline++)
    fprintf(unit,"col%02d_x@@ col%02d_y@@\n",iline,iline);
fprintf(unit,";\ncards;\n");
for (ipoint=0;ipoint<npoint;ipoint++)
{
    for (iline=0;iline<nline;iline++)
        fprintf(unit,"%10.3e %10.3e\n",x_point[iline][ipoint],
            y_point[iline][ipoint]);
}
fprintf(unit,";\nrun;\n");

fprintf(unit,"proc gplot;\n");
fprintf
    (unit,"axis1 label=(f=swiss j=c '%s')\nvalue=(f=simplex);\n",
        x_label);
fprintf
    (unit,"axis2 label=(f=swiss j=c '%s')\nvalue=(f=simplex);\n",
        y_label);
fprintf(unit,"plot\n");
for (iline=0;iline<nline;iline++)
    fprintf(unit,"col%02d_y * col%02d_x\n",iline,iline);
fprintf(unit,"/overlay haxis=axis1 vaxis=axis2;\n");
fprintf(unit,"title f=centb '%s';\n",title);
for (iline=0;iline<nline;iline++)
{
    fprintf(unit,"symbol%-3d f=simplex i=join v='%d';\n",
        iline+1,iline);
    fprintf(unit,"footnote%-3d f=simplex j=l '%3d = %s';\n",
        iline+1,iline,key[iline]);
}
fprintf(unit,"run;\n");
```

digit.exe

Images were grabbed from moving video tape using a Matrox PIP1024B digitizing card. The software supplied with the card is fairly basic, and it has no facilities for grabbing a sequential set of frames from the video. Indeed, there is no capability for separating out the two fields making up each frame. The card has sufficient memory to hold 4 full frames (8

fields), but tests showed that it could not switch between grabbing areas quickly enough to grab sequential frames. The best that could be achieved was to grab every other frame.

The design goal was a system that could automatically recognize the position on the video film to a sufficient degree of accuracy to guarantee to be able to grab specifically identified frames on multiple passes of the film through the video recorder. Commercial systems are available that can do this by using animation controllers and SMPTE time codes recorded onto the unseen portions of the video image or onto one of the sound tracks. However, such a system was not available, so a poor man's imitation was designed and built.

Software

To get reliable grabbing of numbered video frames requires some sort of machine readable frame numbering system. This involves three steps. Firstly, a set of numbers need to be written to the sound-track of the film. Then, these numbers need to be displayed whilst the film is being played back so the operator can select the code numbers of the frames he or she wishes to grab. Thirdly, the required frames need to be grabbed.

The communication between the computer and the sound-track of the video was achieved using a modified modem circuit attached to the serial port. The initial numbering sequence was achieved by counting field number changes directly off the video grabbing card. For the first step, the video tape containing the required images is fully re-wound, the computer program is set to write the sound-track, and then the video tape recorder is set to audio dubbing mode. This means that while the recorder plays the video, it simultaneously records a new sound-track. The computer program monitors the change in field number and uses

this to increment a frame count. This value is then written to the serial port every 8 frames where the modem circuit will convert it into a set of coded tones that it will send to the video recorder. In this way, the whole of the video tape is given a sound-track that uniquely marks any section of the tape.

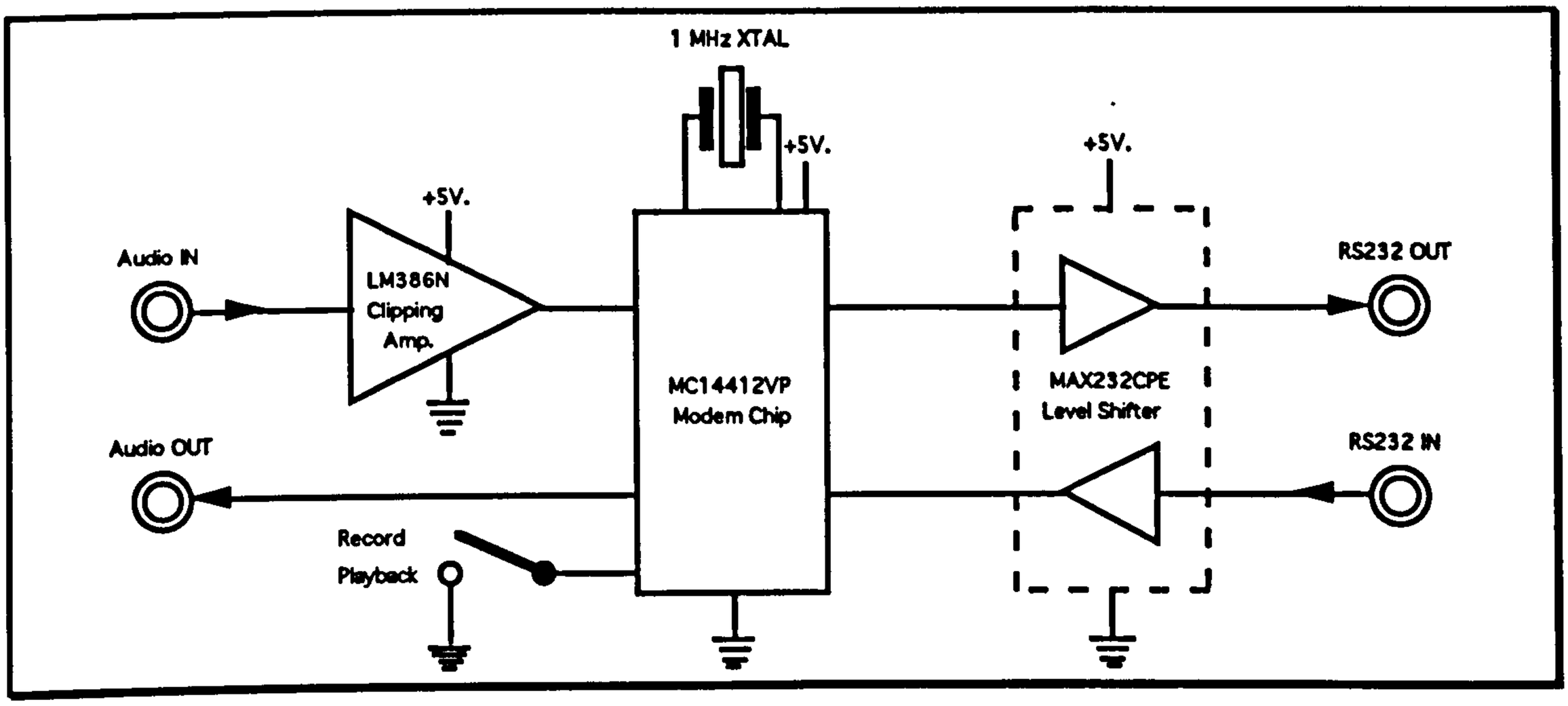
For display, the program is switched to its display mode, and the video allowed to play through the sequence of film that the operator is interested in. The modem decodes the sound-track, and sends the values to the serial port of the computer. These are then displayed on the screen, and can be frozen as required by the user.

For grabbing, the user types in the numbers of the first frame of interest and the number of sets of 16 fields that are required. The video is then re-wound to before the required part of the sequence, and switched to play. The computer again reads off the numbers coded onto the sound-track, but this time, when the required value is read, it grabs 8 of the next 16 fields (every other frame for the next 4 frames) and saves them to disk with their appropriate field numbers. The user then rewinds the tape and repeats the operation. This time, the computer delays 1 frame before starting its grab so that the missing 8 fields are grabbed. This whole process is repeated for the number of times originally selected by the user. The grabbing precision is perfect, but if an animation controller had been available, then the play/rewind/play cycle could also have been automated.

Hardware

A standard modem could probably have been used to write the sound-track, but when this was tried, the signal levels were found to be quite unsuitable for recording onto the video. In addition, only a 300 baud

modem was available, and this was really too slow a data rate for the very limited amount of time available for writing the numerical data onto the tape. Buying a faster modem is one option, but this does not solve the signal level problem, so instead, I decided to build a suitable circuit. This is appreciably simpler than a full modem circuit because only one set of signals is present at any time: it is either sending or receiving; never both at the same time.[55] It is based around the MC14412VP modem chip which does almost all the hard work. It also needs a suitable amplifier to convert the input analogue signal into a clipped 5V digital signal (approximately), and the output voltage needs to be converted from 0 to 5V TTL levels to -12V to +12V RS232 levels.



Schematic diagram of the modem circuit used to record and read a timing soundtrack on the video recorder. The only additional circuitry is a stabilized +5V power supply, and some ancillary passive components required for the correct functioning of the integrated circuits. Full details are available in the appropriate manufacturers data sheets.

[55]If two sets of signals are present, then notch filters need to be used for the input and output sections so that only the frequency that they are interested in is presented to them.

File Type

The only file used is the image file which the program produces. This is a 512 by 256 image in Visilog format. It needs to be processed by **stretchpic** before it is read into **gap**. (**gap** will read this image directly, but in the default setup, it will not alter its size. 512 by 256 looks very small and distorted when viewed on a 1280 by 1024 monitor.)

This is the code segment that does the actual write:

```
void FrameSave(filename,quadrant,field)
char filename[];                                    /* file name */
int quadrant;                                       /* sector number */
int field;                                          /* field number */
{
    FILE *unit;                 /* file unit */
    int rows=256;                                   /* number of rows in picture */
    int columns=512;                                /* number of columns in picture */
    int irow;                 /* row counter */
    int ycount;                /* y coordinate counter */
    char buffer[512];                               /* row buffer */
    struct visilogImageHeader fileHeader;   /* visilog file header */

    printf("Saving %s ....\n",filename);

    unit=fopen(filename,"wb");          /* open file */

    /* write out file header */

    fileHeader.magicNumber=VisilogConvert(0x6931);
    fileHeader.pixelsPerLine=VisilogConvert(columns);
    fileHeader.numberOfLines=VisilogConvert(rows);
    fileHeader.res1=VisilogConvert(1);
    fileHeader.res2=VisilogConvert(0);
    fileHeader.res3=VisilogConvert(0);
    fileHeader.gridType=VisilogConvert(1);                  /* rectangular */
    fileHeader.res4=VisilogConvert(0);
    fileHeader.arithmaticType=VisilogConvert(0x14);         /* integer */
    fileHeader.bitsPerPixel=VisilogConvert(8);
    fileHeader.res5=VisilogConvert(0);
    fileHeader.xOrigin=VisilogConvert(1);
    fileHeader.yOrigin=VisilogConvert(1);
    fileHeader.res6=VisilogConvert(1);
    fileHeader.res7=VisilogConvert(0);
    fileHeader.visilogHeaderSize=VisilogConvert(76);
    fileHeader.userHeaderSize=VisilogConvert(0);
    fileHeader.res8=VisilogConvert(0);
    fileHeader.totalHeaderSize=VisilogConvert(76);

    fwrite(&fileHeader,sizeof(fileHeader),1,unit);

    /* set up to read from right part of screen */

    ycount=field;

    for (irow=0;irow<rows;irow++)
    {
        fg_rowr(ycount,quadrant,buffer);/* read row into memory */
        fwrite(buffer,512,1,unit);    /* and write it to disk */
        ycount++;              /* increment ycount twice */
        ycount++;
    }

    fclose(unit);                   /* close file */
}
```

The function **VisilogConvert** is to convert two byte integers which are written most significant byte first on Motorola 68000 based machines

such as the Hewlett-Packard, and least significant byte first on Intel 8086 architecture.

stretchpic

This program is extremely simple. It reads in the standard 512 by 256 image file produced by **digit**, and writes out a file 1024 by 768. This corrects the aspect ratio so that each pixel is now square, and provides an image of a suitable size for display on the 1280 by 1024 workstation monitor. The only slight complication is that the program does not overwrite the existing image file directly. The target image is written to a temporary file that is then copied onto the original file. This means that if the program is interrupted for any reason, then there is always a copy of the data somewhere on the disk which can be recovered if necessary. Otherwise, an interrupt could cause the loss of the image file.

The program runs from the command line and accepts a list of file names. This is to allow the program to be used in shell programs on a Unix machine, and to allow wildcard file specification in the normal fashion.

Predictive Leaping Model

Features

This program is designed to permit easy explorative modelling: answering 'what if' style questions. For this reason, ease of use is of prime importance. It runs on any Macintosh computer running System 6.7 or higher with at least 1 Mbyte of RAM and a hard disk. It makes full use of the graphical interface, using menus, dialog boxes, and supporting desk accessories and the Multifinder.

Modelling parameters are entered into fields in various dialog boxes, with sensible defaults being provided where appropriate. This data is stored in binary files which preserves full accuracy at the expense of easy portability. It is not envisaged that these files will be used elsewhere. The output is a **.node** file suitable for GAP, where all the further analysis can be done.

Structure

Support for standard Macintosh features requires a great deal of extra programming. Much of the code is for initialization and operation of the user interface. The program is fully event driven, with functions being accessed of a main event loop depending on mouse clicks or command key combinations. Within dialog boxes, the program is modal: only interrupt driven system queues are active. During the modelling calculations, a dialog box allows cancellation, though, due to the non preemptive nature of multi-tasking on the Macintosh, this has a sluggish response on slower machines.

The modelling calculations are performed by **Calculate** and **LengthFunction. LengthFunction** uses a successive approximation method to find the length that corresponds sufficiently closely to a required time interval. Here, it is modelled using a mathematical function, but it could equally well interpolate between experimentally derived data points for time and distance. This would almost certainly be a worthwhile addition to the program since it would overcome the problem of the arbitrary nature of the force function chosen in the model.

The model data is encapsulated in a set of structures: **Vector**, **Coordinate**, **ModelCoordinates**, **ModelCOMs**, **ModelMass** and **ModelVectors**. This makes the code appreciably more readable, though

not necessarily more compact. It would certainly make model alterations quicker. It also simplifies the routines used to read and write the modelling data file, since they can simply dump the binary image of the contents of the relevant data structures to a disk file without any conversion.

The following code segment is used to write the modelling data file (the definitions for the structures are in **Params.h**):

```
/* create and open file */

FSDelete(gDefinitionFile.fName,gDefinitionFile.vRefNum);
Create(gDefinitionFile.fName,gDefinitionFile.vRefNum,
    FILE_OWNER,FILE_TYPE);
FSOpen(gDefinitionFile.fName,gDefinitionFile.vRefNum,&refNum);

/* write out data */

numBytes=(long)sizeof(gUserModel);
FSWrite(refNum,&numBytes,(char *)&gUserModel);

numBytes=(long)sizeof(gSegmentMass);
FSWrite(refNum,&numBytes,(char *)&gSegmentMass);

numBytes=(long)sizeof(gCOMs);
FSWrite(refNum,&numBytes,(char *)&gCOMs);

numBytes=(long)sizeof(gMass);
FSWrite(refNum,&numBytes,(char *)&gMass);

numBytes=(long)sizeof(g);
FSWrite(refNum,&numBytes,(char *)&g);

numBytes=(long)sizeof(gTimeTolerance);
FSWrite(refNum,&numBytes,(char *)&gTimeTolerance);

numBytes=(long)sizeof(gRange);
FSWrite(refNum,&numBytes,(char *)&gRange);

numBytes=(long)sizeof(gNumberOfTimes);
FSWrite(refNum,&numBytes,(char *)&gNumberOfTimes);

numBytes=(long)sizeof(gMaxIterations);
FSWrite(refNum,&numBytes,(char *)&gMaxIterations);

numBytes=(long)sizeof(gExtensionFraction);
FSWrite(refNum,&numBytes,(char *)&gExtensionFraction);

/* close file */

FSClose(refNum);
```

The routines **FSDelete**, **Create**, **FSOpen**, **FSWrite** and **FSClose** are Macintosh toolbox routines for file handling. They need to be used instead of the very similar C library functions so that features such as icons can be supported.

Appendix - Source Code

gap

C Routines

params.h

```
/* This is the general include file for the gaitan set of programs */

/* include files */

#include <stdio.h>
#include <math.h>
#include <string.h>

#ifdef HP
#include <fcntl.h>
#include "/users/bill/include/colour.h"
#include <starbase.c.h>
#include <X11/Xlib.h>
#include <Xr11/Xrlib.h>
#endif

/* bological and physical constants */

#define DENSITY 1.0                    /* mean body density */
#define G -9.80665                    /* acceleration due to gravity */

/* constant definitions */

#define SHELL "/bin/ksh"              /* shell option */

#define NPS 2                          /* Number of nodes to define a segment */
#define STRING_SIZE 80                  /* Size of strings */
#define MAX_NODES 10                   /* Maximum number of nodes */
#define MAX_SEGS MAX_NODES              /* Maximum number of segments */
#define MAX_FRAMES 100                  /* Maximum number of frames */
#define MAX_LINES (MAX_NODES+MAX_SEGS)        /* Maximum number of lines per graph */
#define MAX_POINTS MAX_FRAMES          /* Maximum number of points per line */

#define MAX_REF 20                     /* max number of reference points */
#define MIN_REF 6                     /* minimum */
#define DLT_FILE "dlt_file"              /* filename for DLT intermediates */

#define ASPECT_RATIO 1.0               /* World dy/dx */

#define MENU_PAGE 20                   /* size of a single menu page */
#define GRAPHIC_NAME "Gait_Analysis"         /* name of graphic window */
#define GRAPH_WIDTH 1255               /* width of graph window */
#define GRAPH_HEIGHT 846               /* height of graph window */
#define XSEETHRU "xseethru -geometry 1255x846+0+0 &"
                        /* xseethru window command */
#define TEXT_WIDTH 1255                  /* width of text window */
#define TEXT_HEIGHT 102                  /* height of text window */
#define DEV_MAX_X 1280                  /* max device x coordinate */
#define DEV_MAX_Y 1024                  /* max device y coordinate */
```

```
#define FILE_MAX_X 640                  /* max file x coordinate */
#define FILE_MAX_Y 574                  /* max file y coordinate */
#define CURSOR_OFFSET_X 5               /* graphic cursor x offset */
#define CURSOR_OFFSET_Y 5               /* graphic cursor y offset */
#define TOP_BORDER 27                   /* mwm top border in pixels */
#define SIDE_BORDER 11                  /* mwm side border in pixels */
#define BOTTOM_BORDER 11                /* mwm bottom border in pixels */

#define DIRECTORY_ENTRIES 120           /* number of directory entries */
#define LIMB_DIRECTORY ""               /* limb data directory */
#define LIMB_PREFIX ""                  /* limb file prefix */
#define LIMB_SUFFIX ".limb"             /* limb file suffix */
#define NODE_DIRECTORY ""               /* node data directory */
#define NODE_PREFIX ""                  /* node file prefix */
#define NODE_SUFFIX ".node"             /* node file suffix */
#define FRAME_DIRECTORY ""              /* frame data directory */
#define FRAME_PREFIX ""                 /* frame file prefix */
#define FRAME_SUFFIX ""                 /* frame file suffix */
#define PICTURE_DIRECTORY ""            /* frame data directory */
#define PICTURE_PREFIX ""               /* frame file prefix */
#define PICTURE_SUFFIX ".pic"           /* frame file suffix */
#define RECON_DIRECTORY ""              /* recon data directory */
#define RECON_PREFIX ""                 /* recon file prefix */
#define RECON_SUFFIX_2D ".rec2d"        /* recon file suffix for 2d */
#define RECON_SUFFIX_3D1 ".rec3d1"      /* recon file suffix for 3d */
#define RECON_SUFFIX_3D2 ".rec3d2"      /* recon file suffix for 3d */
#define ANALYSIS_DIRECTORY ""           /* analysis data directory */
#define ANALYSIS_PREFIX ""              /* analysis file prefix */
#define ANALYSIS_SUFFIX_123 ".prn"      /* analysis file suffix for 123 */
#define ANALYSIS_SUFFIX_EXCEL ".txt"    /* analysis file suffix for EXCEL */
#define ANALYSIS_SUFFIX_SAS ".sas"      /* analysis file suffix for SAS */

#define COPPER_COLOUR 0.4,0.1,0.06      /* define some 'interesting' colours */
#define COPPER_SURFACE 8,0.6,0.5,0.1      /* and surface properties */
#define RUBBER_COLOUR 0.3,0.03,0.03
#define RUBBER_SURFACE 5,0.15,0.1,0.1
#define PLASTIC_COLOUR 0.6,0.05,0.05
#define PLASTIC_SURFACE 40,1.0,1.0,1.0
#define OBSIDIAN_COLOUR 0.01,0.01,0.01
#define OBSIDIAN_SURFACE 50,1.0,1.0,1.0
#define POTTERY_COLOUR 0.2,0.2,0.2
#define POTTERY_SURFACE 5,0.2,0.2,0.2
#define BRASS_COLOUR 0.4,0.2,0.08
#define BRASS_SURFACE 15,0.5,0.5,0.1

#define DEFAULT_COLOUR COPPER_COLOUR
#define DEFAULT_SURFACE COPPER_SURFACE

#define LA 0x0001                       /* define some light switches */
#define L1 0x0002
#define L2 0x0004
#define L3 0x0008

#define INIT_ZOOM 40.0                  /* camera field of view */
#define CAMERA_ZOOM_MIN 2.0             /* minimum field of view */
#define CAMERA_ZOOM_MAX 90.0            /* maximum field of view */
#define INIT_VIEW_POS 0.0,0.0,1.0       /* initial view position (RH) */
#define CAMERA_POS_RANGE -4.0,-4.0,-4.0,4.0,4.0,4.0    /* camera position range */
#define INIT_TARGET_POS 0.0,0.0,0.0     /* initial target position */
#define CAMERA_TARGET_RANGE -4.0,-4.0,-4.0,4.0,4.0,4.0    /* camera target range */
#define K_SENSITIVITY 10.0              /* # knob revolutions for range */
```

```
#define DC_CHAR_WIDTH 8                  /* device coordinate character size */
#define DC_CHAR_HEIGHT 10               /* device coordinate character height */
#define KEY_GRAPH 0.8                   /* relative position of graph/key join */
#define VDC_CHAR_SIZE 0.01               /* virtual device coordinate char size */
#define CHAR_SLIM 0.75                   /* character width slimming factor */

#define DEFAULT_DIAMETER 0.01              /* default limb diameter */

/* subroutine definitions */

#ifdef HP
#define CLEAR_TEXT printf("\33h\33J\n\33h")     /* clear text window */
#define CLEAR_GRAPH clear_view_surface(display),make_picture_current(display);
                        /* clear graphics */
#else
#define CLEAR_TEXT
#define CLEAR_GRAPH
#endif

/* external variables */

/* file names */

extern char node_file[STRING_SIZE];          /* node file name */
extern char limb_file[STRING_SIZE];          /* limb file name */

/* options flags */

extern int flag_2d;                    /* 2d only flag */
extern int flag_simple_reconstruction;         /* simple 3d reconstruction flag */
extern int fiducial_flag;              /* fiducial mark registration flag */
extern int frame_increment;              /* frame sequence increment */
extern int filtration_number;             /* filtration cutoff number */
extern int flag_filter;                  /* filter/smooth flag */
extern int smooth_number;               /* number of values in moving average */

/* 3d reconstruction values */

extern float l1[11];                   /* DLT parameters */
extern float l2[11];

extern int x_mirror_1;                   /* x mirroring flag */
extern float x_offset_1;                /* x offset of picture origin 1 */
extern float y_offset_1;                /* y offset of picture origin 1 */
extern float scale_factor_1;               /* scale factor from picture to world (m) 1 */
extern float rotation_1[2][2];               /* rotation matrix picture to world 1 */
extern int x_mirror_2;                   /* x mirroring flag */
extern float x_offset_2;                /* x offset of picture origin 2 */
extern float y_offset_2;                /* y offset of picture origin 2 */
extern float scale_factor_2;               /* scale factor from picture to world (m) 2 */
extern float rotation_2[2][2];               /* rotation matrix picture to world 2 */
extern int x_axis_source;              /* 3d axes source: 1 picture 1 x axis */
extern int y_axis_source;              /*                 2 picture 1 y axis */
extern int z_axis_source;              /*                 3 picture 2 x axis */
                        /*                 4 picture 2 y axis */
/* 2d conversion values */

extern float x_offset;                   /* x offset of picture origin */
extern float y_offset;                   /* y offset of picture origin */
extern float scale_factor;               /* scale factor from picture to world (m) */
extern float rotation[2][2];               /* rotation matrix picture to world */
```

```
/* registration values */

extern float fiducial_x;            /* global x registration value */
extern float fiducial_y;            /* global y registration value */
extern float correction_x;            /* framewide x registration */
extern float correction_y;            /* framewide y registration */

/* graphics device pointers */

extern int display;                 /* graphics display */
extern int locator;                 /* locator device (mouse) */
extern int knobs1,knobs2,knobs3;        /* knob box (3 rows) */
extern int bbox;                  /* button box */

#ifdef HP
/* window device pointers */

extern Window graphic_window;            /* graphic window */
extern Window text_window;             /* text window */
extern Window menu_window;             /* menu window */
extern Display *xdisplay;            /* x display pointer */
extern int xscreen;                /* screen number */

/* original hpterm window parameters */

extern unsigned int orig_width,orig_height;
extern int orig_x,orig_y;

/* global brightness and contrast values */

extern float brightness;            /* brightness */
extern float contrast;                /* contrast */
#endif
```

close_dev()

```
#include "params.h"

void close_dev()

/* This routine closes the various input devices */

{
    /* Screen */

    CLEAR_GRAPH;
    gclose(display);

    /* Knobs */

    gclose(knobs1);
    gclose(knobs2);
    gclose(knobs3);

    /* button box */

    gclose(bbox);

    /* close windows */

    XDestroyWindow(xdisplay,graphic_window);
```

```
    XDestroyWindow(xdisplay,menu_window);

    /* reset hpterm */

    XFlush(xdisplay);
    XMoveWindow(xdisplay,text_window,SIDE_BORDER,TOP_BORDER);
    XResizeWindow(xdisplay,text_window,orig_width,orig_height);
    XFlush(xdisplay);

    XCloseDisplay(xdisplay);
}
```

com()

```
#include "params.h"

void com(xpos,ypos,zpos,nnodes,nframes,nodes_per_seg,nsegs,seg_com,seg_mass,
body_comx,body_comy,body_comz,body_mass,comx,comy,comz)

/* calculate the centres of mass of the segments and the overall centres of mass */

float xpos[MAX_NODES][MAX_FRAMES];    /* x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];    /* y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];    /* z world coordinates */
int nnodes;                          /* number of nodes */
int nframes;                          /* number of frame */
int nodes_per_seg[NPS][MAX_SEGS];     /* nodes per segment */
int nsegs;                           /* number of segments */
float seg_com[MAX_SEGS];              /* relative segment radii */
float seg_mass[MAX_SEGS];            /* segment masses */
float body_comx[MAX_FRAMES];         /* x centre of mass of animal (m) */
float body_comy[MAX_FRAMES];         /* y centre of mass of animal (m) */
float body_comz[MAX_FRAMES];         /* z centre of mass of animal (m) */
float *body_mass;                     /* mass of animal (kg) */
float comx[MAX_SEGS][MAX_FRAMES];     /* x component of segment COM */
float comy[MAX_SEGS][MAX_FRAMES];     /* y component of segment COM */
float comz[MAX_SEGS][MAX_FRAMES];     /* z component of segment COM */
{
    int iframes;                     /* frame counter */
    int isegs;                      /* segment counter */
    int inps;                        /* nodes per segment counter */
    int nstart,nend;                 /* start and end nodes for a segment */

    /* find total mass */

    *body_mass=0;
    for (isegs=0;isegs<nsegs;isegs++)
    {
        *body_mass+=seg_mass[isegs];
    }

    /* loop over frames */

    for (iframes=0;iframes<nframes;iframes++)
    {
        body_comx[iframes]=0.0;
        body_comy[iframes]=0.0;
        body_comz[iframes]=0.0;

        /* loop over segments */
```

```
        for (isegs=0;isegs<nsegs;isegs++)
        {

            /* find start and end nodes */

            nstart=nodes_per_seg[0][isegs];
            nend=nodes_per_seg[1][isegs];

            /* perform the centre of mass calculation */

            limb_com(xpos[nstart][iframes],ypos[nstart][iframes],
                zpos[nstart][iframes],
                xpos[nend][iframes],ypos[nend][iframes],zpos[nend][iframes],
                seg_com[isegs],
                &comx[isegs][iframes],&comy[isegs][iframes],
                &comz[isegs][iframes]);

            body_comx[iframes]+=seg_mass[isegs]*comx[isegs][iframes];
            body_comy[iframes]+=seg_mass[isegs]*comy[isegs][iframes];
            body_comz[iframes]+=seg_mass[isegs]*comz[isegs][iframes];
        }

        body_comx[iframes]=body_comx[iframes]/(*body_mass);
        body_comy[iframes]=body_comy[iframes]/(*body_mass);
        body_comz[iframes]=body_comz[iframes]/(*body_mass);
    }
}
```

dgnode()

```
#include "params.h"

void dgnode(title,nodes,nnodes,xpos,ypos,zpos,nframe)

/* this routine prompts the user for the positions of the nodes named in array nodes */

char title[STRING_SIZE];              /* file title line */
char nodes[MAX_NODES][STRING_SIZE];    /* names of the nodes of the model */
int nnodes;                           /* the number of nodes */
float xpos[MAX_NODES][MAX_FRAMES];    /* the x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];    /* the y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];    /* the z world coordinates */
int *nframe;                          /* the number of frames */

{

    int inodes;                       /* node counter */
    int iret;                          /* menu return value */
    static char menu1[][STRING_SIZE]=    /* menu */
    {
        "Next frame",
        "Repeat last frame",
        "Exit"
    };

    /* get into right graphics mode */

    CLEAR_GRAPH;

    /* loop round frames */
```

```
    do
    {

        /* print out file title */

        printf("%s\n",title);
        printf("Frame #%d\n",*nframe);

        /* read in picture file */

        readpic();

        /* fiducial point */

        if (fiducial_flag)
        {
            printf("Select fiducial point\n");
            digrd(&correction_x,&correction_y);
        }

        /* loop round the nodes */

        for (inodes=0;inodes<nnodes;inodes++)
        {

            /* prompt for node */

            printf("%s\n",nodes[inodes]);

            /* read coordinates */

            if (flag_2d==TRUE)
            {
                read2d(&xpos[inodes][*nframe],&ypos[inodes][*nframe]);
                zpos[inodes][*nframe]=0.0;
            }
            else
            {
                read3d(&xpos[inodes][*nframe],&ypos[inodes][*nframe],
                    &zpos[inodes][*nframe]);
            }
        }

        while ((iret=menu("Select option:",menu1,3))==0);

        if (iret!=2) *nframe+=1;

    } while (iret!=3);

    /* finished */

    CLEAR_GRAPH;

}
```

different()

```
#include "params.h"

/* this routine differentiates twice by a finite difference method, the array y on x */
```

```
void different(x,y,n,dy,d2y,angle_flag)
float x[];              /* array of x values */
float y[];              /* array of y values */
int n;                      /* number of values */
float dy[];             /* diff'd values of y, minus 1 at each end */
float d2y[];             /* double diff'd value of y, minus 2 at each end */
int angle_flag;         /* flag for angular correction */
{
    int i;                  /* counter */
    float dely;         /* y difference */
    float del2y;         /* y difference for 2nd order differentiation */

    for (i=1;i<n-1;i++)
    {
        dely=y[i+1]-y[i-1];
        del2y=y[i+1]-2*y[i]+y[i-1];
        if (angle_flag)
        {
            if (dely>M_PI) dely-=(2*M_PI);
            if (dely<=(-M_PI)) dely+=(2*M_PI);

            if (del2y>M_PI) del2y-=(2*M_PI);
            if (del2y<=(-M_PI)) del2y+=(2*M_PI);
        }
        dy[i]=dely/(x[i+1]-x[i-1]);
        d2y[i]=del2y/(((x[i+1]-x[i-1])/2.0)*((x[i+1]-x[i-1])/2.0));
}


dy[0]=0.0;
dy[n-1]=0.0;
d2y[0]=0.0;
d2y[n-1]=0.0;
}
```

digrd()

```
#include "params.h"

void digrd(x,y)

/* This routine reads the position of the locator device */

float *x,*y;                    /* Position of pointer in device coords */
{
    static char marker[9][9]=    /* marker bit map */
    {
        0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0,-1,-1,-1, 0, 0, 0,
        0, 0,-1,-1,-1,-1,-1, 0, 0,
        0,-1,-1, 0, 0, 0,-1,-1, 0,
        0,-1,-1, 0,-1, 0,-1,-1, 0,
        0,-1,-1, 0, 0, 0,-1,-1, 0,
        0, 0,-1,-1,-1,-1,-1, 0, 0,
        0, 0, 0,-1,-1,-1, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0
    };
    Window window_return;
    Window root_return;
    int root_x,root_y;
    int win_x,win_y;
```

```
unsigned int mask;
float colourTable[256][3];
float red,green,blue;
int i;
int offset,range;
float grey;
int dum,button,valid;
float dummy;
float oldbrightness=0.0,oldcontrast=0.0;

/* wait for no buttons to be pressed */

do
{
    XQueryPointer(xdisplay,graphic_window,&window_return,&root_return,
        &root_x,&root_y,&win_x,&win_y,&mask);
} while (mask!=0);

/* get pointer position etc */

do
{
    XQueryPointer(xdisplay,graphic_window,&window_return,&root_return,
        &root_x,&root_y,&win_x,&win_y,&mask);

    /* interactive contrast/brightness control */

    sample_locator(knobs3,1,&valid,&brightness,&contrast,&dummy);
    if (brightness!=oldbrightness || contrast!=oldcontrast)
    {
        oldbrightness=brightness;
        oldcontrast=contrast;

        offset=(int)512.0*(0.5-brightness);
        range=512*(1.0-contrast);
        if (range==0) range=1;

        for (i=0;i<256;i++)
        {
            if (i<offset)
            {
                grey=0.0;
            }
            else
            {
                if (i<(offset+range))
                {
                    grey=(float)(i-offset)/(float)range;
                }
                else
                {
                    grey=1.0;
                }
            }
            colourTable[i][0]=grey;
            colourTable[i][1]=grey;
            colourTable[i][2]=grey;
        }
        define_color_table(display,0,256,colourTable);
        make_picture_current(display);
    }
```

```
    } while (mask!=Button3Mask);
    *x=(float)root_x*ASPECT_RATIO;
    *y=(float)(DEV_MAX_Y-root_y-1);
#if 0
    printf("(%f,%f)\n",*x,*y);
#endif
    /* plot a marker */

    dcblock_write(display,root_x-CURSOR_OFFSET_X,root_y-CURSOR_OFFSET_Y,
        9,9,marker,0);
    make_picture_current(display);

}
```

displa()

```
#include "params.h"

void displa(title,nodes,nnodes,xpos,ypos,zpos,
    xpos_filt,ypos_filt,zpos_filt,nframe,
    segs,nsegs,nodes_per_seg,seg_radii)

/* this routine displays the node data as defined by limb model data */

char title[STRING_SIZE];                /* file title line */
char nodes[MAX_NODES][STRING_SIZE];     /* names of the nodes */
int nnodes;                             /* the number of nodes */
float xpos[MAX_NODES][MAX_FRAMES];      /* the x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];      /* the y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];      /* the z world coordinates */
float xpos_filt[MAX_NODES][MAX_FRAMES]; /* the filtered x world coordinates */
float ypos_filt[MAX_NODES][MAX_FRAMES]; /* the filtered y world coordinates */
float zpos_filt[MAX_NODES][MAX_FRAMES]; /* the filtered z world coordinates */
int nframe;                             /* the number of frames */
char segs[MAX_SEGS][STRING_SIZE];       /* names of the segments */
int nsegs;                              /* the number of segments */
int nodes_per_seg[NPS][MAX_SEGS];       /* nodes for each segment */
float seg_radii[NPS][MAX_SEGS];         /* relative radii of ends of segments */

{
    camera_arg camera;              /* view camera structure */
    char fname[STRING_SIZE];        /* picture filename */
    char filename[STRING_SIZE];     /* full filename */
    int buffer=0;                   /* double buffer switch */
    int button;                     /* button number */
    int dum;                        /* dummy value */
    int anode;                      /* number of target fixed node */
    int iframe;                     /* frame counter */
    int ifix;                       /* fix reference point to node flag */
    int pause;                      /* pause between frames flag */
    int min_frame;                  /* min frame number for repeat loop */
    int max_frame;                  /* max frame number for repeat loop */
    float krange;                   /* knob range for frame select */
    int iret;                       /* menu option main */
    int icol;                       /* menu option colour */
    static int smooth_flag=TRUE;    /* smoothed data flag */
    static int solid_flag=TRUE;     /* interior fill flag */
    static int flag_fixed_diameter=TRUE;    /* fixed limb display radii */
    static char menu1[][STRING_SIZE]=       /* options menu */
    {
        "Start sequence",
```

```
        "Select options",
        "Exit"
    };
    static char menu2[][STRING_SIZE]=    /* colours menu */
    {
        "Colour: Copper",
        "        Rubber",
        "        Plastic",
        "        Obsidian",
        "        Pottery",
        "        Brass",
        "Style : Hollow",
        "Raw data",
        "Variable limb radius",
        "Exit"
    };

    /* get into correct graphics mode */

    CLEAR_GRAPH;
    vdc_extent(display,0.0,0.0,0.0,1.25,1.0,1.0);
    mapping_mode(display,ISOTROPIC);

    /* enable button box events */

    enable_events(bbox,CHOICE,1);

    /* set up knob ranges and sensitivity */

    krange=(float)nframe+0.1;
    mapping_mode(knobs1,DISTORT);
    mapping_mode(knobs2,DISTORT);
    mapping_mode(knobs3,DISTORT);
    vdc_extent(knobs1,CAMERA_ZOOM_MIN,0.05,1.05,CAMERA_ZOOM_MAX,
        krange-1.0,krange);
    vdc_extent(knobs2,CAMERA_TARGET_RANGE);
    vdc_extent(knobs3,CAMERA_POS_RANGE);
    set_p1_p2(knobs1,FRACTIONAL,0.0,0.0,0.0,K_SENSITIVITY,K_SENSITIVITY,
        K_SENSITIVITY);
    set_p1_p2(knobs2,FRACTIONAL,0.0,0.0,0.0,K_SENSITIVITY,K_SENSITIVITY,
        K_SENSITIVITY);
    set_p1_p2(knobs3,FRACTIONAL,0.0,0.0,0.0,K_SENSITIVITY,K_SENSITIVITY,
        K_SENSITIVITY);
    set_locator(knobs1,1,INIT_ZOOM,0.1,krange);
    set_locator(knobs2,1,INIT_TARGET_POS);
    set_locator(knobs3,1,INIT_VIEW_POS);

    /* Set up viewing position and transformation */


    camera.upx=0.0;
    camera.upy=1.0;
    camera.upz=0.0;
    camera.front=camera.back=0.0;
    camera.projection=CAM_PERSPECTIVE;

    /* Set up drawing attributes */

    if (solid_flag==TRUE) interior_style(display,INT_SOLID,FALSE);
    else interior_style(display,INT_HOLLOW,TRUE);
    text_color(display,RED);
    fill_color(display,DEFAULT_COLOUR);
    perimeter_color(display,WHITE);
```

```
surface_model(display,TRUE,DEFAULT_SURFACE);
shade_mode(display,CMAP_FULL|INIT,TRUE);
light_ambient(display,GREY);
light_source(display,1,DIRECTIONAL,LIGHT_BLUE,0.0,0.0,-1.0);
light_source(display,2,DIRECTIONAL,LIGHT_STEEL_BLUE,0.0,1.0,0.0);
light_source(display,3,DIRECTIONAL,DARK_SLATE_BLUE,1.0,0.0,0.0);
light_switch(display,LA|L1|L2|L3);

dccharacter_width(display,DC_CHAR_WIDTH);
dccharacter_height(display,DC_CHAR_HEIGHT);

/* set up display buffering */

double_buffer(display,TRUE|INIT,12);
if (hidden_surface(display,TRUE,TRUE)!=1)
{
    printf("\n\nInsufficient space for z buffer\n\n");
    exit(-1);
}

while ((iret=menu("Select option:",menu1,3))!=3)
{

    switch (iret)
    {
    case 1:

        /* start sequence */

        if ((ifix=yesno("Fix target position to node?"))==TRUE)
        {
            while ((anode=menu("Select node:",nodes,nnodes))==0);
            anode--;
        }

        /* pause ? */

        pause=yesno("Pause between frames?");

        /* loop until button 32 */

        frame_range(display,&min_frame,&max_frame);
        iframe=min_frame;
        do
        {

            /* loop until button 1 or 32 */

            do
            {

                /* print out frame range */

                frame_range(display,&min_frame,&max_frame);

                /* print out help */

                draw_help(display);

                /* test smooth flag */

                if (smooth_flag)
```

```
{

    /* move camera */

    if (ifix) set_locator(knobs2,1,
        xpos_filt[anode][iframe],
        ypos_filt[anode][iframe],
        zpos_filt[anode][iframe]);
    move_cam(display,&camera);

    /* draw figure */

    draw_fig(display,xpos_filt,ypos_filt,zpos_filt,
        nnodes,iframe,nodes_per_seg,nsegs,
        seg_radii,flag_fixed_diameter);
}
else
{

    /* move camera */

    if (ifix) set_locator(knobs2,1,
        xpos[anode][iframe],
        ypos[anode][iframe],
        zpos[anode][iframe]);
    move_cam(display,&camera);

    /* draw figure */

    draw_fig(display,xpos,ypos,zpos,
        nnodes,iframe,nodes_per_seg,nsegs,
        seg_radii,flag_fixed_diameter);
}

/* switch buffer plane */

dbuffer_switch(display,buffer=!buffer);

/* read button box */

if (read_choice_event(bbox,&dum,&dum,&button,&dum,&dum)!=0)
    button=0;

/* check for screen copy */

if (button==2)
{

    /* write out picture file */

    printf("Input picture file name : ");
    scanf("%s",fname);
    strcpy(filename,PICTURE_DIRECTORY);
    strcat(filename,PICTURE_PREFIX);
    strcat(filename,fname);
    strcat(filename,PICTURE_SUFFIX);

    bitmap_to_file(display,TRUE,0,0,filename,
        TRUE,0.0,0.0,0,0,TRUE);
    printf("%s written successfully\n",
        filename);
}
```

```
            } while ((button!=1 && button!=32) && (pause==TRUE));

            iframe++;
            if (iframe>=max_frame) iframe=min_frame;

        } while (button!=32);
        break;

    case 2:

        /* select options */

        do
        {
            /* set up correct menu prompts */

            if (solid_flag==TRUE) strcpy(menu2[6],"Style : Hollow");
            else strcpy(menu2[6],"Style : Solid");

            if (smooth_flag==TRUE) strcpy(menu2[7],"Raw data");
            else strcpy(menu2[7],"Smoothed data");

            if (flag_fixed_diameter==TRUE)
                strcpy(menu2[8],"Variable limb radius");
            else strcpy(menu2[8],"Fixed limb radius");

            icol=menu("Select option:",menu2,10);

            switch (icol)
            {

            case 1:
                fill_color(display,COPPER_COLOUR);
                perimeter_color(display,COPPER_COLOUR);
                surface_model(display,TRUE,COPPER_SURFACE);
                break;

            case 2:
                fill_color(display,RUBBER_COLOUR);
                perimeter_color(display,RUBBER_COLOUR);
                surface_model(display,TRUE,RUBBER_SURFACE);
                break;

            case 3:
                fill_color(display,PLASTIC_COLOUR);
                perimeter_color(display,PLASTIC_COLOUR);
                surface_model(display,TRUE,PLASTIC_SURFACE);
                break;

            case 4:
                fill_color(display,OBSIDIAN_COLOUR);
                perimeter_color(display,WHITE);
                surface_model(display,TRUE,OBSIDIAN_SURFACE);
                break;

            case 5:
                fill_color(display,POTTERY_COLOUR);
                perimeter_color(display,POTTERY_COLOUR);
                surface_model(display,TRUE,POTTERY_SURFACE);
                break;

            case 6:
```

```
                fill_color(display,BRASS_COLOUR);
                perimeter_color(display,BRASS_COLOUR);
                surface_model(display,TRUE,BRASS_SURFACE);
                break;

            case 7:
                if (solid_flag==TRUE)
                {
                    interior_style(display,INT_HOLLOW,TRUE);
                    solid_flag=FALSE;
                }
                else
                {
                    interior_style(display,INT_SOLID,FALSE);
                    solid_flag=TRUE;
                }
                break;

            case 8:
                if (smooth_flag==TRUE) smooth_flag=FALSE;
                else smooth_flag=TRUE;
                break;

            case 9:
                if (flag_fixed_diameter==TRUE) flag_fixed_diameter=FALSE;
                else flag_fixed_diameter=TRUE;
                break;
            }
        } while (icol!=10);
    }
}

    /* disable button box events */

    disable_events(bbox,CHOICE,1);

    /* back to text mode */

    hidden_surface(display,FALSE,FALSE);
    double_buffer(display,FALSE,0);
    flush_matrices(display);
    CLEAR_GRAPH;

}
```

domenu()

```
/* menu routine using Xrlib toolkit */

#include "params.h"

int domenu(title,prompts,nprompt)
char title[STRING_SIZE];              /* menu title */
char prompts[][STRING_SIZE];     /* menu prompts */
int nprompt;                          /* number of prompts */
{
#if 1
    int iret;                          /* return value */
    int iprompt;                      /* prompt counter */
    XEvent xinput;                  /* x event structure */
    xrEvent *input;                  /* input structure */
```

```
xrMenu *menu;                          /* menu pointer */
xrMenuInfo menuinfo;                  /* menu information structure */
INT8 *menuitems[MENU_PAGE+1];/* array of pointers to meu items */

/* cast xr event onto x event */

input=(xrEvent *)&xinput;

/* get menu items */

for (iprompt=0;iprompt<nprompt;iprompt++)
    menuitems[iprompt]=(INT8 *)prompts[iprompt];

/* get values into menu information structure */

menuinfo.menuTitle=title;
menuinfo.menuItems=menuitems;
menuinfo.numItems=(INT32)nprompt;
menuinfo.menuContext=NULL;
menuinfo.menuId=0;
menuinfo.menuStyle=0;

/* create and display menu */

menu=XrMenu(NULL,MSG_NEW,&menuinfo);
XrMenu(menu,MSG_ACTIVATEMENU,menu_window);

input->type=ButtonRelease;
xrMenuItemSelect=XrRIGHTBUTTONDOWN;
XrMenu(menu,MSG_EDIT,input);

/* get input */

while(1)
{
    if (XrInput(0,MSG_BLKHOTREAD,input)!=FALSE)
    {
        if (input->type==XrXRAY && input->inputType==XrMENU)
        {
            iret=input->value3+1;
            break;
        }
        else
        {
            iret=0;
            break;
        }
    }
}

/* remove menu */

    XrMenu(menu,MSG_DEACTIVATEMENU,menu_window);
    XrMenu(menu,MSG_FREE,NULL);
    xrMenuItemSelect=XrRIGHTBUTTONUP;

    return(iret);
#else
    int iprompt;                    /* prompt counter */
    int iret;                        /* return value */

    CLEAR_TEXT;
```

```
    printf("%s\n\n",title);

    for (iprompt=0;iprompt<nprompt;iprompt++)
    {
        printf("%d. %s\n",iprompt+1,prompts[iprompt]);
    }

    printf("\nInput selection: ");
    scanf("%d",&iret);

    if (iret<1 || iret>nprompt) iret=0;
    return(iret);
#endif
}
```

draw_fig()

```
#include "params.h"

void draw_fig(fd,xpos,ypos,zpos,nnodes,iframe,nodes_per_seg,nsegs,seg_radii,
    flag_fixed_diameter)

/* draw the figure */

int fd;                                  /* output file descriptor */
float xpos[MAX_NODES][MAX_FRAMES];      /* x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];      /* y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];      /* z world coordinates */
int nnodes;                              /* number of nodes */
int iframe;                              /* frame counter */
int nodes_per_seg[NPS][MAX_SEGS];       /* nodes per segment */
int nsegs;                               /* number of segments */
float seg_radii[NPS][MAX_SEGS];          /* radii of segment ends */
int flag_fixed_diameter;                 /* fixed segment diameter flag */
{
    int isegs;                           /* segment counter */
    int inps;                               /* nodes per segment counter */
    int nstart,nend;                        /* start and end nodes for a segment */
    char string[STRING_SIZE];           /* frame number string */

    /* write out frame number */

    sprintf(string,"Frame # %d",iframe);
    dctext(display,SIDE_BORDER+10,GRAPH_HEIGHT+TOP_BORDER-10,string);

    /* loop over segments */

    for (isegs=0;isegs<nsegs;isegs++)
    {

        /* find start and end nodes */

        nstart=nodes_per_seg[0][isegs];
        nend=nodes_per_seg[1][isegs];

        /* draw the limb */

        draw_limb(fd,xpos[nstart][iframe],ypos[nstart][iframe],
            zpos[nstart][iframe],xpos[nend][iframe],
            ypos[nend][iframe],zpos[nend][iframe],
            seg_radii[0][isegs],seg_radii[1][isegs],
```

```
            flag_fixed_diameter);
    }
}
```

draw_help()

```
#include "params.h"

/* this routine draws up the help prompt on the rhs of screen */

void draw_help(fd)

int fd;                                 /* output device file pointer */

{
    dctext(display,GRAPH_WIDTH+SIDE_BORDER-80,TOP_BORDER+20,"Buttons:");
    dctext(display,GRAPH_WIDTH+SIDE_BORDER-80,TOP_BORDER+40,"1.  Step");
    dctext(display,GRAPH_WIDTH+SIDE_BORDER-80,TOP_BORDER+60,"2.  Save");
    dctext(display,GRAPH_WIDTH+SIDE_BORDER-80,TOP_BORDER+80,"32. Quit");
}
```

draw_limb()

```
#include "params.h"

/* Subroutine draw_limb draws an octagonal truncated cone */

/* Axis from (x1,y1,z1) to (x2,y2,z2), with proximal diameter prox_diam and */
/* distal diameter dist_diam */

/* converts axis system to right handed */

void draw_limb(fd,x1,y1,z1,x2,y2,z2,rel_prox_radius,rel_dist_radius,
    flag_fixed_diameter)

int fd;                                 /* output file descriptor */
double x1,y1,z1;                         /* axis start coordinates */
double x2,y2,z2;                         /* axis end coordinates */
double rel_prox_radius;                 /* proximal radius / length */
double rel_dist_radius;                 /* distal radius / length */
int flag_fixed_diameter;                 /* fixed limb diameter flag */

{

    /* Define octagon data */

    static float octagon[8][2]=         /* define an octagon */
    {
        0.5,0.0,
        M_SQRT2/4.0,M_SQRT2/4.0,
        0.0,0.5,
        -M_SQRT2/4.0,M_SQRT2/4.0,
        -0.5,0.0,
        -M_SQRT2/4.0,-M_SQRT2/4.0,
        0.0,-0.5,
        M_SQRT2/4.0,-M_SQRT2/4.0
    };
    static float prox_octagon[8][3]=    /* proximal octagon polygon */
    {
        24*0.0
```

```
};
static float dist_octagon[8][3]=    /* distal octagon polygon */
{
    24*0.0
};
static float figure_2d[4][3]=    /* 2d quadralateral */
{
    12*0.0
};
static float quadr[8][4][6]=    /* side quadralaterals */
{
    192*0.0
};
static float mirror_z[4][4]=    /* z inversion to convert to RH axes */
{
    1.0,0.0,0.0,0.0,
    0.0,1.0,0.0,0.0,
    0.0,0.0,-1.0,0.0,
    0.0,0.0,0.0,1.0
};
double prox_diam,dist_diam;         /* segment diameters */
double x,y,z;                         /* axis vector */
double length;                      /* length of axis vector */
double z_rot,y_rot;                  /* z and x axis rotations */
float transform[4][4];              /* transform matrix */
int vertex,pvertex;                  /* vertex counters */


/* Calculate axis vector */

x=x2-x1;
y=y2-y1;
z=z2-z1;

/* Calculate length of axis vector */

length=sqrt(x*x+y*y+z*z);

/* check diameter model */

if (flag_fixed_diameter==TRUE)
{

    /* set fixed diameters */

    prox_diam=DEFAULT_DIAMETER;
    dist_diam=DEFAULT_DIAMETER;
}
else
{

    /* Calculate diameters */

    prox_diam=2.0*length*rel_prox_radius;
    dist_diam=2.0*length*rel_dist_radius;
}

if (length!=0.0)
{

    /* Calculate the z axis rotation */
```

```
        if (y>0.0)
            z_rot=acos(y/length)-M_PI/2.0;
        else
            z_rot=M_PI/2.0-acos(-y/length);

        /* Calculate the y axis rotation */

        if (z==0.0)
            y_rot= x<0.0 ? M_PI:0.0;
        else
        {
            if (x==0.0)
                y_rot= z<0.0 ? M_PI/-2.0:M_PI/2.0;
            else
            {
                if (x<0.0)
                {
                    if (z<0.0)
                        y_rot=M_PI+atan(z/x);
                    else
                        y_rot=M_PI-atan(-z/x);
                }
                else
                {
                    if (z<0.0)
                        y_rot=(-atan(-z/x));
                    else
                        y_rot=atan(z/x);
                }
            }
        }

        /* Perform required transformations in reverse order */

        /* z inversion for RH axes conversion */

        concat_transformation3d(fd,mirror_z,PRE,PUSH);

        /* Translation */

        translate3d(x1,y1,z1,transform);
        concat_transformation3d(fd,transform,PRE,REPLACE);

        /* y axis rotation */

        rotate3d('y',y_rot,transform);
        concat_transformation3d(fd,transform,PRE,REPLACE);

        /* z axis rotation */

        rotate3d('z',z_rot,transform);
        concat_transformation3d(fd,transform,PRE,REPLACE);

        /* Calculate sized proximal and distal octagons */
        /* Also side quadralaterals */

        for (vertex=0;vertex<8;vertex++)
        {
            prox_octagon[vertex][1]=octagon[vertex][0]*(float)prox_diam;
            prox_octagon[vertex][2]=octagon[vertex][1]*(float)prox_diam;

            /* Reverse direction for distal polygon to preserve */
```

```
        /* anti-clockwise order from outside */

        dist_octagon[7-vertex][0]=(float)length;
        dist_octagon[7-vertex][1]=octagon[vertex][0]*(float)dist_diam;
        dist_octagon[7-vertex][2]=octagon[vertex][1]*(float)dist_diam;

        /* Side quadralaterals */

        if (vertex==0)
            pvertex=7;
        else
            pvertex=vertex-1;

        quadr[vertex][0][1]=quadr[vertex][0][4]=octagon[vertex][0]*
            (float)prox_diam;
        quadr[vertex][0][2]=quadr[vertex][0][5]=octagon[vertex][1]*
            (float)prox_diam;
        quadr[vertex][1][1]=quadr[vertex][1][4]=octagon[pvertex][0]*
            (float)prox_diam;
        quadr[vertex][1][2]=quadr[vertex][1][5]=octagon[pvertex][1]*
            (float)prox_diam;
        quadr[vertex][2][0]=(float)length;
        quadr[vertex][2][1]=quadr[vertex][2][4]=octagon[pvertex][0]*
            (float)dist_diam;
        quadr[vertex][2][2]=quadr[vertex][2][5]=octagon[pvertex][1]*
            (float)dist_diam;
        quadr[vertex][3][0]=(float)length;
        quadr[vertex][3][1]=quadr[vertex][3][4]=octagon[vertex][0]*
            (float)dist_diam;
        quadr[vertex][3][2]=quadr[vertex][3][5]=octagon[vertex][1]*
            (float)dist_diam;
    }

    /* do all calculation for 2d and 3d versions to keep the timing approx */
    /* equal */

    if (flag_2d==TRUE)
    {
        figure_2d[0][0]=figure_2d[1][0]=(float)length;
        figure_2d[0][1]=figure_2d[3][1]=(float)(-dist_diam*0.5);
        figure_2d[1][1]=figure_2d[2][1]=(float)dist_diam*0.5;
        vertex_format(fd,0,0,0,0,COUNTER_CLOCKWISE);
        polygon3d(fd,figure_2d,4,FALSE);
        vertex_format(fd,0,0,0,0,CLOCKWISE);
        polygon3d(fd,figure_2d,4,FALSE);
    }
    else
    {

        /* Draw the two octagons */

        vertex_format(fd,0,0,0,0,COUNTER_CLOCKWISE);
        polygon3d(fd,prox_octagon,8,FALSE);
        polygon3d(fd,dist_octagon,8,FALSE);

        /* Draw the eight quadralaterals */

        vertex_format(fd,3,3,0,0,COUNTER_CLOCKWISE);
        for (vertex=0;vertex<8;vertex++)
        {
            polygon3d(fd,quadr[vertex],4,FALSE);
        }
```

```
        }

        /* Return stack to previous condition */

        pop_matrix(fd);
    }
}
```

d_graph()

```
#include "params.h"

void d_graph(device,title,x_label,y_label,sx_point,sy_point,npoint,nline,key,ennum)

/* this routine plots a general line graph of the data contained in x_point and y_point
*/

/* calls double precision NAG subroutines */

int device;                              /* output device pointer */
char title[STRING_SIZE];                  /* graph title */
char x_label[STRING_SIZE];               /* x axis label */
char y_label[STRING_SIZE];               /* y axis label */
float sx_point[MAX_LINES][MAX_POINTS];    /* x coordinates */
float sy_point[MAX_LINES][MAX_POINTS];    /* y coordinates */
int npoint;                               /* number of points */
int nline;                               /* number of lines */
char key[MAX_LINES][STRING_SIZE];        /* key for multiple lines */
int ennum;                               /* number markers flag */

{
    int iline;                           /* line counter */
    int ipoint;                           /* point counter */
    int iwidth;                           /* width of integer field */
    double xmin,xmax;                       /* x coordinate range */
    double ymin,ymax;                       /* y coordinate range */
    double vxmin,vxmax;                   /* x viewport */
    double vymin,vymax;                   /* y viewport */
    int margin;                           /* margin flag */
    int lstring;                           /* string lengths */
    int fout;                               /* fortran out channel */
    double temp;                           /* temporary real value */
    int itemp;                           /* temporary integer */
    int icntl;                           /* control value */
    int itype;                           /* graph type */
    int isym;                               /* marker type */
    int ifail;                           /* fail flag */
    double dx,dy;                           /* character spacing */
    double cwidth,cheight;               /* character size */
    int key_len;                          /* length of key string */
    double cx,cy;                          /* key string position */
    double yinc;                          /* key spacing */
    double range;                          /* y extent of key */
    double xsym,ysym;                      /* symbol position */
    double x_point[MAX_LINES][MAX_POINTS];/* x coordinates */
    double y_point[MAX_LINES][MAX_POINTS];/* y coordinates */
    char string1[80],string2[80];

    /* convert to double precision */

    for (iline=0;iline<nline;iline++)
```

```
{
    for (ipoint=0;ipoint<npoint;ipoint++)
    {
        x_point[iline][ipoint]=(double)sx_point[iline][ipoint];
        y_point[iline][ipoint]=(double)sy_point[iline][ipoint];
    }
}

/* initialize secondary graphics system */

fout=6;
itemp=1;
j06vbf(&itemp,&fout);
itemp=0;
strcpy(string1,getenv("SB_OUTDEV"));
strcpy(string2,getenv("SB_OUTDRIVER"));
cnagsti(string1,string2,&itemp);
j06waf();

/* find coordinate range and key string length */

key_len=0;
xmin=xmax=x_point[0][0];
ymin=ymax=y_point[0][0];
for (iline=0;iline<nline;iline++)
{
    for (ipoint=0;ipoint<npoint;ipoint++)
    {
        temp=x_point[iline][ipoint];
        if (temp<xmin) xmin=temp;
        if (temp>xmax) xmax=temp;
        temp=y_point[iline][ipoint];
        if (temp<ymin) ymin=temp;
        if (temp>ymax) ymax=temp;
    }

    /* while here, find out max key string length */

    itemp=strlen(key[iline]);
    key_len = itemp>key_len ? itemp:key_len;
}

/* set up data mapping */

margin=1;
j06wbf(&xmin,&xmax,&ymin,&ymax,&margin);
vxmin=0.0;
vxmax=KEY_GRAPH;
if (device==display)
{
    vymin=0.18;
    vymax=0.95;
}
else
{
    vymin=0.0;
    vymax=1.0;
}
j06wcf(&vxmin,&vxmax,&vymin,&vymax);

/* labels */
```

```
    cj06ahf(title);
    icntl=1;
    cj06ajf(&icntl,x_label);
    icntl=2;
    cj06ajf(&icntl,y_label);

    /* draw axes */

    j06aaf();

    /* draw lines */

    itype=2;
    for (iline=0;iline<nline;iline++)
    {
        isym=iline+1;
        ifail=0;
        j06baf(x_point[iline],y_point[iline],&npoint,&itype,&isym,&ifail);
    }

    /* check if markers need to be ennumerated */

    if (ennum)
    {

        /* character sizes */

        dx=(xmax-xmin)*VDC_CHAR_SIZE;
        dy=0;
        cwidth=dx*CHAR_SLIM;
        cheight=(ymax-ymin)*VDC_CHAR_SIZE;
        j06ylf(&dx,&dy);
        j06ykf(&cwidth,&cheight);

        for (iline=0;iline<nline;iline++)
        {
            for (ipoint=0;ipoint<npoint;ipoint++)
            {
                cx=x_point[iline][ipoint]+dx/2.0;
                cy=y_point[iline][ipoint]+cheight/2.0;
                j06yaf(&cx,&cy);
                iwidth= ipoint==0 ? 1:(int)log10((float)ipoint)+1;
                j06zbf(&ipoint,&iwidth);
            }
        }
    }

    /* now do key */

    xmin=0.0;
    xmax=1.0;
    ymin=0.0;
    ymax=1.0;
    margin=0;
    j06wbf(&xmin,&xmax,&ymin,&ymax,&margin);
    range=nline/((float)MAX_LINES*2.0);
    vxmin=KEY_GRAPH;
    vxmax=1.0;
    vymin=(1.0-range)/2.0;
    vymax=1.0-vymin;
    j06wcf(&vxmin,&vxmax,&vymin,&vymax);
```

```
    /* set character size and spacing */

    dx=1.0/(float)(key_len+4);
    dy=0.0;
    cwidth=dx*CHAR_SLIM;
    cheight=1.0/(float)(2*nline);
    j06ylf(&dx,&dy);
    j06ykf(&cwidth,&cheight);
    j06yjf(&cwidth);

    /* draw key */

    xsym=dx;
    cx=dx*3;
    yinc=1.0/nline;
    ysym=1.0-yinc/2.0;
    for (iline=0;iline<nline;iline++)
    {
        isym=iline+1;
        j06yaf(&xsym,&ysym);
        j06ygf(&isym);
        cy=ysym-cheight/2.0;
        j06yaf(&cx,&cy);
        cj06zaf(key[iline]);
        ysym-=yinc;
    }

    /* finished */

    j06wzf();
}
```

energetics()

```
/* routine to calculate potential energy, translational kinetic energy and */
/* rotational kinetic energy for all the segments */

#include "params.h"

void energetics(comy,comxvel,comyvel,comzvel,xavel,yavel,zavel,seg_mass,seg_moi,
    nsegs,nframes,seg_PE,seg_LKE,seg_RKE)

float comy[MAX_SEGS][MAX_FRAMES];          /* y component of segment COM */
float comxvel[MAX_SEGS][MAX_FRAMES];         /* x component of segment COM vel */
float comyvel[MAX_SEGS][MAX_FRAMES];         /* y component of segment COM vel */
float comzvel[MAX_SEGS][MAX_FRAMES];         /* z component of segment COM vel */
float xavel[MAX_SEGS][MAX_FRAMES];          /* calculated angular velocities (rad/s) */
float yavel[MAX_SEGS][MAX_FRAMES];
float zavel[MAX_SEGS][MAX_FRAMES];
float seg_mass[MAX_SEGS];                  /* array of segment masses */
float seg_moi[MAX_SEGS];                     /* array of segment MOIs */
int nsegs;                                  /* Number of segments */
int nframes;                                 /* number of frames */
float seg_PE[MAX_SEGS][MAX_FRAMES];         /* segment potential energy */
float seg_LKE[MAX_SEGS][MAX_FRAMES];         /* segment linear kinetic energy */
float seg_RKE[MAX_SEGS][MAX_FRAMES];         /* segment rotational kinetic energy */
{
    int isegs;                  /* segment counter */
    int iframes;                  /* frame counter */

    /* loop round segments */
```

```
for (isegs=0;isegs<nsegs;isegs++)
{

    /* loop round frames */

    for (iframes=0;iframes<nframes;iframes++)
    {

        /* calculate Potential Energy (-mgh) */

        seg_PE[isegs][iframes]=(-1.0)*seg_mass[isegs]*G*
            comy[isegs][iframes];

if (flag_2d)
{

    /* calculate Linear Kinetic Energy (0.5mv^2) */

    seg_LKE[isegs][iframes]=0.5*seg_mass[isegs]*
        (comxvel[isegs][iframes]*comxvel[isegs][iframes]+
        comyvel[isegs][iframes]*comyvel[isegs][iframes]);

    /* calculate Rotational Kinetic Energy (0.5Iomega^2) */

    seg_RKE[isegs][iframes]=0.5*seg_moi[isegs]*
        zavel[isegs][iframes]*zavel[isegs][iframes];
}
else
{

    /* calculate Linear Kinetic Energy (0.5mv^2) */

                seg_LKE[isegs][iframes]=0.5*seg_mass[isegs]*
                    (comxvel[isegs][iframes]*comxvel[isegs][iframes]+
                    comyvel[isegs][iframes]*comyvel[isegs][iframes]+
                    comzvel[isegs][iframes]*comzvel[isegs][iframes]);

                /* calculate Rotational Kinetic Energy (0.5Iomega^2) */

                seg_RKE[isegs][iframes]=0.5*seg_moi[isegs]*
                    (xavel[isegs][iframes]*xavel[isegs][iframes]+
                    yavel[isegs][iframes]*yavel[isegs][iframes]+
                    zavel[isegs][iframes]*zavel[isegs][iframes]);
            }
        }
    }
}
```

energy_plot()

```
#include "params.h"

void energy_plot(segs,nsegs,seg_PE,seg_LKE,seg_RKE,startframe,endframe,times)

/* this routine plots the selected segment energetics */
/* depending on the data arrays passed over to the routine */

char segs[][STRING_SIZE];              /* segment names */
int nsegs;                              /* number of segments */
float seg_PE[MAX_SEGS][MAX_FRAMES];    /* segment potential energy */
```

```
float seg_LKE[MAX_SEGS][MAX_FRAMES];    /* segment linear kinetic energy */
float seg_RKE[MAX_SEGS][MAX_FRAMES];    /* segment rotational kinetic energy */
int startframe;                        /* start frame number */
int endframe;                           /* end frame number */
float times[MAX_FRAMES];                /* times (s) */

{
    float pl_times[MAX_LINES][MAX_POINTS];    /* times to be plotted (s) */
    float energy[MAX_LINES][MAX_POINTS];    /* calculated energies */
    char key[MAX_LINES+1][STRING_SIZE];        /* graph key string */
    char title[STRING_SIZE];                  /* graph title */
    char x_label[STRING_SIZE];               /* graph x axis label */
    char y_label[STRING_SIZE];               /* graph y axis label */
    char fname[STRING_SIZE];                  /* picture filename */
    char filename[STRING_SIZE];              /* full picture filename */
    int iframe;                              /* frame counter */
    int inode;                              /* node counter */
    int ienergy;                              /* energy control */
    int iplot;                              /* plotter control */
    int nlines;                              /* number of lines on graph */
    int aseg;                                 /* finish node number */

    static char menu1[][STRING_SIZE]=        /* energy menu */
    {
        "Potential Energy",
        "Linear Kinetic Energy",
        "Rotational Kinetic Energy"
    };

    static char menu2[][STRING_SIZE]=        /* plotting menu */
    {
        "Save data to file",
        "Save picture to file",
        "Exit"
    };

    /* all segments */

    if (nsegs<MAX_LINES && yesno("Plot all segments?")==TRUE)
    {

        while ((ienergy=menu("Option:",menu1,3))==0);

        strcpy(key[nsegs],menu1[ienergy-1]);
        strcat(key[nsegs]," ");

        for (nlines=0;nlines<nsegs;nlines++)
        {
            strcpy(key[nlines],key[nsegs]);
            strcat(key[nlines],segs[nlines]);

            /* loop round frames */

            for (iframe=startframe;iframe<endframe;iframe++)
            {

                /* calculate times */

                pl_times[nlines][iframe-startframe]=times[iframe];

                switch (ienergy)
                {
```

```
                case 1:
                    energy[nlines][iframe-startframe]=
                        seg_PE[nlines][iframe];
                    break;

                case 2:
                    energy[nlines][iframe-startframe]=
                        seg_LKE[nlines][iframe];
                    break;

                case 3:
                    energy[nlines][iframe-startframe]=
                        seg_RKE[nlines][iframe];
                    break;
                }
            }
        }
    }
    else
    {

        /* loop round number of lines */

        nlines=0;
        do
        {

            /* first option */

            while ((ienergy=menu("Option:",menu1,3))==0);

            strcpy(key[nlines],menu1[ienergy-1]);
            strcat(key[nlines]," ");

            /* second option */

            while ((aseg=menu("Select segment:",segs,nsegs))==0);
            aseg--;
            strcat(key[nlines],segs[aseg]);

            /* loop round frames */

            for (iframe=startframe;iframe<endframe;iframe++)
            {

                /* calculate times */

                pl_times[nlines][iframe-startframe]=times[iframe];

                switch (ienergy)
                {
                case 1:
                    energy[nlines][iframe-startframe]=seg_PE[aseg][iframe];
                    break;

                case 2:
                    energy[nlines][iframe-startframe]=seg_LKE[aseg][iframe];
                    break;

                case 3:
                    energy[nlines][iframe-startframe]=seg_RKE[aseg][iframe];
                    break;
```

```
            }
        }
        nlines++;
    } while (nlines<MAX_LINES && yesno("Another line?")==TRUE);
}

/* draw graph */

strcpy(title,"Energetics");
strcpy(x_label,"Time (s)");
strcpy(y_label,"Energy (J)");
d_graph(display,title,x_label,y_label,pl_times,energy,endframe -
    startframe,nlines,key,FALSE);
while ((iplot=menu("Select option:",menu2,3))!=3)
{
    switch (iplot)
    {
    case 1:
        save_an(title,x_label,y_label,pl_times,energy,endframe-startframe,
            nlines,key);
        break;
    case 2:
        printf("Input picture file name : ");
        scanf("%s",fname);
        strcpy(filename,PICTURE_DIRECTORY);
        strcat(filename,PICTURE_PREFIX);
        strcat(filename,fname);
        strcat(filename,PICTURE_SUFFIX);

        bitmap_to_file(display,TRUE,0,0,filename,TRUE,0.0,0.0,0,0,TRUE);
        break;
    }
    CLEAR_GRAPH;

    /* finished */

}
```

filter()

```
#include "params.h"

void filter(nnodes,xpos,ypos,zpos,xpos_filt,ypos_filt,zpos_filt,nframe,fspeed)

/* this routine performs a forth order, zero phase Butterworth digital filtration on the
*/
/* node position data */

/* the second pass is reversed for zero overall phase shift */

/* the cutoff frequency is set to 1/5 of the sampling frequency */

int nnodes;                         /* Number of nodes */
float xpos[MAX_NODES][MAX_FRAMES];        /* x world coordinates (m) */
float ypos[MAX_NODES][MAX_FRAMES];        /* y world coordinates (m) */
float zpos[MAX_NODES][MAX_FRAMES];        /* z world coordinates (m) */
float xpos_filt[MAX_NODES][MAX_FRAMES];        /* filtered x world coordinates (m) */
float ypos_filt[MAX_NODES][MAX_FRAMES];        /* filtered y world coordinates (m) */
float zpos_filt[MAX_NODES][MAX_FRAMES];        /* filtered z world coordinates (m) */
int nframe;                         /* number of frames */
```

```
float fspeed;                       /* Film frame interval (s) */

{
    float xfilter[MAX_NODES][MAX_FRAMES];    /* temporary filtered x data store */
    float yfilter[MAX_NODES][MAX_FRAMES];    /* temporary filtered y data store */
    float zfilter[MAX_NODES][MAX_FRAMES];    /* temporary filtered z data store */
    int inodes;                 /* node counter */
    int iframe;                 /* frame counter */
    static float data_block[11][5]=     /* filtration coefficients */
    {
        .2929, .5858,.2929, 0.0000,-.1716,
        .2066, .4132,.2066, 0.3695,-.1959,
        .15505,.3101,.15505,0.6202,-.2404,
        .1212, .2424,.1212, 0.8030,-.2878,
        .0884, .1768,.0884, 1.0011,-.3547,
        .06745,.1349,.06745,1.1430,-.4128,
        .0495, .0990,.0495, 1.2796,-.4776,
        .0379, .0758,.0379, 1.3789,-.5305,
        .02995,.0599,.02995,1.4542,-.5740,
        .0243, .0486,.0243, 1.5134,-.6106,
        .0201, .0402,.0201, 1.5610,-.6414
    };

    /* filtration coefficients (Winter 1979 "Biomechanics of Human Movement") */

    float a0,a1,a2,b1,b2;

    a0=data_block[filtration_number-1][0];
    a1=data_block[filtration_number-1][1];
    a2=data_block[filtration_number-1][2];
    b1=data_block[filtration_number-1][3];
    b2=data_block[filtration_number-1][4];

    /* test for suitable number of frames */

    if (nframe<5)
    {
        printf("Too few frames for filtration\n");
        for (inodes=0;inodes<nnodes;inodes++)
        {
            for (iframe=0;iframe<nframe;iframe++)
            {
                xpos_filt[inodes][iframe]=xpos[inodes][iframe];
                ypos_filt[inodes][iframe]=ypos[inodes][iframe];
                zpos_filt[inodes][iframe]=zpos[inodes][iframe];
            }
        }
        return;
    }

    /* loop over nodes */

    for (inodes=0;inodes<nnodes;inodes++)
    {

        /* first pass */

        xfilter[inodes][0]=xpos[inodes][0];
        yfilter[inodes][0]=ypos[inodes][0];
        zfilter[inodes][0]=zpos[inodes][0];
        xfilter[inodes][1]=xpos[inodes][1];
        yfilter[inodes][1]=ypos[inodes][1];
```

```
        zfilter[inodes][1]=zpos[inodes][1];
        for (iframe=2;iframe<nframe;iframe++)
        {
            xfilter[inodes][iframe]=a0*xpos[inodes][iframe]+
                a1*xpos[inodes][iframe-1]+
                a2*xpos[inodes][iframe-2]+b1*xfilter[inodes][iframe-1]+
                b2*xfilter[inodes][iframe-2];

            yfilter[inodes][iframe]=a0*ypos[inodes][iframe]+
                a1*ypos[inodes][iframe-1]+
                a2*ypos[inodes][iframe-2]+b1*yfilter[inodes][iframe-1]+
                b2*yfilter[inodes][iframe-2];

            zfilter[inodes][iframe]=a0*zpos[inodes][iframe]+
                a1*zpos[inodes][iframe-1]+
                a2*zpos[inodes][iframe-2]+b1*zfilter[inodes][iframe-1]+
                b2*zfilter[inodes][iframe-2];
        }

        /* second pass */
#if 1
        /* reverse version for zero phase shift */

        xpos_filt[inodes][nframe-1]=xfilter[inodes][nframe-1];
        ypos_filt[inodes][nframe-1]=yfilter[inodes][nframe-1];
        zpos_filt[inodes][nframe-1]=zfilter[inodes][nframe-1];
        xpos_filt[inodes][nframe-2]=xfilter[inodes][nframe-2];
        ypos_filt[inodes][nframe-2]=yfilter[inodes][nframe-2];
        zpos_filt[inodes][nframe-2]=zfilter[inodes][nframe-2];
        for (iframe=nframe-3;iframe>=0;iframe--)
        {
            xpos_filt[inodes][iframe]=a0*xfilter[inodes][iframe]+
                a1*xfilter[inodes][iframe+1]+a2*xfilter[inodes][iframe+2]+
                b1*xpos_filt[inodes][iframe+1]+b2*xpos_filt[inodes][iframe+2];

            ypos_filt[inodes][iframe]=a0*yfilter[inodes][iframe]+
                a1*yfilter[inodes][iframe+1]+a2*yfilter[inodes][iframe+2]+
                b1*ypos_filt[inodes][iframe+1]+b2*ypos_filt[inodes][iframe+2];

            zpos_filt[inodes][iframe]=a0*zfilter[inodes][iframe]+
                a1*zfilter[inodes][iframe+1]+a2*zfilter[inodes][iframe+2]+
                b1*zpos_filt[inodes][iframe+1]+b2*zpos_filt[inodes][iframe+2];
        }
#else
        /* unreversed for improved last points smoothing */

        xpos_filt[inodes][0]=xfilter[inodes][0];
        ypos_filt[inodes][0]=yfilter[inodes][0];
        zpos_filt[inodes][0]=zfilter[inodes][0];
        xpos_filt[inodes][1]=xfilter[inodes][1];
        ypos_filt[inodes][1]=yfilter[inodes][1];
        zpos_filt[inodes][1]=zfilter[inodes][1];
        for (iframe=2;iframe<nframe;iframe++)
        {
            xpos_filt[inodes][iframe]=a0*xfilter[inodes][iframe]+
                a1*xfilter[inodes][iframe-1]+a2*xfilter[inodes][iframe-2]+
                b1*xpos_filt[inodes][iframe-1]+b2*xpos_filt[inodes][iframe-2];

            ypos_filt[inodes][iframe]=a0*yfilter[inodes][iframe]+
                a1*yfilter[inodes][iframe-1]+a2*yfilter[inodes][iframe-2]+
                b1*ypos_filt[inodes][iframe-1]+b2*ypos_filt[inodes][iframe-2];
```

```
            zpos_filt[inodes][iframe]=a0*zfilter[inodes][iframe]+
                a1*zfilter[inodes][iframe-1]+a2*zfilter[inodes][iframe-2]+
                b1*zpos_filt[inodes][iframe-1]+b2*zpos_filt[inodes][iframe-2];
        }
#endif
    }

    /* finished */

    printf("Filtration successful\n");

}
```

frame_range()

```
#include "params.h"

/* this routine uses the knobs to move the repeat frame range */

frame_range(fd,min_frame,max_frame)
int fd;                                 /* display device */
int *min_frame;                         /* minimum frame number */
int *max_frame;                         /* maximum frame number */

{
    int valid;                        /* valid response flag */
    float rdum;                        /* dummy value */
    float rmin;                        /* minimum range knob */
    float rmax;                        /* maximum range knob */
    int string[STRING_SIZE];           /* text output string */

    sample_locator(knobs1,1,&valid,&rdum,&rmin,&rmax);
    *min_frame=(int)rmin;
    *max_frame=(int)rmax;

    sprintf(string,"Frames %3d to %3d",*min_frame,(*max_frame)-1);
    dctext(fd,SIDE_BORDER+10,TOP_BORDER+80,string);

}
```

gap()

```
/* This module contains the main routine for the 'G'ait 'A'nalysis 'P'rogram (GAP) */

#include "params.h"

/* global variables */

/* options flags */

int flag_2d=TRUE;                          /* 2d only flag */
int flag_simple_reconstruction=FALSE;     /* simple (orthogonal camera) 3d reconstruction
flag */
int fiducial_flag=FALSE;                  /* fiducial mark recostruction flag */
int frame_increment=1;                   /* frame sequence increment */
int filtration_number=2;                  /* filtration cutoff number */
int flag_filter=FALSE;                   /* filter/smooth flag */
int smooth_number=3;                      /* number of values in moving average */

/* 3d reconstruction values */
```

```
float l1[11];                   /* DLT parameters */
float l2[11];

int x_mirror_1;                   /* x mirroring flag */
float x_offset_1;                   /* x offset of picture origin 1 */
float y_offset_1;                   /* y offset of picture origin 1 */
float scale_factor_1;             /* scale factor from picture to world (m) 1 */
float rotation_1[2][2];             /* rotation matrix picture to world 1 */
int x_mirror_2;                   /* x mirroring flag */
float x_offset_2;               /         * x offset of picture origin 2 */
float y_offset_2;                  /* y offset of picture origin 2 */
float scale_factor_2;             /* scale factor from picture to world (m) 2 */
float rotation_2[2][2];             /* rotation matrix picture to world 2 */
int x_axis_source;                /* 3d axes source: 1 picture 1 x axis */
int y_axis_source;                /*                 2 picture 1 y axis */
int z_axis_source;                /*                 3 picture 2 x axis */
/*                                                    4 picture 2 y axis */
/* 2d conversion values */

float x_offset;                 /* x offset of picture origin */
float y_offset;                 /* y offset of picture origin */
float scale_factor;             /* scale factor from picture to world (m) */
float rotation[2][2];           /* rotation matrix picture to world */

/* registration values */

float fiducial_x=0.0;             /* global x registration value */
float fiducial_y=0.0;             /* global y registration value */
float correction_x=0.0;             /* framewide x registration */
float correction_y=0.0;             /* framewide y registration */

/* graphics device pointers */

int display;                        /* graphics display */
int locator;                        /* locator device (mouse) */
int knobs1,knobs2,knobs3;         /* knob box (3 rows) */
int bbox;                           /* button box */

/* window device pointers */

Window graphic_window;            /* graphic window */
Window text_window;                /* text window */
Window menu_window;                /* menu window */
Display *xdisplay;                /* x display pointer */
int xscreen;                        /* screen number */

/* original hpterm window parameters */

unsigned int orig_width,orig_height;
int orig_x,orig_y;

/* global brightness and contrast values */

float brightness=0.5;             /* brightness */
float contrast=0.5;                 /* contrast */

void gap()
{

    /* This program records frames of 3d gait analysis data obtained from */
    /* split image video film */
```

```
int limb_data=FALSE;              /* Read limb file flag */
int data_to_write=FALSE;          /* New data flag */
int nframe=0;                      /* Frame number */
int iret;                          /* Menu return code */
int nnodes;                       /* Number of nodes */
int nsegs;                       /* Number of segments */
int iopt;                          /* y world coordinate option */

float fspeed;                      /* Film frame interval (s) */

char title[STRING_SIZE];         /* File title line */

int nodes_per_seg[NPS][MAX_SEGS];    /* Nodes in segment */
float seg_radii[NPS][MAX_SEGS];    /* relative radii of segment ends */

float xpos[MAX_NODES][MAX_FRAMES];    /* x world coordinates (m) */
float ypos[MAX_NODES][MAX_FRAMES];    /* y world coordinates (m) */
float zpos[MAX_NODES][MAX_FRAMES];    /* z world coordinates (m) */

float xpos_filt[MAX_NODES][MAX_FRAMES];/* filtered x world coordinates (m) */
float ypos_filt[MAX_NODES][MAX_FRAMES];/* filtered y world coordinates (m) */
float zpos_filt[MAX_NODES][MAX_FRAMES];/* filtered z world coordinates (m) */

float seg_mass[MAX_SEGS];         /* array of segment masses */
float seg_com[MAX_SEGS];             /* array of segment relative COMs */
float seg_moi[MAX_SEGS];             /* array of segment MOIs */

char nodes[MAX_NODES][STRING_SIZE];    /* Names of nodes */
char segs[MAX_SEGS][STRING_SIZE];    /* Names of segments */

static char menu1[][STRING_SIZE]=    /* Main menu */
{
    "Read limb file",
    "Read node file",
    "Write node file",
    "Digitize new sequence",
    "Digitize additional frames",
    "Display frames",
    "Analyse gait",
    "View video frames",
    "Set global options",
    "Shell to Unix",
    "Exit",
};

/* open devices */

open_dev();

CLEAR_TEXT;
printf("Gait Analysis Program\n\n");

while (TRUE)
{

    /* write main menu */

    iret=menu("Gait analysis program",menu1,11);

    /* select options */
```

```
        if (iret==1)
        {

            /* test to see if unsaved data exists */

            if (data_to_write==TRUE)
            {
                if (yesno("Save current frame data ?"))
                    wrnode(title,nodes,nnodes,xpos,ypos,zpos,nframe,fspeed);
                data_to_write=FALSE;
            }

            /* read the limb model file */

            rdlimb(title,nodes,&nnodes,segs,&nsegs,nodes_per_seg,seg_mass,
                seg_com,seg_moi);
            nframe=0;
            data_to_write=FALSE;
            limb_data=TRUE;
        }

        if (iret==2 && limb_data==TRUE)
        {

            /* test to see if unsaved data exists */

            if (data_to_write==TRUE)
            {
                if (yesno("Save current frame data ?"))
                    wrnode(title,nodes,nnodes,xpos,ypos,zpos,
                    nframe,fspeed);
                data_to_write=FALSE;
            }

            /* read a node data file */

            rdnode(title,nodes,&nnodes,xpos,ypos,zpos,&nframe,&fspeed);

            /* filter data */

            if (flag_filter) filter(nnodes,xpos,ypos,zpos,xpos_filt,
                ypos_filt,zpos_filt,nframe,fspeed);
            else smooth(nnodes,xpos,ypos,zpos,xpos_filt,
                ypos_filt,zpos_filt,nframe,fspeed);

        }

        if (iret==3 && nframe!=0)
        {

            /* save frame data file */

            wrnode(title,nodes,nnodes,xpos,ypos,zpos,nframe,fspeed);
            data_to_write=FALSE;
        }

        if (iret==4 && limb_data==TRUE)
        {
            /* test to see if unsaved data exists */

            if (data_to_write==TRUE)
            {
```

```
        if (yesno("Save current frame data ?"))
            wrnode(title,nodes,nnodes,xpos,ypos,zpos,nframe,fspeed);
        data_to_write=FALSE;
    }

    /* initialize the digitizer */

    initrd();

    /* get frame speed */

    printf("Input interval between frames (s) : ");
    scanf("%f",&fspeed);

    /* digitize frames required */

    nframe=0;
    dgnode(title,nodes,nnodes,xpos,ypos,zpos,&nframe);
    data_to_write=TRUE;

    /* filter data */

    if (flag_filter) filter(nnodes,xpos,ypos,zpos,xpos_filt,
        ypos_filt,zpos_filt,nframe,fspeed);
    else smooth(nnodes,xpos,ypos,zpos,xpos_filt,
        ypos_filt,zpos_filt,nframe,fspeed);

}

if (iret==5 && limb_data==TRUE && nframe!=0)
{

    /* initialize the digitizer */

    if (data_to_write==FALSE)
    {
        initrd();

        /* get frame speed */

        printf("Input interval between frames (s) : ");
        scanf("%f",&fspeed);

    }
    else
    {
        if (yesno("Reinitialize digitizing ?"))
        {
            initrd();

            /* get frame speed */

            printf("Input interval between frames (s) : ");
            scanf("%f",&fspeed);

        }
    }

    /* digitize frames required */

    dgnode(title,nodes,nnodes,xpos,ypos,zpos,&nframe);
    data_to_write=TRUE;
```

```
        /* filter data */

        if (flag_filter) filter(nnodes,xpos,ypos,zpos,xpos_filt,
            ypos_filt,zpos_filt,nframe,fspeed);
        else smooth(nnodes,xpos,ypos,zpos,xpos_filt,
            ypos_filt,zpos_filt,nframe,fspeed);

    }

    if (iret==6 && nframe!=0)

        /* display digitized data */

        displa(title,nodes,nnodes,xpos,ypos,zpos,xpos_filt,ypos_filt,zpos_filt,
            nframe,segs,nsegs,nodes_per_seg,seg_radii);

    if (iret==7 && nframe!=0)

        /* perform data analysis */

        stats(title,nodes,nnodes,xpos,ypos,zpos,xpos_filt,ypos_filt,
            zpos_filt, nframe,segs,nsegs,nodes_per_seg,fspeed,seg_mass,
            seg_com,seg_moi);

    if (iret==8)

        /* view video frames */

        view();

    if (iret==9)
    {

        /* set global options */

        options();

        /* re-filter data */

        if (flag_filter) filter(nnodes,xpos,ypos,zpos,xpos_filt,
            ypos_filt,zpos_filt,nframe,fspeed);
        else smooth(nnodes,xpos,ypos,zpos,xpos_filt,
            ypos_filt,zpos_filt,nframe,fspeed);
    }

    if (iret==10)
    {

        /* shell out to Unix */

        system(SHELL);
    }

    if (iret==11)
    {
        if (yesno("Are you sure you want to quit?"))
        {

            /* test to see if unsaved data exists */

            if (data_to_write==TRUE)
```

```
            {
                if (yesno("Save current frame data ?"))
                    wrnode(title,nodes,nnodes,xpos,ypos,zpos,
                        nframe,fspeed);
            }

            /* terminate program */

            break;
        }
    }

    /* loop back to menu */

    }

    /* close devices */

    close_dev();

    /* finished */

}
```

identity3d()

```
/* Subroutine identity3d produces the identity matrix */

#include "params.h"


void identity3d(m)
float m[4][4];
{
    int i,j;

    for (i=0;i<4;i++)
        for (j=0;j<4;j++)
            m[i][j]=(float)(i==j);
}
```

initrd()

```
/* routine to select initialize options depending on flag_2d */

#include "params.h"

void initrd()
{
    if (flag_2d==TRUE) initrd_2d();
    else
    {
        if (flag_simple_reconstruction==TRUE) initrd_3d2();
        else initrd_3d1();
    }
}
```

initrd_2d()

```
#include "params.h"
```

```
void initrd_2d()

/* this routine sets up the transformation values for the 2d input */

{
    double theta;                    /* rotation angle */
    float x_pos,y_pos;              /* screen position of reference point */
    float x_dis,y_dis;              /* screen displacement of reference point */
    float x_ref,y_ref;              /* reference point */
    FILE *unit;                      /* file pointer */
    char filename[STRING_SIZE];      /* file name */
    char fname[STRING_SIZE];         /* input file name */
    int iret;                        /* menu option */
static char menu1[][STRING_SIZE]=    /* initialization option menu */
{
    "Load reconstruction parameters",
    "Calculate reconstruction parameters"
};

/* do menu */

while ((iret=menu("Select option:",menu1,2))==0);

switch (iret)
{
case 1:

    /* load up parameters file */

    do
    {
        sel_file(RECON_DIRECTORY,RECON_PREFIX,RECON_SUFFIX_2D,filename);
        unit=fopen(filename,"r");
    } while (unit==NULL);

    fscanf(unit,"%e%e",&fiducial_x,&fiducial_y);

    fscanf(unit,"%e",&x_offset);
    fscanf(unit,"%e",&y_offset);
    fscanf(unit,"%e",&scale_factor);
    fscanf(unit,"%e%e",&rotation[0][0],&rotation[0][1]);
    fscanf(unit,"%e%e",&rotation[1][0],&rotation[1][1]);

    fclose(unit);
    printf("File %s read successfully\n",filename);
    break;

case 2:

    /* get into right graphics mode */

    CLEAR_GRAPH;

    /* read in picture file */

    readpic();

    /* fiducial point */

    if (fiducial_flag)
    {
```

```
        printf("Select fiducial point\n");
        digrd(&fiducial_x,&fiducial_y);
    }

    /* get origin */

    printf("Select origin from picture\n");
    digrd(&x_offset,&y_offset);

    /* get reference point */

    printf("Input coordinates of a reference point : ");
    scanf("%f%f",&x_ref,&y_ref);
    printf("Select reference point from picture\n");
    digrd(&x_pos,&y_pos);

    /* calculate scale factor */

    x_dis=x_pos-x_offset;
    y_dis=y_pos-y_offset;
    scale_factor=sqrt((double)(x_ref*x_ref+y_ref*y_ref))/
        sqrt((double)(x_dis*x_dis+y_dis*y_dis));

    /* calculate rotation */

    theta=atan2((double)y_ref,(double)x_ref)-
        atan2((double)y_dis,(double)x_dis);

    /* and rotation array */

    rotation[0][0]=(float)cos(theta);
    rotation[0][1]=(-(float)sin(theta));
    rotation[1][0]=(float)sin(theta);
    rotation[1][1]=(float)cos(theta);

    /* terminate graphics */

    CLEAR_GRAPH;

    /* write out reconstruction parameters file */

    if (yesno("Write parameters file?"))
    {
        do
        {
            printf("Input reconstruction parameters file name : ");
            scanf("%s",fname);
            strcpy(filename,RECON_DIRECTORY);
            strcat(filename,RECON_PREFIX);
            strcat(filename,fname);
            strcat(filename,RECON_SUFFIX_2D);

        unit=fopen(filename,"w");
    } while (unit==NULL);

    fprintf(unit,"%e %e\n",fiducial_x,fiducial_y);

    fprintf(unit,"%e\n",x_offset);
    fprintf(unit,"%e\n",y_offset);
    fprintf(unit,"%e\n",scale_factor);
    fprintf(unit,"%e %e\n",rotation[0][0],rotation[0][1]);
    fprintf(unit,"%e %e\n",rotation[1][0],rotation[1][1]);
```

```
            fclose(unit);
            printf("File %s written successfully\n",filename);
        }
        break;
    }
}
```

Initrd_3d1()

```
#include "params.h"

void initrd_3d1()

/* this routine sets up the transformation values for the 2d to 3d conversion */
/* it uses the Marzan algorithms to enable the cameras to be in any convenient position
*/

{

    FILE *unit;                         /* file pointer */
    char filename[STRING_SIZE];         /* file name */
    char fname[STRING_SIZE];             /* input file name */
    float xw[MAX_REF];                 /* reference point world coordinates */
    float yw[MAX_REF];
    float zw[MAX_REF];
    float q1[MAX_REF];                 /* reference points screen coordinates */
    float r1[MAX_REF];
    float q2[MAX_REF];                 /* reference points screen coordinates */
    float r2[MAX_REF];
    int iret;                             /* menu option */
    int nrefs;                         /* reference value number */
    int irefs;                         /* reference value counter */
    static char menu1[][STRING_SIZE]=    /* initialization option menu */
    {
        "Load reconstruction parameters",
        "Calculate reconstruction parameters"
    };
    int i;                                 /* load up counters */

    /* do menu */

    while ((iret=menu("Select option:",menu1,2))==0);

    switch (iret)
    {
    case 1:

        /* load up parameters file. */

        do
        {
            sel_file(RECON_DIRECTORY,RECON_PREFIX,RECON_SUFFIX_3D1,filename);
            unit=fopen(filename,"r");
        } while (unit==NULL);

        for (i=0;i<11;i++) fscanf(unit,"%e",&l1[i]);
        for (i=0;i<11;i++) fscanf(unit,"%e",&l2[i]);
        fscanf(unit,"%e%e",&fiducial_x,&fiducial_y);

        fclose(unit);
```

```
        printf("File %s read successfully\n",filename);
        break;

    case 2:

        /* get into right graphics mode */

        CLEAR_GRAPH;

        /* read in picture file */

        readpic();

        /* fiducial point */

        if (fiducial_flag)
        {
            printf("Select fiducial point\n");
            digrd(&fiducial_x,&fiducial_y);
        }
        else
        {
            fiducial_x=0;
            fiducial_y=0;
        }

        /* get world coordinates of reference points */

        do
        {
            printf("Input number of reference points : ");
            scanf("%d",&nrefs);
        }while (nrefs<MIN_REF || nrefs>=MAX_REF);

        for (irefs=0;irefs<nrefs;irefs++)
        {
            printf("Reference point %d :",irefs+1);
            scanf("%f %f %f",&xw[irefs],&yw[irefs],&zw[irefs]);
        }

        /* select the reference points */

        for (irefs=0;irefs<nrefs;irefs++)
        {
            printf("Picture 1, reference point %d\n",irefs+1);
            digrd(&q1[irefs],&r1[irefs]);
            printf("Picture 2, reference point %d\n",irefs+1);
            digrd(&q2[irefs],&r2[irefs]);
        }

        /* calculate dlt parameters */

        dlt_parameters(xw,yw,zw,q1,r1,&nrefs,l1);
        dlt_parameters(xw,yw,zw,q2,r2,&nrefs,l2);

        /* terminate graphics */

        CLEAR_GRAPH;

        /* write out reconstruction parameters file */

        printf("Input reconstruction parameters file name : ");
```

```
        scanf("%s",fname);
        strcpy(filename,RECON_DIRECTORY);
        strcat(filename,RECON_PREFIX);
        strcat(filename,fname);
        strcat(filename,RECON_SUFFIX_3D1);

        unit=fopen(filename,"w");
        for (i=0;i<11;i++) fprintf(unit,"%e\n",l1[i]);
        fprintf(unit,"\n");
        for (i=0;i<11;i++) fprintf(unit,"%e\n",l2[i]);
        fprintf(unit,"\n%e %e\n",fiducial_x,fiducial_y);
        fclose(unit);
        printf("File %s written successfully\n",filename);
        break;
    }
}
```

initrd_3d2()

```
#include "params.h"

void initrd_3d2()

/* this routine sets up the transformation values for the simple 3d reconstruction */

{
    double theta;                   /* rotation angle */
    float x_pos,y_pos;             /* screen position of reference point */
    float x_dis,y_dis;             /* screen displacement of reference point */
    float x_ref,y_ref;              /* reference point */
    FILE *unit;                    /* file pointer */
    char filename[STRING_SIZE];    /* file name */
    char fname[STRING_SIZE];        /* input file name */
    int iret;                       /* menu option */
    static char menu1[][STRING_SIZE]=    /* initialization option menu */
    {
        "Load reconstruction parameters",
        "Calculate reconstruction parameters"
    };
    char menu2[4][STRING_SIZE];    /* 3d information source menu */

    /* do menu */

    while ((iret=menu("Select option:",menu1,2))==0);

    switch (iret)
    {
    case 1:

        /* load up parameters file */

        do
        {
            sel_file(RECON_DIRECTORY,RECON_PREFIX,RECON_SUFFIX_3D2,filename);
            unit=fopen(filename,"r");
        } while (unit==NULL);

        fscanf(unit,"%e%e",&fiducial_x,&fiducial_y);

        fscanf(unit,"%e",&x_offset_1);
        fscanf(unit,"%e",&y_offset_1);
```

```
        fscanf(unit,"%e",&scale_factor_1);
        fscanf(unit,"%e%e",&rotation_1[0][0],&rotation_1[0][1]);
        fscanf(unit,"%e%e",&rotation_1[1][0],&rotation_1[1][1]);
        fscanf(unit,"%d",&x_mirror_1);

        fscanf(unit,"%e",&x_offset_2);
        fscanf(unit,"%e",&y_offset_2);
        fscanf(unit,"%e",&scale_factor_2);
        fscanf(unit,"%e%e",&rotation_2[0][0],&rotation_2[0][1]);
        fscanf(unit,"%e%e",&rotation_2[1][0],&rotation_2[1][1]);
        fscanf(unit,"%d",&x_mirror_2);

        fscanf(unit,"%d",&x_axis_source);
        fscanf(unit,"%d",&y_axis_source);
        fscanf(unit,"%d",&z_axis_source);

        fclose(unit);
        printf("File %s read successfully\n",filename);
        break;

    case 2:

        /* get into right graphics mode */

        CLEAR_GRAPH;

        /* read in picture file */

        readpic();

        /* fiducial point */

        if (fiducial_flag)
        {
            printf("Select fiducial point\n");
            digrd(&fiducial_x,&fiducial_y);
        }

        /* PICTURE 1 */

        /* get x mirror status */

        printf("PICTURE 1\n");
        x_mirror_1=yesno("Mirror x axis on picture 1");

        /* get origin */

        printf("Select origin\n");
        digrd(&x_offset_1,&y_offset_1);
        if (x_mirror_1) x_offset_1=(-x_offset_1);

        /* get reference point */

        printf("Input coordinates of a reference point : ");
        scanf("%f%f",&x_ref,&y_ref);
        printf("Select reference point from picture\n");
        digrd(&x_pos,&y_pos);
        if (x_mirror_1) x_pos=(-x_pos);

        /* calculate scale factor */

        x_dis=x_pos-x_offset_1;
```

```
y_dis=y_pos-y_offset_1;
scale_factor_1=sqrt((double)(x_ref*x_ref+y_ref*y_ref))/
    sqrt((double)(x_dis*x_dis+y_dis*y_dis));

/* calculate rotation */

theta=atan2((double)y_ref,(double)x_ref)-
    atan2((double)y_dis,(double)x_dis);

/* and rotation array */

rotation_1[0][0]=(float)cos(theta);
rotation_1[0][1]=(-(float)sin(theta));
rotation_1[1][0]=(float)sin(theta);
rotation_1[1][1]=(float)cos(theta);

/* PICTURE 2 */

/* get x mirror status */

printf("PICTURE 2\n");
x_mirror_2=yesno("Mirror x axis on picture 2");

/* get origin */

printf("Select origin\n");
digrd(&x_offset_2,&y_offset_2);
if (x_mirror_2) x_offset_2=(-x_offset_2);

/* get reference point */

printf("Input coordinates of a reference point : ");
scanf("%f%f",&x_ref,&y_ref);
printf("Select reference point from picture\n");
digrd(&x_pos,&y_pos);
if (x_mirror_2) x_pos=(-x_pos);

/* calculate scale factor */

x_dis=x_pos-x_offset_2;
y_dis=y_pos-y_offset_2;
scale_factor_2=sqrt((double)(x_ref*x_ref+y_ref*y_ref))/
    sqrt((double)(x_dis*x_dis+y_dis*y_dis));

/* calculate rotation */

theta=atan2((double)y_ref,(double)x_ref)-
    atan2((double)y_dis,(double)x_dis);

/* and rotation array */

rotation_2[0][0]=(float)cos(theta);
rotation_2[0][1]=(-(float)sin(theta));
rotation_2[1][0]=(float)sin(theta);
rotation_2[1][1]=(float)cos(theta);

/* terminate graphics */

CLEAR_GRAPH;

/* select sources of 3d information */
```

```
strcpy(menu2[0],"Picture 1 x axis");
strcpy(menu2[1],"          y axis");
strcpy(menu2[2],"Picture 2 x axis");
strcpy(menu2[3],"          y axis");

do
{
    x_axis_source=menu("3d x axis data from:",menu2,4);
}       while (x_axis_source==0);

strcat(menu2[x_axis_source-1]," x");

do
{
    y_axis_source=menu("3d y axis data from:",menu2,4);
}       while (y_axis_source==0 || y_axis_source==x_axis_source);

strcat(menu2[y_axis_source-1]," y");

do
{
    z_axis_source=menu("3d z axis data from:",menu2,4);
}       while (z_axis_source==0 || z_axis_source==x_axis_source ||
    z_axis_source==y_axis_source);

/* write out reconstruction parameters file */

if (yesno("Write parameters file?"))
{
    do
    {
        printf("Input reconstruction parameters file name : ");
        scanf("%s",fname);
        strcpy(filename,RECON_DIRECTORY);
        strcat(filename,RECON_PREFIX);
        strcat(filename,fname);
        strcat(filename,RECON_SUFFIX_3D2);

        unit=fopen(filename,"w");
    } while (unit==NULL);

    fprintf(unit,"%e %e\n",fiducial_x,fiducial_y);

    fprintf(unit,"%e\n",x_offset_1);
    fprintf(unit,"%e\n",y_offset_1);
    fprintf(unit,"%e\n",scale_factor_1);
    fprintf(unit,"%e %e\n",rotation_1[0][0],rotation_1[0][1]);
    fprintf(unit,"%e %e\n",rotation_1[1][0],rotation_1[1][1]);
    fprintf(unit,"%d\n",x_mirror_1);

    fprintf(unit,"%e\n",x_offset_2);
    fprintf(unit,"%e\n",y_offset_2);
    fprintf(unit,"%e\n",scale_factor_2);
    fprintf(unit,"%e %e\n",rotation_2[0][0],rotation_2[0][1]);
    fprintf(unit,"%e %e\n",rotation_2[1][0],rotation_2[1][1]);
    fprintf(unit,"%d\n",x_mirror_2);

    fprintf(unit,"%d\n",x_axis_source);
    fprintf(unit,"%d\n",y_axis_source);
    fprintf(unit,"%d\n",z_axis_source);

    fclose(unit);
```

```
            printf("File %s written successfully\n",filename);
        }
        break;
    }
}
```

length()

```
#include "params.h"

void length(xpos,ypos,zpos,nnodes,nframes,nodes_per_seg,nsegs,seg_length)

/* calculate the mean lengths of the segments */

float xpos[MAX_NODES][MAX_FRAMES];    /* x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];    /* y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];    /* z world coordinates */
int nnodes;                           /* number of nodes */
int nframes;                          /* number of frames */
int nodes_per_seg[NPS][MAX_SEGS];    /* nodes per segment */
int nsegs;                            /* number of segments */
float seg_length[MAX_SEGS];           /* mean segment lengths */
{
    int iframes;                      /* frame counter */
    int isegs;                       /* segment counter */
    int nstart,nend;                  /* start and end nodes for a seg */
    float x,y,z;                      /* intermediate lengths */

    /* find mean segment lengths */

    for (isegs=0;isegs<nsegs;isegs++)
    {
        /* find start and end nodes */

        nstart=nodes_per_seg[0][isegs];
        nend=nodes_per_seg[1][isegs];

        /* loop over frames */

        seg_length[isegs]=0.0;
        for (iframes=0;iframes<nframes;iframes++)
        {
            x=xpos[nend][iframes]-xpos[nstart][iframes];
            y=ypos[nend][iframes]-ypos[nstart][iframes];
            z=zpos[nend][iframes]-zpos[nstart][iframes];

            seg_length[isegs]+=(float)sqrt((double)x*x+y*y+z*z);
        }
        seg_length[isegs]=seg_length[isegs]/(float)nframes;
    }
}
```

limb_com()

```
/* limb_com calculates the position of the centre of mass for the limb */

#include "params.h"

void limb_com(x1,y1,z1,x2,y2,z2,rel_com,x,y,z)
float x1,y1,z1;                       /* start of axis of conic section */
```

```
float x2,y2,z2;                          /* end of axis of conic section */
float rel_com;                          /* relative COM position */
float *x,*y,*z;                          /* calculated position of the centre of mass */
{
    float dx,dy,dz;                      /* displacement along axis */

    dx=x2-x1;
    dy=y2-y1;
    dz=z2-z1;

    *x=x1+rel_com*dx;
    *y=y1+rel_com*dy;
    *z=z1+rel_com*dz;
}
```

locus()

```
#include "params.h"

void locus(nodes,nnodes,xpos,ypos,zpos,nframe,fspeed)

/* this routine plots the selected node locus */

int nnodes;                              /* Number of nodes */
char nodes[][STRING_SIZE];              /* Names of nodes */
float xpos[][MAX_FRAMES];               /* x world coordinates (m) */
float ypos[][MAX_FRAMES];               /* y world coordinates (m) */
float zpos[][MAX_FRAMES];               /* z world coordinates (m) */
int nframe;                              /* number of frames */

{
    float x_locus[MAX_LINES][MAX_FRAMES];/* locus x axis */
    float y_locus[MAX_LINES][MAX_FRAMES];/* locus y axis */
    char key[MAX_LINES][STRING_SIZE];    /* graph key string */
    char title[STRING_SIZE];             /* graph title */
    char x_label[STRING_SIZE];          /* graph x axis label */
    char y_label[STRING_SIZE];          /* graph y axis label */
    char fname[STRING_SIZE];             /* picture filename */
    char filename[STRING_SIZE];         /* full picture filename */
    int iabs;                            /* absolute/relative indicator */
    int iframe;                         /* frame counter */
    int iplot;                          /* plotter control */
    int nlines;                         /* number of lines on graph */
    int anode;                          /* selected node */
    int bnode;                          /* reference node */
    int x_axis;                         /* x_axis selector */
    int y_axis;                         /* y_axis selector */
    float refx=0.0,refy=0.0,refz=0.0;    /* node reference values */

    static char menu1[][STRING_SIZE]=    /* locus x selector menu */
    {
        "x Value",
        "y Value",
        "z Value"
    };
    static char menu2[][STRING_SIZE]=    /* locus y selector menu */
    {
        "x Value",
        "y Value",
        "z Value"
    };
```

```
static char menu3[][STRING_SIZE]=    /* absolute/relative menu */
{
    "Absolute",
    "Relative"
};
static char menu4[][STRING_SIZE]=    /* plotting menu */
{
    "Save data to file",
    "Save picture file",
    "Exit"
};

/* select absolute/relative */

while ((iabs=menu("Select option:",menu3,2))==0);

/* select axes */

if (flag_2d==TRUE)
{
    x_axis=1;
    y_axis=2;
}
else
{
    while ((x_axis=menu("Select x axis:",menu1,3))==0);
    while ((y_axis=menu("Select y axis:",menu2,3))==0);
}

/* loop round number of lines */

nlines=0;
do
{

    /* select nodes */

    if (iabs==2)
    {
        while ((bnode=menu("Select reference node:",nodes,nnodes))==0);
        bnode--;
        while ((anode=menu("Select node:",nodes,nnodes))==0);
        anode--;
        strcpy(key[nlines],nodes[bnode]);
        strcat(key[nlines]," ");
        strcat(key[nlines],nodes[anode]);
    }
    else
    {
        while ((anode=menu("Select node:",nodes,nnodes))==0);
        anode--;
        strcpy(key[nlines],nodes[anode]);
    }

    /* loop round frames */

    for (iframe=0;iframe<nframe;iframe++)
    {

        /* assign reference node values */

        if (iabs==2)
```

```
        {
            refx=xpos[bnode][iframe];
            refy=ypos[bnode][iframe];
            refz=zpos[bnode][iframe];
        }

        /* put correct values into locus */

        switch (x_axis)
        {
        case 1:
            x_locus[nlines][iframe]=xpos[anode][iframe]-refx;
            break;

        case 2:
            x_locus[nlines][iframe]=ypos[anode][iframe]-refy;
            break;

        case 3:
            x_locus[nlines][iframe]=zpos[anode][iframe]-refz;
            break;
        }
        switch (y_axis)
        {
        case 1:
            y_locus[nlines][iframe]=xpos[anode][iframe]-refx;
            break;

        case 2:
            y_locus[nlines][iframe]=ypos[anode][iframe]-refy;
            break;

            case 3:
                y_locus[nlines][iframe]=zpos[anode][iframe]-refz;
                break;
            }
        }
        nlines++;
    } while (yesno("Another line?")==TRUE);

    /* draw graph */

    strcpy(title,menu3[iabs-1]);
    strcat(title," Node Locus");
    strcpy(x_label,menu1[x_axis-1]);
    strcat(x_label," (m)");
    strcpy(y_label,menu2[y_axis-1]);
    strcat(y_label," (m)");
    d_graph(display,title,x_label,y_label,x_locus,y_locus,nframe,nlines,key,TRUE);
    while ((iplot=menu("Select option:",menu4,3))!=3)
    {
        switch (iplot)
        {
        case 1:
            save_an(title,x_label,y_label,x_locus,y_locus,nframe,nlines,key);
            break;
        case 2:
            printf("Input picture file name : ");
            scanf("%s",fname);
            strcpy(filename,PICTURE_DIRECTORY);
            strcat(filename,PICTURE_PREFIX);
            strcat(filename,fname);
```

```
            strcat(filename,PICTURE_SUFFIX);

            bitmap_to_file(display,TRUE,0,0,filename,TRUE,0.0,0.0,0,0,TRUE);
            break;
        }
    }
    CLEAR_GRAPH;

    /* finished */

}
```

l_dynam()

```
#include "params.h"

/* routine to calculate linear dynamics */
/* NB. calculates resultant force on each segment */

void l_dynam(comxacc,comyacc,comzacc,nsegs,nframe,seg_mass,
    x_force,y_force,z_force)

float comxacc[][MAX_FRAMES];        /* x component of segment COM accln */
float comyacc[][MAX_FRAMES];        /* y component of segment COM accln */
float comzacc[][MAX_FRAMES];        /* z component of segment COM accln */
int nsegs;                           /* number of segments */
int nframe;                          /* number of frames */
float seg_mass[];                      /* segment masses */
float x_force[][MAX_FRAMES];        /* x component of force */
float y_force[][MAX_FRAMES];        /* y component of force */
float z_force[][MAX_FRAMES];        /* z component of force */
{
    int isegs;                        /* segment counter */
    int iframe;                        /* frame counter */

    /* loop round frames and segments */

    for (isegs=0;isegs<nsegs;isegs++)
    {
        for (iframe=0;iframe<nframe;iframe++)
        {
            x_force[isegs][iframe]=seg_mass[isegs]*comxacc[isegs][iframe];
            y_force[isegs][iframe]=seg_mass[isegs]*(comyacc[isegs][iframe]);
            z_force[isegs][iframe]=seg_mass[isegs]*comzacc[isegs][iframe];
        }
    }
}
```

l_kinem()

```
#include "params.h"

void l_kinem(nnodes,xpos,ypos,zpos,nframe,fspeed,times,xvel,yvel,zvel,
    xacc,yacc,zacc)

/* this routine calculates the linear kinematic parameters */

int nnodes;                          /* Number of nodes */
float xpos[][MAX_FRAMES];     /* x world coordinates (m) */
```

```
float ypos[][MAX_FRAMES];    /* y world coordinates (m) */
float zpos[][MAX_FRAMES];    /* z world coordinates (m) */
int nframe;                    /* number of frames */
float fspeed;                   /* Film frame interval (s) */
float times[MAX_FRAMES];       /* times (s) */
float xvel[][MAX_FRAMES];    /* calculated velocities (m/s) */
float yvel[][MAX_FRAMES];
float zvel[][MAX_FRAMES];
float xacc[][MAX_FRAMES];    /* calculated accelerations (m/s/s) */
float yacc[][MAX_FRAMES];
float zacc[][MAX_FRAMES];

{
    int iframe;                 /* frame counter */
    int inode;                 /* node counter */
    float time;                 /* time counter (s) */

    /* check sufficient frames */

    if (nframe<5)
    {
        printf("Insufficient frames\n");
        return;
    }

    /* loop over nodes */

    for (inode=0;inode<nnodes;inode++)
    {

        /* calculate velocities and accelerations */

        different(times,xpos[inode],nframe,xvel[inode],xacc[inode],FALSE);
        different(times,ypos[inode],nframe,yvel[inode],yacc[inode],FALSE);
        different(times,zpos[inode],nframe,zvel[inode],zacc[inode],FALSE);
    }
}
```

menu()

```
#include "params.h"

int menu(title,prompt,nprompt)

/* this routine prints up on the screen a general format mouse selection menu */

char title[];                    /* the title centred at the top of the screen */
char prompt[][STRING_SIZE];     /* these are the selection options */
int nprompt;                     /* this is the number of prompts */

{

    int iret;                     /* this is the value of the returned selection*/
    char menu_prompts[MENU_PAGE+1][STRING_SIZE];    /* this is the menu buffer */
    int iprompt;                  /* this is the prompt pointer */
    int jprompt;                  /* prompt index in multi-page menu */
    int kprompt;                  /* menu size in multi-page menu */
    int npage;                   /* number of pages in multi-page menu */
    int ipage;                   /* page index */

    /* test menu size */
```

```
if (nprompt<=MENU_PAGE)
{

    /* get the string into the correct format for 'domenu' */

    for (iprompt=0;iprompt<nprompt;iprompt++)
    {
        strcpy(menu_prompts[iprompt],prompt[iprompt]);
    }

    /* and call the menu routine */

    iret=domenu(title,menu_prompts,nprompt);
}
else
{

    /* calculate the number of pages */

    npage=nprompt/MENU_PAGE;
    ipage=0;
    do
    {

        /* get the string into the correct format for 'domenu' */

        kprompt=0;
        for (iprompt=0;iprompt<MENU_PAGE;iprompt++)
        {
            jprompt=iprompt+MENU_PAGE*ipage;
            if (jprompt<nprompt)
            {
                strcpy(menu_prompts[kprompt],prompt[jprompt]);
                kprompt++;
            }
        }
        strcpy(menu_prompts[kprompt],"Next page");
        kprompt++;

        /* and call the menu routine */

        iret=domenu(title,menu_prompts,kprompt);

        /* calculate next page number */

        ipage= ipage<npage ? ipage+1:0;
    } while (iret==kprompt);

    /* recalculate page number */

        ipage= ipage==0 ? npage:ipage-1;
        if (iret!=0) iret=iret+ipage*MENU_PAGE;
    }

    /* return menu selection value */

    return(iret);

}
```

move_cam()

```
#include "params.h"

/* this routine uses the knobs to move the camera view position */

void move_cam(fd,camera)

int fd;                              /* output device file pointer */
camera_arg *camera;                  /* camera model structure */

{
    int valid;                      /* valid response flag */
    float value[9];                  /* new value of knobs */
    int string[STRING_SIZE];         /* text output string */

sample_locator(knobs3,1,&valid,&value[0],&value[1],&value[2]);
sample_locator(knobs2,1,&valid,&value[3],&value[4],&value[5]);
sample_locator(knobs1,1,&valid,&value[6],&value[7],&value[8]);

sprintf(string,"Viewpoint: X =%5.2f  Y =%5.2f  Z =%5.2f",
    value[0],value[1],value[2]);
dctext(display,SIDE_BORDER+10,TOP_BORDER+20,string);
sprintf(string,"Target   : X =%5.2f  Y =%5.2f  Z =%5.2f",
    value[3],value[4],value[5]);
dctext(display,SIDE_BORDER+10,TOP_BORDER+40,string);
sprintf(string,"Field    :   =%5.2f",value[6]);
dctext(display,SIDE_BORDER+10,TOP_BORDER+60,string);

/* camera position */

(*camera).camx=value[0];
(*camera).camy=value[1];
(*camera).camz=(-value[2]);

/* camera target */

    (*camera).refx=value[3];
    (*camera).refy=value[4];
    (*camera).refz=(-value[5]);

    /* field of view */

    (*camera).field_of_view=value[6];

    view_camera(fd,camera);

}
```

node_plot()

```
#include "params.h"

void node_plot(nodes,nnodes,xpos,ypos,zpos,startframe,endframe,
    times,p_title,p_ylabel)

/* this routine plots the selected node positions/velocities/accelerations */
/* depending on the data arrays passed over to the routine */

int nnodes;                          /* Number of nodes */
char nodes[][STRING_SIZE];           /* Names of nodes */
float xpos[][MAX_FRAMES];            /* x world coordinates (pos/vel/acc) */
float ypos[][MAX_FRAMES];            /* y world coordinates */
```

```
float zpos[][MAX_FRAMES];        /* z world coordinates */
int startframe;                   /* start frame number */
int endframe;                      /* end frame number */
float times[MAX_FRAMES];          /* times (s) */
char p_title[STRING_SIZE];       /* plot title */
char p_ylabel[STRING_SIZE];       /* plot y axis label */

{
    float pl_times[MAX_LINES][MAX_POINTS];    /* times to be plotted (s) */
    float distance[MAX_LINES][MAX_POINTS];    /* calculated
distances/velocities/accelerations */
    char key[MAX_LINES+1][STRING_SIZE];    /* graph key string */
    char title[STRING_SIZE];          /* graph title */
    char x_label[STRING_SIZE];    /* graph x axis label */
    char y_label[STRING_SIZE];    /* graph y axis label */
    char fname[STRING_SIZE];       /* picture filename */
    char filename[STRING_SIZE];    /* full picture filename */
    int iframe;                    /* frame counter */
    int inode;                    /* node counter */
    float xref=0.0,yref=0.0,zref=0.0;    /* x y z refence values */
    float xint,yint,zint;            /* x y z intervals */
    int iabs=1;                    /* relative/absolute indicator */
    int idir;                       /* direction indicator */
    int iplot;                    /* plotter control */
    int nlines;                    /* number of lines on graph */
    int anode;                    /* start node number */
    int bnode;                    /* finish node number */

    static char menu1[][STRING_SIZE]=    /* absolute or relative menu */
    {
        "Absolute",
        "Relative"
    };
    static char menu2[][STRING_SIZE]=    /* direction menu */
    {
        "X",
        "Y",
        "Z",
        "3D"
    };
    static char menu2b[][STRING_SIZE]=    /* direction menu for 2d */
    {
        "X",
        "Y",
        "2D"
    };
    static char menu3[][STRING_SIZE]=    /* plotting menu */
    {
        "Save data to file",
        "Save picture to file",
        "Exit"
    };

    /* all nodes */

    if (nnodes<MAX_LINES && yesno("Plot all nodes?")==TRUE)
    {

        if (flag_2d==TRUE)
        {
            while ((idir=menu("Direction:",menu2b,3))==0);
            strcpy(key[nnodes],menu2b[idir-1]);
```

```
        strcat(key[nnodes]," ");
        if (idir==3) idir=4;
    }
    else
    {
        while ((idir=menu("Direction:",menu2,4))==0);
        strcpy(key[nnodes],menu2[idir-1]);
        strcat(key[nnodes]," ");
    }

    for (nlines=0;nlines<nnodes;nlines++)
    {
        strcpy(key[nlines],key[nnodes]);
        strcat(key[nlines],nodes[nlines]);

    /* loop round frames */

    for (iframe=startframe;iframe<endframe;iframe++)
    {
        /* calculate times */

        pl_times[nlines][iframe-startframe]=times[iframe];

        /* calculate distance */

        switch (idir)
        {
        case 1:
            distance[nlines][iframe-startframe]=
                xpos[nlines][iframe];
            break;

        case 2:
            distance[nlines][iframe-startframe]=
                ypos[nlines][iframe];
            break;

        case 3:
            distance[nlines][iframe-startframe]=
                zpos[nlines][iframe];
            break;

        case 4:
            xint=xpos[nlines][iframe];
            yint=ypos[nlines][iframe];
            zint=zpos[nlines][iframe];
            distance[nlines][iframe-startframe]=
                sqrt(xint*xint+yint*yint+zint*zint);
            break;
        }
            }
        }
    }
    else
    {

        /* first option */

        while ((iabs=menu("Option:",menu1,2))==0);

        /* loop round number of lines */
```

```
nlines=0;
do
{

    /* second option */

    if (flag_2d==TRUE)
    {
        while ((idir=menu("Direction:",menu2b,3))==0);
        strcpy(key[nlines],menu2b[idir-1]);
        strcat(key[nlines]," ");
        if (idir==3) idir=4;
    }
    else
    {
        while ((idir=menu("Direction:",menu2,4))==0);
        strcpy(key[nlines],menu2[idir-1]);
        strcat(key[nlines]," ");
    }

    if (iabs==2)
    {
        while ((anode=menu("Select reference node:",nodes,nnodes))
            ==0);
        anode--;
        strcat(key[nlines],nodes[anode]);
        strcat(key[nlines]," ");
    }
    while ((bnode=menu("Select node:",nodes,nnodes))==0);
    bnode--;
    strcat(key[nlines],nodes[bnode]);

    /* loop round frames */

    for (iframe=startframe;iframe<endframe;iframe++)
    {

        /* calculate times */

        pl_times[nlines][iframe-startframe]=times[iframe];

        /* calculate distance */

        if (iabs==2)
        {
            xref=xpos[anode][iframe];
            yref=ypos[anode][iframe];
            zref=zpos[anode][iframe];
        }

        switch (idir)
        {
        case 1:
            distance[nlines][iframe-startframe]=
                xpos[bnode][iframe]-xref;
            break;

        case 2:
            distance[nlines][iframe-startframe]=
                ypos[bnode][iframe]-yref;
            break;
```

```
                case 3:
                    distance[nlines][iframe-startframe]=
                        zpos[bnode][iframe]-zref;
                    break;

                case 4:
                    xint=xpos[bnode][iframe]-xref;
                    yint=ypos[bnode][iframe]-yref;
                    zint=zpos[bnode][iframe]-zref;
                    distance[nlines][iframe-startframe]=
                        sqrt(xint*xint+yint*yint+zint*zint);
                    break;
                }
            }
            nlines++;
        } while (nlines<MAX_LINES && yesno("Another line?")==TRUE);
}

/* draw graph */

strcpy(title,menu1[iabs-1]);
strcat(title," ");
strcat(title,p_title);
strcpy(x_label,"Time (s)");
strcpy(y_label,p_ylabel);
d_graph(display,title,x_label,y_label,pl_times,distance,
    endframe-startframe,nlines,
    key,FALSE);
while ((iplot=menu("Select option:",menu3,3))!=3)
{
    switch (iplot)
    {
    case 1:
        save_an(title,x_label,y_label,pl_times,distance,endframe-startframe,
            nlines,key);
        break;
    case 2:
        printf("Input picture file name : ");
        scanf("%s",fname);
        strcpy(filename,PICTURE_DIRECTORY);
        strcat(filename,PICTURE_PREFIX);
        strcat(filename,fname);
        strcat(filename,PICTURE_SUFFIX);

        bitmap_to_file(display,TRUE,0,0,filename,TRUE,0.0,0.0,0,0,TRUE);
        break;
    }
}
CLEAR_GRAPH;

    /* finished */

}
```

open_dev()

```
#include "params.h"

void open_dev()

/* This routine opens the devices using Starbase gopen routines */
```

```
/* It also checks that the program is being run from a window */

{
    char *graphic_window_name;     /* graphic window path name */
    char *text_window_name;         /* graphic window path name */
    Window wdummy;                 /* dummy window return */
    int idummy;                     /* dummy return */
    xrWindowData windowdata;        /* Xr window data structure */

    /* connect to X windows */

    if ((xdisplay=XOpenDisplay(NULL))==NULL) exit(-1);
    xscreen=XDefaultScreen(xdisplay);

    /* add Xrlib */

    if (XrInit(xdisplay,xscreen,NULL)==FALSE) exit(-1);

    /* get hold of terminal window */

    XGetInputFocus(xdisplay,&text_window,&idummy);
    XGetGeometry(xdisplay,text_window,&wdummy,&orig_x,&orig_y,&orig_width,
        &orig_height,&idummy,&idummy);

    /* create graphics window */

    system(XSEETHRU);
    XFlush(xdisplay);

    /* screen */

    if ((display=gopen(getenv("SB_OUTDEV"),OUTDEV,getenv("SB_OUTDRIVER"),
        INIT|THREE_D|MODEL_XFORM))==-1) exit(-1);

    /* set up some useful defaults */

    clear_control(display,CLEAR_DISPLAY_SURFACE|CLEAR_ZBUFFER);
    shade_mode(display,CMAP_FULL|INIT,FALSE);

    /* knob box */

    if ((knobs1=gopen("/dev/knob1",INDEV,"hp-hil",INIT))==-1) exit(-1);
    knobs2=gopen("/dev/knob2",INDEV,"hp-hil",INIT);
    knobs3=gopen("/dev/knob3",INDEV,"hp-hil",INIT);

    /* button box */

    if ((bbox=gopen("/dev/bbox",INDEV,"hp-hil",INIT))==-1) exit(-1);

    /* get hold of see thru window - done now to give time to create */

    XGetInputFocus(xdisplay,&graphic_window,&idummy);

    /* menu window - not mapped */

    menu_window=XCreateSimpleWindow(xdisplay,RootWindow(xdisplay,xscreen),
        20,20,40,40,0,BlackPixel(xdisplay,xscreen),WhitePixel(xdisplay,xscreen));

    /* set up XrInput */

    XrInput(menu_window,MSG_ADDWINDOW,&windowdata);
```

```
    XSelectInput(xdisplay,menu_window,ButtonPressMask|ButtonReleaseMask|
        KeyPressMask|ExposureMask);

    /* shift around the main input window */

    XFlush(xdisplay);
    XMoveWindow(xdisplay,text_window,SIDE_BORDER,GRAPH_HEIGHT+2*TOP_BORDER+
        BOTTOM_BORDER);
    XResizeWindow(xdisplay,text_window,TEXT_WIDTH,TEXT_HEIGHT);
    XRaiseWindow(xdisplay,text_window);

    /* get everything up to date */

    XFlush(xdisplay);

    /* attach keyboard to text window */

    XSetInputFocus(xdisplay,text_window,RevertToParent,CurrentTime);

}
/* routine to set up global options */
```

options()

```
#include "params.h"

void options()
{
    int iret;                   /* menu return value */
    static char menu1[][STRING_SIZE]=    /* menu */
    {
        "2d On",
        "Flexible 3d reconstruction",
        "No fiducial marks",
        "Set frame increment",
        "Set filtration cutoff",
        "Change working directory",
        "Smoothing",
        "Set smoothing number",
        "Exit"
    };
    int nmenu1=9;                   /* number of menu items */
    static char menu2[][STRING_SIZE]=    /* menu for filtration stuff */
    {
        "4.0",
        "5.0",
        "6.0",
        "7.0",
        "8.5",
        "10.0",
        "12.0",
        "14.0",
        "16.0",
        "18.0",
        "20.0"
    };
    int nmenu2=11;                  /* number of menu items */
    static char menu3[][STRING_SIZE]=    /* menu for smoothing */
    {
        "3",
        "5",
```

```
        "7",
        "9",
        "11"
    };
    int nmenu3=5;
    char directory[STRING_SIZE];        /* directory string */

    do
    {

        /* copy in correct menu prompts */

        if (flag_2d==TRUE) strcpy(menu1[0],"2d off");
        else strcpy(menu1[0],"2d on");

        if (flag_simple_reconstruction==TRUE)
            strcpy(menu1[1],"Flexible 3d reconstruction");
        else strcpy(menu1[1],"Simple 3d reconstruction");

        if (fiducial_flag==TRUE) strcpy(menu1[2],"No fiducial marks");
        else strcpy(menu1[2],"Fiducial marks");

        if (flag_filter==TRUE) strcpy(menu1[6],"Smoothing");
        else strcpy(menu1[6],"Filtration");

        /* write menu */

        iret=menu("Select option:",menu1,nmenu1);

        switch(iret)
        {

        case 1:
            if (flag_2d==TRUE) flag_2d=FALSE;
            else flag_2d=TRUE;
            break;

        case 2:
            if (flag_simple_reconstruction==TRUE)
                flag_simple_reconstruction=FALSE;
            else flag_simple_reconstruction=TRUE;
            break;

        case 3:
            if (fiducial_flag==TRUE) fiducial_flag=FALSE;
            else fiducial_flag=TRUE;
            break;

        case 4:
            printf("Current frame increment is %d\n",frame_increment);
            printf("Required increment is : ");
            scanf("%d",&frame_increment);
            break;

        case 5:
            while ((filtration_number=menu("Select fs/fc",menu2,nmenu2))==0);
            break;

        case 6:
            printf("Directory name : ");
            scanf("%s",directory);
            chdir(directory);
```

```
        getcwd(directory,STRING_SIZE);
        printf("Changed to : %s\n",directory);
        break;

    case 7:
        if (flag_filter==TRUE) flag_filter=FALSE;
        else flag_filter=TRUE;
        break;

    case 8:
        while((smooth_number=menu("Select smoothing number:",menu3,
            nmenu3))==0);
        smooth_number=smooth_number*2+1;
        break;
    }
  } while (iret!=nmenu1);
}
```

predictive_analysis()

```
#include "params.h"

/* routine to calculate joint torques and reaction forces for predictive */
/* leaping model */

predictive_analysis(seg_length,seg_mass,seg_com,phi,x_force,y_force,torque,
    segs,nsegs,nframe,times)

float seg_length[MAX_SEGS];            /* mean segment lengths */
float seg_mass[MAX_SEGS];             /* segment masses */
float seg_com[MAX_SEGS];               /* relative COM positions */
float phi[MAX_SEGS][MAX_FRAMES];    /* segment angle */
float x_force[MAX_SEGS][MAX_FRAMES];    /* x component of linear force */
float y_force[MAX_SEGS][MAX_FRAMES];    /* y component of linear force */
float torque[MAX_SEGS][MAX_FRAMES];    /* torques about segments */
char segs[MAX_SEGS][STRING_SIZE];    /* segment names */
int nsegs;                             /* number of segments */
int nframe;                            /* number of frames */
float times[MAX_FRAMES];               /* frame times */
{
    int iframe;                        /* frame counter */
    float d1,d2;                       /* distances from COM */
    struct body                        /* structure for free body data */
    {
        float r1x[MAX_FRAMES];         /* reaction force at 'proximal' end */
        float r1y[MAX_FRAMES];
        float r2x[MAX_FRAMES];         /* reaction forces at 'distal' end */
        float r2y[MAX_FRAMES];
        float m1[MAX_FRAMES];          /* torque at 'proximal' end */
        float m2[MAX_FRAMES];          /* torque at 'distal' end */
        int segnum;                    /* segment number */
    };
    struct body forefoot;              /* segments */
    struct body hindfoot;
    struct body calf;
    struct body thigh;
    struct body torso;
    float x_react[MAX_SEGS][MAX_FRAMES];    /* reaction forces */
    float y_react[MAX_SEGS][MAX_FRAMES];    /* reaction forces */
    float j_torque[MAX_SEGS][MAX_FRAMES];    /* joint torques */
    float work_done[MAX_SEGS][MAX_FRAMES];    /* work done at joints */
```

```
float bend[MAX_SEGS][MAX_FRAMES];          /* bending moment on segments */
float dummy[MAX_SEGS][MAX_FRAMES];          /* dummy values for plots */
float angle;                               /* intermediate angle value */
static char joints[][STRING_SIZE]=          /* joints etc. */
{
    "Contact point",
    "Mid-tarsal joint",
    "Ankle",
    "Knee",
    "Hip",
};
int njoints=5;                             /* number of joints */
static char menu1[][STRING_SIZE]=           /* printout menu */
{
    "Joint reactions",
    "Joint torques",
    "Work done per frame",
    "Bending moments",
    "Exit"
};
int nmenu1=5;                              /* number of menu items */
int iret;                                  /* menu return value */

/* initialize segment ID numbers */

forefoot.segnum=0;
hindfoot.segnum=1;
calf.segnum=2;
thigh.segnum=3;
torso.segnum=4;

/* loop over frames */

for (iframe=1;iframe<(nframe-1);iframe++)
{
    d1=seg_com[torso.segnum]*seg_length[torso.segnum];
    d2=(1-seg_com[torso.segnum])*seg_length[torso.segnum];
    torso.r2x[iframe]=0.0;
    torso.r2y[iframe]=0.0;
    torso.m2[iframe]=0.0;
    torso.r1x[iframe]=x_force[torso.segnum][iframe]-torso.r2x[iframe];
    torso.r1y[iframe]=y_force[torso.segnum][iframe]-torso.r2y[iframe]-
        seg_mass[torso.segnum]*G;
    torso.m1[iframe]=torque[torso.segnum][iframe]-
        torso.r1x[iframe]*d1*
        (float)sin((double)phi[torso.segnum][iframe])+
        torso.r2x[iframe]*d2*
        (float)sin((double)phi[torso.segnum][iframe])+
        torso.r1y[iframe]*d1*
        (float)cos((double)phi[torso.segnum][iframe])-
        torso.r2y[iframe]*d2*
        (float)cos((double)phi[torso.segnum][iframe])-
        torso.m2[iframe];

    d1=seg_com[thigh.segnum]*seg_length[thigh.segnum];
    d2=(1-seg_com[thigh.segnum])*seg_length[thigh.segnum];
    thigh.r2x[iframe]=(-torso.r1x[iframe]);
    thigh.r2y[iframe]=(-torso.r1y[iframe]);
    thigh.m2[iframe]=(-torso.m1[iframe]);
    thigh.r1x[iframe]=x_force[thigh.segnum][iframe]-thigh.r2x[iframe];
    thigh.r1y[iframe]=y_force[thigh.segnum][iframe]-thigh.r2y[iframe]-
        seg_mass[thigh.segnum]*G;
```

```
        thigh.m1[iframe]=torque[thigh.segnum][iframe]-
            thigh.r1x[iframe]*d1*
            (float)sin((double)phi[thigh.segnum][iframe])+
            thigh.r2x[iframe]*d2*
            (float)sin((double)phi[thigh.segnum][iframe])+
            thigh.r1y[iframe]*d1*
            (float)cos((double)phi[thigh.segnum][iframe])-
            thigh.r2y[iframe]*d2*
            (float)cos((double)phi[thigh.segnum][iframe])-
            thigh.m2[iframe];

        d1=seg_com[calf.segnum]*seg_length[calf.segnum];
        d2=(1-seg_com[calf.segnum])*seg_length[calf.segnum];
        calf.r2x[iframe]=(-thigh.r1x[iframe]);
        calf.r2y[iframe]=(-thigh.r1y[iframe]);
        calf.m2[iframe]=(-thigh.m1[iframe]);
        calf.r1x[iframe]=x_force[calf.segnum][iframe]-calf.r2x[iframe];
        calf.r1y[iframe]=y_force[calf.segnum][iframe]-calf.r2y[iframe]-
            seg_mass[calf.segnum]*G;
        calf.m1[iframe]=torque[calf.segnum][iframe]-
            calf.r1x[iframe]*d1*
            (float)sin((double)phi[calf.segnum][iframe])+
            calf.r2x[iframe]*d2*
            (float)sin((double)phi[calf.segnum][iframe])+
            calf.r1y[iframe]*d1*
            (float)cos((double)phi[calf.segnum][iframe])-
            calf.r2y[iframe]*d2*
            (float)cos((double)phi[calf.segnum][iframe])-
            calf.m2[iframe];

        d1=seg_com[hindfoot.segnum]*seg_length[hindfoot.segnum];
        d2=(1-seg_com[hindfoot.segnum])*seg_length[hindfoot.segnum];
        hindfoot.r2x[iframe]=(-calf.r1x[iframe]);
        hindfoot.r2y[iframe]=(-calf.r1y[iframe]);
        hindfoot.m2[iframe]=(-calf.m1[iframe]);
        hindfoot.r1x[iframe]=x_force[hindfoot.segnum][iframe]-hindfoot.r2x[iframe];
        hindfoot.r1y[iframe]=y_force[hindfoot.segnum][iframe]-hindfoot.r2y[iframe]-
            seg_mass[hindfoot.segnum]*G;
        hindfoot.m1[iframe]=torque[hindfoot.segnum][iframe]-
            hindfoot.r1x[iframe]*d1*
            (float)sin((double)phi[hindfoot.segnum][iframe])+
            hindfoot.r2x[iframe]*d2*
            (float)sin((double)phi[hindfoot.segnum][iframe])+
            hindfoot.r1y[iframe]*d1*
            (float)cos((double)phi[hindfoot.segnum][iframe])-
            hindfoot.r2y[iframe]*d2*
            (float)cos((double)phi[hindfoot.segnum][iframe])-
            hindfoot.m2[iframe];

        d1=seg_com[forefoot.segnum]*seg_length[forefoot.segnum];
        d2=(1-seg_com[forefoot.segnum])*seg_length[forefoot.segnum];
        forefoot.r2x[iframe]=(-hindfoot.r1x[iframe]);
        forefoot.r2y[iframe]=(-hindfoot.r1y[iframe]);
        forefoot.m2[iframe]=(-hindfoot.m1[iframe]);
        forefoot.r1x[iframe]=x_force[forefoot.segnum][iframe]-forefoot.r2x[iframe];
        forefoot.r1y[iframe]=y_force[forefoot.segnum][iframe]-forefoot.r2y[iframe]-
            seg_mass[forefoot.segnum]*G;
        forefoot.m1[iframe]=torque[forefoot.segnum][iframe]-
            forefoot.r1x[iframe]*d1*
            (float)sin((double)phi[forefoot.segnum][iframe])+
            forefoot.r2x[iframe]*d2*
            (float)sin((double)phi[forefoot.segnum][iframe])+
```

```
        forefoot.r1y[iframe]*d1*
        (float)cos((double)phi[forefoot.segnum][iframe])-
        forefoot.r2y[iframe]*d2*
        (float)cos((double)phi[forefoot.segnum][iframe])-
        forefoot.m2[iframe];

    /* put into array for display */

    x_react[0][iframe]=forefoot.r1x[iframe];    /* contact point */
    y_react[0][iframe]=forefoot.r1y[iframe];
    j_torque[0][iframe]=forefoot.m1[iframe];
    angle=phi[forefoot.segnum][iframe+1]-phi[forefoot.segnum][iframe-1];
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[0][iframe]=j_torque[0][iframe]*0.5*angle;

    x_react[1][iframe]=hindfoot.r1x[iframe];    /* mid-tarsal joint */
    y_react[1][iframe]=hindfoot.r1y[iframe];
    j_torque[1][iframe]=hindfoot.m1[iframe];
    angle=(phi[hindfoot.segnum][iframe+1]-phi[hindfoot.segnum][iframe-1]-
        phi[forefoot.segnum][iframe+1]+phi[forefoot.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[1][iframe]=j_torque[1][iframe]*0.5*angle;

    x_react[2][iframe]=calf.r1x[iframe];        /* ankle */
    y_react[2][iframe]=calf.r1y[iframe];
    j_torque[2][iframe]=calf.m1[iframe];
    angle=(phi[calf.segnum][iframe+1]-phi[calf.segnum][iframe-1]-
        phi[hindfoot.segnum][iframe+1]+phi[hindfoot.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[2][iframe]=j_torque[2][iframe]*0.5*angle;

    x_react[3][iframe]=thigh.r1x[iframe];       /* knee */
    y_react[3][iframe]=thigh.r1y[iframe];
    j_torque[3][iframe]=thigh.m1[iframe];
    angle=(phi[thigh.segnum][iframe+1]-phi[thigh.segnum][iframe-1]-
        phi[calf.segnum][iframe+1]+phi[calf.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[3][iframe]=j_torque[3][iframe]*0.5*angle;

    x_react[4][iframe]=torso.r1x[iframe];       /* hip */
    y_react[4][iframe]=torso.r1y[iframe];
    j_torque[4][iframe]=torso.m1[iframe];
    angle=(phi[torso.segnum][iframe+1]-phi[torso.segnum][iframe-1]-
        phi[thigh.segnum][iframe+1]+phi[thigh.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[4][iframe]=j_torque[4][iframe]*0.5*angle;

    bend[forefoot.segnum][iframe]=forefoot.m1[iframe]-forefoot.m2[iframe];
    bend[hindfoot.segnum][iframe]=hindfoot.m1[iframe]-hindfoot.m2[iframe];
    bend[calf.segnum][iframe]=calf.m1[iframe]-calf.m2[iframe];
    bend[thigh.segnum][iframe]=thigh.m1[iframe]-thigh.m2[iframe];
    bend[torso.segnum][iframe]=torso.m1[iframe]-torso.m2[iframe];

}

while ((iret=menu("Select option:",menu1,nmenu1))!=nmenu1)
{
```

```
        switch (iret)
        {
        case 1:
            node_plot(joints,njoints,x_react,y_react,dummy,1,nframe-1,
                times,"Reaction Forces","Force (N)");
            break;

        case 2:
            seg_plot(dummy,dummy,j_torque,1,nframe-1,joints,njoints,
                times,"Joint Torques","Torques (Nm)");
            break;


        case 3:
            seg_plot(dummy,dummy,work_done,1,nframe-1,joints,njoints,
                times,"Work Done per frame","Energy (J)");
            break;


        case 4:
            seg_plot(dummy,dummy,bend,1,nframe-1,segs,nsegs,
                times,"Bending Moments","Moment (Nm)");
            break;
        }
    }
}
```

rdlimb()

```
#include "params.h"

void rdlimb(title,nodes,nnodes,segs,nsegs,nodes_per_seg,seg_mass,seg_com,seg_moi)

/* this routine reads in the limb model data from the specified file */

char title[STRING_SIZE];                /* file title line */
char nodes[MAX_NODES][STRING_SIZE];     /* names of the nodes of the model */
int *nnodes;                            /* the number of nodes */
char segs[MAX_SEGS][STRING_SIZE];       /* names of the segments in the model */
int *nsegs;                             /* the number of segments */
int nodes_per_seg[NPS][MAX_SEGS];       /* array of node pairs for each segment */
float seg_mass[MAX_SEGS];               /* array of segment masses */
float seg_com[MAX_SEGS];                /* array of segment relative COMs */
float seg_moi[MAX_SEGS];                /* array of segment MOIs */
{

    FILE *unit;                         /* file pointer */
    int inode;                          /* node number from file */
    int inodes;                         /* node number from counter */
    int iseg;                           /* segment number from file */
    int isegs;                          /* segment number from counter */
    int i;                              /* counter */
    char fname[STRING_SIZE];            /* filename */
    char a,b,c,d,e;

    do
    {

        /* get file name */

        sel_file(LIMB_DIRECTORY,LIMB_PREFIX,LIMB_SUFFIX,fname);
```

```
    /* open the file */

    unit=fopen(fname,"r");
}while (unit==NULL);

/* read the data */

fscanf(unit,"'%[^']'",title);
fscanf(unit,"%d",nnodes);
for (inodes=0;inodes<*nnodes;inodes++)
{
    fscanf(unit,"%d '%[^']'",&inode,nodes[inodes]);
    if (inode!=inodes)
    {
        printf("Node number mismatch during file read error\n");
        exit(-1);
    }
}

fscanf(unit,"%d",nsegs);
for (isegs=0;isegs<*nsegs;isegs++)
{
    fscanf(unit,"%d '%[^']'",&iseg,segs[isegs]);
    if (iseg!=isegs)
    {
        printf("Segment number mismatch during file read error\n");
        exit(-1);
    }
    for (i=0;i<NPS;i++)
    {
        fscanf(unit,"%d",&nodes_per_seg[i][isegs]);
    }
    fscanf(unit,"%f",&seg_mass[isegs]);
    fscanf(unit,"%f",&seg_com[isegs]);
    fscanf(unit,"%f",&seg_moi[isegs]);
}

/* close the file */

fclose(unit);

/* print success message */

printf("File %s read successfully\n",fname);

/* finished */

}
```

rdnode()

```
#include "params.h"

void rdnode(title,nodes,nnodes,xpos,ypos,zpos,nframe,fspeed)

/* this routine reads in the node position file */

char title[STRING_SIZE];                    /* file title line */
char nodes[MAX_NODES][STRING_SIZE];        /* names of the nodes of the model */
int *nnodes;                                /* the number of nodes */
```

```
float xpos[MAX_NODES][MAX_FRAMES];    /* the x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];    /* the y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];    /* the z world coordinates */
int *nframe;                           /* the number of frames */
int *fspeed;                           /* the interval between frames */

{

    FILE *unit;                      /* file pointer */
    char fname[STRING_SIZE];          /* filename */
    int iframe;                      /* file frame number */
    int iframes;                      /* counter frame number */
    int inode;                      /* file node number */
    int inodes;                       /* counter node number */

do
{

    /* get filename */

    sel_file(NODE_DIRECTORY,NODE_PREFIX,NODE_SUFFIX,fname);

    /* open file */

    unit=fopen(fname,"r");
} while (unit==NULL);

/* read data */

fscanf(unit,"%[^\n]\n",title);
fscanf(unit,"%f\n",fspeed);
fscanf(unit,"%d\n",nframe);
for (iframes=0;iframes<*nframe;iframes++)
{
    fscanf(unit,"%d\n",&iframe);
    if (iframe!=iframes)
    {
        printf("Frame number mismatch in node data file\n");
        exit(-1);
    }
    fscanf(unit,"%d\n",nnodes);
    for (inodes=0;inodes<*nnodes;inodes++)
    {
        fscanf(unit,"%d%f%f%f\n",&inode,&xpos[inodes][iframes],
            &ypos[inodes][iframes],&zpos[inodes][iframes]);
        fscanf(unit,"%[^\n]\n",nodes[inodes]);
        if (inode!=inodes)
        {
            printf("Node number mismatch in node data file\n");
            exit(-1);
        }
        if (flag_2d) zpos[inodes][iframes]=0.0;
    }
}

/* close file */

fclose(unit);

/* print success message */

printf("File %s read successfully\n",fname);
```

```
    /* finished */

}
```

read2d()

```
#include "params.h"

void read2d(xcoord,ycoord)

/* this routine returns the 2d world coordinates of a point specified by
   a position located on the picture */

float *xcoord;              /* x coordinate returned by the subroutine */
float *ycoord;              /* y coordinate returned by subroutine */

{

    float qa,ra;              /* screen coordinates */

    /* prompt for coordinate from pictures */

    printf("Select point:\n");
    digrd(&qa,&ra);

    /* fiducial point */

    if (fiducial_flag)
    {
        qa=qa+fiducial_x-correction_x;
        ra=ra+fiducial_y-correction_y;
    }

    /* correct for offset and scale factor */

    qa=scale_factor*(qa-x_offset);
    ra=scale_factor*(ra-y_offset);

    /* correct for rotation */

    *xcoord=rotation[0][0]*qa+rotation[0][1]*ra;
    *ycoord=rotation[1][0]*qa+rotation[1][1]*ra;

}
```

read3d()

```
#include "params.h"

void read3d(xcoord,ycoord,zcoord)

/* this routine returns the 3d world coordinates of a point specified by
   two orthogonal pictures. origin points and camera distances are specified
   as input parameters. */

float *xcoord;              /* x coordinate returned by the subroutine */
float *ycoord;              /* y coordinate returned by subroutine */
float *zcoord;              /* z coordinate returned by the subroutine */
```

```
{

    float qa,ra;            /* camera 1 screen coordinates */
    float qb,rb;            /* camera 2 screen coordinates */

    /* prompt for coordinate from pictures */

    printf("Select point on picture 1\n");
    digrd(&qa,&ra);
    printf("Select point on picture 2\n");
    digrd(&qb,&rb);

    /* fiducial point */

    if (fiducial_flag)
    {
        qa=qa+fiducial_x-correction_x;
        ra=ra+fiducial_y-correction_y;
        qb=qb+fiducial_x-correction_x;
        rb=rb+fiducial_y-correction_y;
    }

    /* perform required reconstruction */

    if (flag_simple_reconstruction==TRUE)
        simple_recon(qa,ra,qb,rb,xcoord,ycoord,zcoord);
    else dlt_recon(l1,l2,&qa,&ra,&qb,&rb,xcoord,ycoord,zcoord);

#if 0
    printf("Position: (%f,%f,%f)\n",*xcoord,*ycoord,*zcoord);
#endif
    /* finished */

}
```

readpic()

```
#include "params.h"

/* This is a routine to read the data contained in an image (.bm) file */

void readpic()

{
    FILE *unit;                         /* file pointer */
    char filename[STRING_SIZE];         /* filename */
    static char filename2[STRING_SIZE];    /* next filename in sequence */
    register char byte;                 /* pixel data read in */
    int xrange,yrange;                 /* range of file image data */
    int xlow,xhigh,ylow,yhigh;         /* range on screen for each file pixel */
    char buffer[FILE_MAX_X*FILE_MAX_Y];    /* buffer for file input */
    char pixel_data[GRAPH_HEIGHT][GRAPH_WIDTH];/* buffer for screen data */
    int x,y;                             /* screen pixel counters */
    int ix,iy;                         /* file data counters */
    char *buf_pointer;                 /* pointer for file buffer */
    int nbytes;                         /* number of bytes read in */
    static char menu1[][STRING_SIZE]=    /* file selection method menu */
    {
        "Specific file",
        "Set sequence name & number",
        "Next file in sequence"
```

```
};
static int sequence_num=0;                /* sequence number */
static char sequence_name[STRING_SIZE]="";    /* sequence name */
int iret;                                  /* return value for menu */
char string[STRING_SIZE];                 /* temporary string */

struct visilogImageHeader
{
    long int magicNumber;
    long int pixelsPerLine;
    long int numberOfLines;
    long int res1;
    long int res2;
    long int res3;
    long int gridType;                    /* rectangular grid */
    long int res4;
    long int arithmaticType;              /* long integer image */
    long int bitsPerPixel;
    long int res5;
    long int xOrigin;
    long int yOrigin;
    long int res6;
    long int res7;
    long int visilogHeaderSize;
    long int userHeaderSize;
    long int res8;
    long int totalHeaderSize;
} imageHeader;
float colourTable[256][3];
float red,green,blue;
int i;
int offset,range;
float grey;
int dum,button,valid;
float dummy;

/* Clear screen */

CLEAR_GRAPH;

/* setup display for anisometric greyscale 2d graphics */

/* Set up display window */

mapping_mode(display,DISTORT);
vdc_extent(display,0.0,0.0,0.0,1.0,1.0,0.0);

/* Set attributes */

shade_mode(display,CMAP_MONOTONIC|INIT,FALSE);

#if 0
    /* pseudo-colour map */

    for (i=0;i<256;i++)
    {
        if (i<0)
        {
            colourTable[i][0]=0.0;
        }
        else
        {
```

```
        if (i<128)
        {
            colourTable[i][0]=(float)(i-0)/128.0;
        }
        else
        {
            colourTable[i][0]=0.0;
        }
    }



    if (i<64)
    {
        colourTable[i][1]=0.0;
    }
    else
    {
        if (i<192)
        {
            colourTable[i][1]=(float)(i-64)/128.0;
        }
        else
        {
            colourTable[i][1]=0.0;
        }
    }


    if (i<128)
    {
        colourTable[i][2]=0.0;
        }
        else
        {
            if (i<256)
            {
                colourTable[i][2]=(float)(i-128)/128.0;
            }
            else
            {
                colourTable[i][2]=0.0;
            }
        }
    }
    define_color_table(display,0,256,colourTable);
#endif
    /* loop till file read */

    do
    {

        /* menu options */

        while ((iret=menu("File selection option:",menu1,3))==0);

        switch (iret)
        {

        case 1:                         /* specific filename */

            sel_file(FRAME_DIRECTORY,FRAME_PREFIX,FRAME_SUFFIX,filename);
```

```
        break;


    case 2:                         /* set name and number of sequence */

        printf("Input sequence name   : ");
        scanf("%s",sequence_name);
        do
        {
            printf("Input sequence number : ");
            scanf("%d",&sequence_num);
        } while (sequence_num<0 || sequence_num>999);

        strcpy(filename2,FRAME_DIRECTORY);    /* get bits of filename */
        strcat(filename2,FRAME_PREFIX);
        strcat(filename2,sequence_name);
        sprintf(string,".%3.3d",sequence_num);    /* get number as string */
        strcat(filename2,string);
        strcat(filename2,FRAME_SUFFIX);

        /* no break after case 2 so runs onto case 3 */

    case 3:                         /* next file in a sequence */

        strcpy(filename,filename2);
        sequence_num+=frame_increment;        /* increment count */

        /* test to see if next filename exists */

        strcpy(filename2,FRAME_DIRECTORY);    /* get bits of filename */
        strcat(filename2,FRAME_PREFIX);
        strcat(filename2,sequence_name);
        sprintf(string,".%3.3d",sequence_num);    /* get number as string */
        strcat(filename2,string);
        strcat(filename2,FRAME_SUFFIX);
        unit=fopen(filename2,"r");
        if (unit==NULL) printf("No subsequent files in sequence\n");
        else fclose(unit);

        break;
    }

    /* Open file and read image data */

    unit=fopen(filename,"r");
    if (unit==NULL) printf("File %s not found\n",filename);
} while (unit==NULL);

/* get x and y range of input file */
#if 0
    xrange=getc(unit);
    xrange=xrange+256*getc(unit);
    yrange=getc(unit);
    yrange=yrange+256*getc(unit);
#endif
    fread(&imageHeader,sizeof(imageHeader),1,unit);
    xrange=imageHeader.pixelsPerLine;
    yrange=imageHeader.numberOfLines;

    /* read data in */
#if 0
    nbytes=fread(buffer,xrange*yrange,1,unit);
```

```
    fclose(unit);
    if (nbytes==1)
    {
        printf("File %s read successfully\n",filename);
    }
    else
    {
        printf("Error reading file %s\n",filename);
    }

    /* sort to correct format */

    buf_pointer=buffer;
    ylow=0;
    for (iy=0;iy<yrange;iy++)
    {
        yhigh=((iy+1)*GRAPH_HEIGHT)/yrange;
        xlow=0;
        for (ix=0;ix<xrange;ix++)
        {
            byte=(*buf_pointer++);
            xhigh=((ix+1)*GRAPH_WIDTH)/xrange;
            for (y=ylow;y<yhigh;y++)
            {
                for (x=xlow;x<xhigh;x++)
                {
                    pixel_data[y][x]=byte;
                }
            }
            xlow=xhigh;
        }
        ylow=yhigh;
    }
#endif

    nbytes=fread(pixel_data,xrange*yrange,1,unit);
    fclose(unit);
    if (nbytes==1)
    {
        printf("File %s read successfully\n",filename);
    }
    else
    {
        printf("Error reading file %s\n",filename);
    }


    /* Write data to screen */
#if 0
    dcblock_write(display,SIDE_BORDER,TOP_BORDER,
        GRAPH_WIDTH,GRAPH_HEIGHT,pixel_data,FALSE);
    make_picture_current(display);
#endif
    dcblock_write(display,SIDE_BORDER,TOP_BORDER,
        xrange,yrange,pixel_data,FALSE);

    /* set up knob ranges and sensitivity */

    mapping_mode(knobs1,DISTORT);
    mapping_mode(knobs2,DISTORT);
    mapping_mode(knobs3,DISTORT);
    vdc_extent(knobs1,0.0,0.0,0.0,1.0,1.0,1.0);
```

```
    vdc_extent(knobs2,0.0,0.0,0.0,1.0,1.0,1.0);
    vdc_extent(knobs3,0.0,0.0,0.0,1.0,1.0,1.0);
    set_p1_p2(knobs1,FRACTIONAL,0.0,0.0,0.0,2.0,2.0,2.0);
    set_p1_p2(knobs2,FRACTIONAL,0.0,0.0,0.0,2.0,2.0,2.0);
    set_p1_p2(knobs3,FRACTIONAL,0.0,0.0,0.0,2.0,2.0,2.0);
    set_locator(knobs1,1,0.5,0.5,0.5);
    set_locator(knobs2,1,0.5,0.5,0.5);
    set_locator(knobs3,1,brightness,contrast,0.5);

    /* interactive contrast/brightness control */

    sample_locator(knobs3,1,&valid,&brightness,&contrast,&dummy);

    offset=(int)512.0*(0.5-brightness);
    range=512*(1.0-contrast);
    if (range==0) range=1;

    for (i=0;i<256;i++)
    {
        if (i<offset)
        {
            grey=0.0;
        }
        else
        {
            if (i<(offset+range))
            {
                grey=(float)(i-offset)/(float)range;
            }
            else
            {
                grey=1.0;
            }
        }
        colourTable[1][0]=grey;
        colourTable[1][1]=grey;
        colourTable[1][2]=grey;
    }
    define_color_table(display,0,256,colourTable);
    make_picture_current(display);

    /* Finish */

}
```

rotate3d()

```
/* Subroutine rotate3d produces rotation matrix around one of the axes */

#include "params.h"

void rotate3d(axis,angle,m)
char axis;
double angle;
float m[4][4];
{
    int top,bottom,left,right;

    identity3d(m);
    left  =(axis=='x'?1:0);
    right =(axis=='z'?1:2);
```

```
    top   =(axis=='x'?1:0);
    bottom=(axis=='z'?1:2);

    m[top][left]=(float)cos(angle);
    m[top][right]=(float)sin(angle);

    if (axis!='y')
        m[top][right]=(-m[top][right]);

    m[bottom][left]=(-m[top][right]);
    m[bottom][right]=m[top][left];
}
```

r_dynam()

```
#include "params.h"

/* routine to calculate linear dynamics */

void r_dynam(xaacc,yaacc,zaacc,nsegs,nframe,seg_moi,x_torque,y_torque,z_torque)

float xaacc[][MAX_FRAMES];        /* angular accn through x=0 plane */
float yaacc[][MAX_FRAMES];        /* angular accn through y=0 plane */
float zaacc[][MAX_FRAMES];        /* angular accn through z=0 plane */
int nsegs;                        /* number of segments */
int nframe;                        /* number of frames */
float seg_moi[];                    /* segment moments of inertia */
float x_torque[][MAX_FRAMES];    /* x component of torque */
float y_torque[][MAX_FRAMES];    /* y component of torque */
float z_torque[][MAX_FRAMES];    /* z component of torque */
{
    int isegs;              /* segment counter */
    int iframe;              /* frame counter */

    /* loop round frames and segments */

    for (isegs=0;isegs<nsegs;isegs++)
    {
        for (iframe=0;iframe<nframe;iframe++)
        {
            x_torque[isegs][iframe]=seg_moi[isegs]*xaacc[isegs][iframe];
            y_torque[isegs][iframe]=seg_moi[isegs]*yaacc[isegs][iframe];
            z_torque[isegs][iframe]=seg_moi[isegs]*zaacc[isegs][iframe];
        }
    }
}
```

r_kinem()

```
#include "params.h"

void r_kinem(xpos,ypos,zpos,nframe,fspeed,nsegs,nodes_per_seg,times,
    xapos,yapos,zapos,
    xavel,yavel,zavel,xaacc,yaacc,zaacc)

/* this routine calculates the rotational kinematic parameters */

float xpos[][MAX_FRAMES];        /* x world coordinates (m) */
float ypos[][MAX_FRAMES];        /* y world coordinates (m) */
```

```
float zpos[][MAX_FRAMES];        /* z world coordinates (m) */
int nframe;                       /* number of frames */
float fspeed;                       /* Film frame interval (s) */
int nsegs;                       /* number of segments */
int nodes_per_seg[NPS][MAX_SEGS];    /* nodes per segment */
float times[MAX_FRAMES];           /* times (s) */
float xapos[][MAX_FRAMES];       /* angle of segment (radian) */
float yapos[][MAX_FRAMES];
float zapos[][MAX_FRAMES];
float xavel[][MAX_FRAMES];       /* calculated angular velocities (rad/s) */
float yavel[][MAX_FRAMES];
float zavel[][MAX_FRAMES];
float xaacc[][MAX_FRAMES];       /* calculated angular accelerations (rad/s/s) */
float yaacc[][MAX_FRAMES];
float zaacc[][MAX_FRAMES];

{
    int iframe;                  /* frame counter */
    int iseg;                      /* segment counter */
    float time;                  /* time counter (s) */
    double xseg,yseg,zseg;     /* segment vector */

    /* check sufficient frames */

    if (nframe<5)
    {
        printf("Insufficient frames\n");
        return;
    }

    /* loop over segments */

    for (iseg=0;iseg<nsegs;iseg++)
    {

        /* calculate angles */

        for (iframe=0;iframe<nframe;iframe++)
        {
            xseg=(double)(xpos[nodes_per_seg[1][iseg]][iframe]-
                xpos[nodes_per_seg[0][iseg]][iframe]);
            yseg=(double)(ypos[nodes_per_seg[1][iseg]][iframe]-
                ypos[nodes_per_seg[0][iseg]][iframe]);
            zseg=(double)(zpos[nodes_per_seg[1][iseg]][iframe]-
                zpos[nodes_per_seg[0][iseg]][iframe]);

            xapos[iseg][iframe]=(float)atan2(zseg,yseg);
            yapos[iseg][iframe]=(float)atan2(xseg,zseg);
            zapos[iseg][iframe]=(float)atan2(yseg,xseg);
        }

        /* calculate angular velocities and accelerations */

        different(times,xapos[iseg],nframe,xavel[iseg],xaacc[iseg],TRUE);
        different(times,yapos[iseg],nframe,yavel[iseg],yaacc[iseg],TRUE);
        different(times,zapos[iseg],nframe,zavel[iseg],zaacc[iseg],TRUE);
    }
}
```

save_an()

```
#include "params.h"
```

```
void save_an(title,x_label,y_label,x_point,y_point,npoint,nline,key)

/* this routine saves the data contained in x_point and y_point to a file */

char title[STRING_SIZE];                  /* graph title */
char x_label[STRING_SIZE];               /* x axis label */
char y_label[STRING_SIZE];               /* y axis label */
float x_point[MAX_LINES][MAX_POINTS];     /* x coordinates */
float y_point[MAX_LINES][MAX_POINTS];     /* y coordinates */
int npoint;                               /* number of points */
int nline;                                /* number of lines */
char key[MAX_LINES][STRING_SIZE];        /* key for multiple lines */

{
    FILE *unit;                          /* file pointer */
    char fname[STRING_SIZE];              /* filename */
    char filename[STRING_SIZE];          /* full filename */
    int iline;                           /* line counter */
    int ipoint;                          /* point counter */
    int icount;                          /* character counter */
    int iret;                            /* menu option */
    static char menu1[][STRING_SIZE]=    /* output selection menu */
    {
        "ASCII file for 123",
        "ASCII file for EXCEL",
        "SAS Program File",
        "Exit"
    };

do
{

    /* select menu option */

    while ((iret=menu("Select file type:",menu1,4))==0);

    switch (iret)
    {

    case 1:

        do
        {

            /* get filename */

            printf("Input 123 data file name : ");
            scanf("%s",fname);
            strcpy(filename,ANALYSIS_DIRECTORY);
            strcat(filename,ANALYSIS_PREFIX);
            strcat(filename,fname);
            strcat(filename,ANALYSIS_SUFFIX_123);

            /* open file */

            unit=fopen(filename,"w");
        } while (unit==NULL);

        /* write out data in ASCII form suitable for LOTUS 123 import */

        fprintf(unit,"\"%s\"\015\012",title);
```

```
for (iline=0;iline<nline;iline++)
    fprintf(unit,"\"%s\" \" \" ",key[iline]);
fprintf(unit,"\015\012");
for (iline=0;iline<nline;iline++)
    fprintf(unit,"\"%s\" \"%s\" ",x_label,y_label);
fprintf(unit,"\015\012");
for (ipoint=0;ipoint<npoint;ipoint++)
{
    for (iline=0;iline<nline;iline++)
        fprintf(unit,"%12.5e %12.5e ",x_point[iline][ipoint],
            y_point[iline][ipoint]);
    fprintf(unit,"\015\012");
}

fclose(unit);

    /* print success message */

    printf("File %s written successfully\n",filename);

    /* finished */

    break;

case 2:

    do
    {

        /* get filename */

        printf("Input EXCEL data file name : ");
        scanf("%s",fname);
        strcpy(filename,ANALYSIS_DIRECTORY);
        strcat(filename,ANALYSIS_PREFIX);
        strcat(filename,fname);
        strcat(filename,ANALYSIS_SUFFIX_EXCEL);

        /* open file */

        unit=fopen(filename,"w");
} while (unit==NULL);

/* write out data in ASCII form suitable for EXCEL import */

fprintf(unit,"%s\015",title);
for (iline=0;iline<nline;iline++)
    fprintf(unit,"%s\011\011",key[iline]);
fprintf(unit,"\015");
for (iline=0;iline<nline;iline++)
    fprintf(unit,"%s\011%s\011",x_label,y_label);
fprintf(unit,"\015");
for (ipoint=0;ipoint<npoint;ipoint++)
{
    for (iline=0;iline<nline;iline++)
        fprintf(unit,"%12.5e\011%12.5e\011",
            x_point[iline][ipoint],
            y_point[iline][ipoint]);
    fprintf(unit,"\015");
}

fclose(unit);
```

```
        /* print success message */

        printf("File %s written successfully\n",filename);

        /* finished */

        break;

    case 3:

        do
        {

            /* get filename */

            printf("Input SAS program file name : ");
            scanf("%s",fname);
            strcpy(filename,ANALYSIS_DIRECTORY);
            strcat(filename,ANALYSIS_PREFIX);
            strcat(filename,fname);
            strcat(filename,ANALYSIS_SUFFIX_SAS);

            /* open file */

            unit=fopen(filename,"w");
        } while (unit==NULL);

        /* write out SAS program in ASCII file */

        fprintf(unit,"/* %s */\n\n",title);
        for (iline=0;iline<nline;iline++)
            fprintf(unit,"/* col%02d - %s */\n",iline,key[iline]);
        fprintf(unit,"\n");
        fprintf(unit,"/* x label - %s */\n/* y label - %s */\n\n",
            x_label,y_label);
        fprintf(unit,"data gait;\ninput\n");
        for (iline=0;iline<nline;iline++)
            fprintf(unit,"col%02d_x@@ col%02d_y@@\n",iline,iline);
        fprintf(unit,";\ncards;\n");
        for (ipoint=0;ipoint<npoint;ipoint++)
        {
            for (iline=0;iline<nline;iline++)
                fprintf(unit,"%10.3e %10.3e\n",x_point[iline][ipoint],
                    y_point[iline][ipoint]);
        }
        fprintf(unit,";\nrun;\n");

        fprintf(unit,"proc gplot;\n");
        fprintf(unit,"axis1 label=(f=swiss j=c '%s')\nvalue=(f=simplex);\n",
            x_label);
        fprintf(unit,"axis2 label=(f=swiss j=c '%s')\nvalue=(f=simplex);\n",
            y_label);
        fprintf(unit,"plot\n");
        for (iline=0;iline<nline;iline++)
            fprintf(unit,"col%02d_y * col%02d_x\n",iline,iline);
        fprintf(unit,"/overlay haxis=axis1 vaxis=axis2;\n");
        fprintf(unit,"title f=centb '%s';\n",title);
        for (iline=0;iline<nline;iline++)
        {
            fprintf(unit,"symbol%-3d f=simplex i=join v='%d';\n",iline+1,iline);
            fprintf(unit,"footnote%-3d f=simplex j=l '%3d = %s';\n",
```

```
                iline+1,iline,key[iline]);
        }
        fprintf(unit,"run;\n");

        fclose(unit);

        /* print success message */

        printf("File %s written successfully\n",filename);

        /* finished */

        break;
      }
    }    while (iret!=4);
}
```

seg_lengths()

```
/* routine to write out lengths file for EXCEL */

#include "params.h"

void seg_lengths(segs,nsegs,seg_length)
char segs[MAX_SEGS][STRING_SIZE];
int nsegs;
float seg_length[MAX_SEGS];
{
    FILE *unit;             /* file unit */
    char filename[STRING_SIZE];    /* file name */
    char fname[STRING_SIZE];    /* intermediate file name */
    int isegs;              /* segment counter */

    do
    {
        /* get filename */

        printf("Input EXCEL data file name : ");
        scanf("%s",fname);
        strcpy(filename,ANALYSIS_DIRECTORY);
        strcat(filename,ANALYSIS_PREFIX);
        strcat(filename,fname);
        strcat(filename,ANALYSIS_SUFFIX_EXCEL);

        /* open file */

        unit=fopen(filename,"w");
    } while (unit==NULL);

    /* write out data */

    for (isegs=0;isegs<nsegs;isegs++)
    {
        fprintf(unit,"%s\011%e\015\012",segs[isegs],seg_length[isegs]);
    }
    fclose(unit);

    printf("File %s written successfully\n",filename);

}
```

seg_plot()

```
#include "params.h"

void seg_plot(xapos,yapos,zapos,startframe,endframe,segs,nsegs,times,
    p_title,p_ylabel)

/* this routine displays the segment angles/angular velocities/angular accelerations */
/* depending on the data arrays passed to the routine */

float xapos[][MAX_FRAMES];        /* segment angles/a vel/a acc */
float yapos[][MAX_FRAMES];
float zapos[][MAX_FRAMES];
int startframe;                    /* starting frame number */
int endframe;                      /* end frame number */
char segs[MAX_SEGS][STRING_SIZE];  /* Names of segments */
int nsegs;                         /* Number of segments */
float times[MAX_FRAMES];           /* times (s) */
char p_title[STRING_SIZE];         /* plot title */
char p_ylabel[STRING_SIZE];        /* plot y axis label */

{
    float pl_times[MAX_LINES][MAX_POINTS];   /* times to be plotted (s) */
    float angle[MAX_LINES][MAX_POINTS];      /* calculated angles/a vels/a accs */
    char key[MAX_LINES+1][STRING_SIZE];      /* graph key string */
    char title[STRING_SIZE];                 /* graph title */
    char x_label[STRING_SIZE];               /* graph x axis label */
    char y_label[STRING_SIZE];               /* graph y axis label */
    char fname[STRING_SIZE];                 /* picture filename */
    char filename[STRING_SIZE];              /* full picture filename */
    float xref=0.0,yref=0.0,zref=0.0;        /* reference angles */
    int iframe;                              /* frame counter */
    int iabs=1;                              /* relative/absolute indicator */
    int idir;                                /* direction indicator */
    int iplot;                               /* plotter control */
    int nlines;                              /* number of lines on graph */
    int aseg;                                /* start segment number */
    int bseg;                                /* finish segment number */

    static char menu1[][STRING_SIZE]=        /* absolute or relative menu */
    {
        "Absolute",
        "Relative"
    };
    static char menu2[][STRING_SIZE]=        /* angle menu */
    {
        "X=0",
        "Y=0",
        "Z=0",
    };
    static char menu3[][STRING_SIZE]=        /* plotting menu */
    {
        "Save data to file",
        "Save picture to file",
        "Exit"
    };

    /* all segments */

    if (nsegs<MAX_LINES && yesno("Plot all segments?")==TRUE)
    {
```

```
        if (flag_2d==TRUE)
        {
            idir=3;
            key[nsegs][0]=0;
        }
        else
        {
            while ((idir=menu("Angle plane:",menu2,3))==0);
            strcpy(key[nsegs],menu2[idir-1]);
            strcat(key[nsegs]," ");
        }

        for (nlines=0;nlines<nsegs;nlines++)
        {
            strcpy(key[nlines],key[nsegs]);
            strcat(key[nlines],segs[nlines]);

            /* loop round frames */

            for (iframe=startframe;iframe<endframe;iframe++)
            {

                /* calculate times */

                pl_times[nlines][iframe-startframe]=times[iframe];

                /* calculate angle */

                switch (idir)
                {
                case 1:
                    angle[nlines][iframe-startframe]=
                        xapos[nlines][iframe];
                    break;
                case 2:
                    angle[nlines][iframe-startframe]=
                        yapos[nlines][iframe];
                    break;
                case 3:
                    angle[nlines][iframe-startframe]=
                        zapos[nlines][iframe];
                    break;
                }
            }
        }

    }
    else
    {

        /* first option */

        while ((iabs=menu("Option:",menu1,2))==0);

        /* loop round number of lines */

        nlines=0;
        do
        {

            /* second option */
```

```
        if (flag_2d==TRUE)
        {
            idir=3;
            key[nlines][0]=0;
        }
        else
        {
            while ((idir=menu("Angle plane:",menu2,3))==0);
            strcpy(key[nlines],menu2[idir-1]);
            strcat(key[nlines]," ");
        }

        if (iabs==2)
        {
            while ((aseg=menu("Select reference segment:",segs,nsegs))==0);
            aseg--;
            strcat(key[nlines],segs[aseg]);
            strcat(key[nlines]," ");
        }
        while ((bseg=menu("Select segment:",segs,nsegs))==0);
        bseg--;
        strcat(key[nlines],segs[bseg]);

        /* loop round frames */

        for (iframe=startframe;iframe<endframe;iframe++)
        {

            /* calculate times */

            pl_times[nlines][iframe-startframe]=times[iframe];

            /* calculate angle */

            if (iabs==2)
            {
                xref=xapos[aseg][iframe];
                yref=yapos[aseg][iframe];
                zref=zapos[aseg][iframe];
            }

            switch (idir)
            {
            case 1:
                angle[nlines][iframe-startframe]=xapos[bseg][iframe]-xref;
                break;
            case 2:
                angle[nlines][iframe-startframe]=yapos[bseg][iframe]-yref;
                break;
            case 3:
                angle[nlines][iframe-startframe]=zapos[bseg][iframe]-zref;
                break;
            }
        }
        nlines++;
    } while (nlines<MAX_LINES && yesno("Another line?")==TRUE);
}

/* draw graph */

strcpy(title,menu1[iabs-1]);
```

```
    strcat(title," ");
    strcat(title,p_title);
    strcpy(x_label,"Time (s)");
    strcpy(y_label,p_ylabel);
    d_graph(display,title,x_label,y_label,pl_times,angle,
        endframe-startframe,nlines,key,FALSE);
    while ((iplot=menu("Select option:",menu3,3))!=3)
    {
        switch (iplot)
        {
        case 1:
            save_an(title,x_label,y_label,pl_times,angle,
                endframe-startframe,nlines,key);
            break;
        case 2:
            printf("Input picture file name : ");
            scanf("%s",fname);
            strcpy(filename,PICTURE_DIRECTORY);
            strcat(filename,PICTURE_PREFIX);
            strcat(filename,fname);
            strcat(filename,PICTURE_SUFFIX);

            bitmap_to_file(display,TRUE,0,0,filename,TRUE,0.0,0.0,0,0,TRUE);
            break;
        }
    }
    CLEAR_GRAPH;

    /* finished */

}
```

sel_file()

```
#include "params.h"

int sel_file(directory,prefix,suffix,filename)

/* this routine allows the user to select a file from a menu list */

char directory[];                /* directory path */
char prefix[];                   /* selection prefix */
char suffix[];                   /* selection suffix */
char filename[STRING_SIZE];           /* returned filename */

{
    FILE *unit;                 /* temporary directory file */
    int error=0;                 /* error return value */
    int nentries=0;                 /* number of directory entries */
    int iret=0;                 /* menu return value */
    int dir_len;                 /* length of directory string */
    int ientries;                 /* directory entry counter */
    char command[STRING_SIZE];        /* system command string */
    char dir_list[DIRECTORY_ENTRIES][STRING_SIZE];    /* directory entries */

    /* make up system command */

    strcpy(command,"ls -l ");
    strcat(command,directory);
    strcat(command,prefix);
    strcat(command,"*");
```

```
    strcat(command,suffix);
    strcat(command," > directory_log");

    /* perform command */

    system(command);

    /* read in directory file */

    unit=fopen("directory_log","r");
    if (unit!=NULL)
    {
        while (fscanf(unit,"%s\n",dir_list[nentries])!=EOF) nentries++;
        fclose(unit);
    }

    /* see if entries exist */

    if (nentries==0)
    {
        printf("Input filename : ");
        scanf("%s",filename);
        unit=fopen(filename,"r");
        if (unit==NULL) error=1;
        else fclose(unit);
    }
    else
    {

        /* remove directory data from list */

        dir_len=strlen(directory);
        for (ientries=0;ientries<nentries;ientries++)
        {
            strcpy(command,dir_list[ientries]+dir_len);
            strcpy(dir_list[ientries],command);
        }


        /* do menu */

        while (iret==0)
            iret=menu("Select file:",dir_list,nentries);
        strcpy(filename,directory);
        strcat(filename,dir_list[iret-1]);
    }

    /* finished */

    return(error);

}
```

simple_recon()

```
/* This routine uses the orthogonal camera assumptions to reconstruct the (x,y,z)
coordinates */
/* from two sets of screen coordinates */

#include "params.h"
```

```
void simple_recon(qa,ra,qb,rb,x,y,z)
float qa,ra;                    /* screen 1 (q,r) coordinates */
float qb,rb;                    /* screen 2 (q,r) coordinates */
float *x,*y,*z;                /* reconstructed world coordinates */
{

    /* PICTURE 1 */

    /* correct for x mirroring */

    if (x_mirror_1) qa=(-qa);

    /* correct for offset and scale factor */

    qa=scale_factor_1*(qa-x_offset_1);
    ra=scale_factor_1*(ra-y_offset_1);

    /* correct for rotation */

    qa=rotation_1[0][0]*qa+rotation_1[0][1]*ra;
    ra=rotation_1[1][0]*qa+rotation_1[1][1]*ra;

    /* PICTURE 2 */

    /* correct for x mirroring */

    if (x_mirror_2) qb=(-qb);

    /* correct for offset and scale factor */

    qb=scale_factor_2*(qb-x_offset_2);
    rb=scale_factor_2*(rb-y_offset_2);

    /* correct for rotation */

    qb=rotation_2[0][0]*qb+rotation_2[0][1]*rb;
    rb=rotation_2[1][0]*qb+rotation_2[1][1]*rb;

    if (x_axis_source==1) *x=qa;
    else
    {
        if (x_axis_source==2) *x=ra;
        else
        {
            if (x_axis_source==3) *x=qb;
            else *x=rb;
        }
    }

    if (y_axis_source==1) *y=qa;
    else
    {
        if (y_axis_source==2) *y=ra;
        else
        {
            if (y_axis_source==3) *y=qb;
            else *y=rb;
        }
    }

    if (z_axis_source==1) *z=qa;
    else
```

```
    {
        if (z_axis_source==2) *z=ra;
        else
        {
            if (z_axis_source==3) *z=qb;
            else *z=rb;
        }
    }
}
```

simplified_quadrupedal()

```
#include "params.h"

/* routine to calculate joint torques and reaction forces for simplified */
/* quadrupedal leaping model */

simplified_quadrupedal(seg_length,seg_mass,seg_com,phi,x_force,y_force,torque,
    segs,nsegs,nframe,times)

float seg_length[MAX_SEGS];            /* mean segment lengths */
float seg_mass[MAX_SEGS];             /* segment masses */
float seg_com[MAX_SEGS];               /* relative COM positions */
float phi[MAX_SEGS][MAX_FRAMES];     /* segment angle */
float x_force[MAX_SEGS][MAX_FRAMES];    /* x component of linear force */
float y_force[MAX_SEGS][MAX_FRAMES];    /* y component of linear force */
float torque[MAX_SEGS][MAX_FRAMES];    /* torques about segments */
char segs[MAX_SEGS][STRING_SIZE];    /* segment names */
int nsegs;                             /* number of segments */
int nframe;                            /* number of frames */
float times[MAX_FRAMES];               /* frame times */
{
    int iframe;                        /* frame counter */
    float d1,d2;                       /* distances from COM */
    struct body                        /* structure for free body data */
    {
        float r1x[MAX_FRAMES];    /* reaction force at 'proximal' end */
        float r1y[MAX_FRAMES];
        float r2x[MAX_FRAMES];    /* reaction forces at 'distal' end */
        float r2y[MAX_FRAMES];
        float m1[MAX_FRAMES];    /* torque at 'proximal' end */
        float m2[MAX_FRAMES];    /* torque at 'distal' end */
        float m1_0[MAX_FRAMES];    /* torque assuming 0 contact torque */
        float m2_0[MAX_FRAMES];    /* torque assuming 0 contact torque */
        int segnum;                /* segment number */
    };
    struct body lower_arm;         /* segment structures */
    struct body upper_arm;
    struct body forefoot;
    struct body hindfoot;
    struct body calf;
    struct body thigh;
    struct body head;
    struct body torso;
    struct body tail;
    float x_react[MAX_SEGS][MAX_FRAMES];    /* reaction forces */
    float y_react[MAX_SEGS][MAX_FRAMES];    /* reaction forces */
    float j_torque[MAX_SEGS][MAX_FRAMES];    /* joint torques */
    float j_torque_0[MAX_SEGS][MAX_FRAMES];    /* joint torques with zero contact torque
*/
    float work_done[MAX_SEGS][MAX_FRAMES];    /* work done at joints */
```

```
float work_done_0[MAX_SEGS][MAX_FRAMES];/* WD at joints for 0 contact torque*/
float dummy[MAX_SEGS][MAX_FRAMES];        /* dummy values for plots */
float angle;                             /* intermediate angle value */
static char joints[][STRING_SIZE]=        /* joints etc. */
{
    "Contact point",
    "Mid-tarsal joint",
    "Ankle",
    "Knee",
    "Hip",
    "Tail base",
    "Neck",
    "Shoulder",
    "Elbow"
};
int njoints=9;                          /* number of joints */
int njoints_0=5;                         /* number of joint for zero torque */
static char menu1[][STRING_SIZE]=        /* printout menu */
{
    "Joint reactions",
    "Joint torques",
    "Zero contact joint torques",
    "Work done per frame",
    "Work done - zero contact torque",
    "Exit"
};
int nmenu1=6;                            /* number of menu items */
int iret;                                /* menu return value */

/* initialize segment ID numbers */

lower_arm.segnum=0;
upper_arm.segnum=1;
forefoot.segnum=2;
hindfoot.segnum=3;
calf.segnum=4;
thigh.segnum=5;
head.segnum=6;
torso.segnum=7;
tail.segnum=8;

/* loop over frames */

for (iframe=1;iframe<(nframe-1);iframe++)
{
    d1=seg_com[lower_arm.segnum]*seg_length[lower_arm.segnum];
    d2=(1-seg_com[lower_arm.segnum])*seg_length[lower_arm.segnum];
    lower_arm.r1x[iframe]=0.0;
    lower_arm.r1y[iframe]=0.0;
    lower_arm.m1[iframe]=0.0;
    lower_arm.r2x[iframe]=x_force[lower_arm.segnum][iframe]-
        lower_arm.r1x[iframe];
    lower_arm.r2y[iframe]=y_force[lower_arm.segnum][iframe]-
        lower_arm.r1y[iframe]-seg_mass[lower_arm.segnum]*G;
    lower_arm.m2[iframe]=torque[lower_arm.segnum][iframe]-
        lower_arm.r1x[iframe]*d1*
        (float)sin((double)phi[lower_arm.segnum][iframe])+
        lower_arm.r2x[iframe]*d2*
        (float)sin((double)phi[lower_arm.segnum][iframe])+
        lower_arm.r1y[iframe]*d1*
        (float)cos((double)phi[lower_arm.segnum][iframe])-
        lower_arm.r2y[iframe]*d2*
```

```
        (float)cos((double)phi[lower_arm.segnum][iframe])-
        lower_arm.m1[iframe];

    d1=seg_com[upper_arm.segnum]*seg_length[upper_arm.segnum];
    d2=(1-seg_com[upper_arm.segnum])*seg_length[upper_arm.segnum];
    upper_arm.r1x[iframe]=(-lower_arm.r2x[iframe]);
    upper_arm.r1y[iframe]=(-lower_arm.r2y[iframe]);
    upper_arm.m1[iframe]=(-lower_arm.m2[iframe]);
    upper_arm.r2x[iframe]=x_force[upper_arm.segnum][iframe]-
        upper_arm.r1x[iframe];
    upper_arm.r2y[iframe]=y_force[upper_arm.segnum][iframe]-
        upper_arm.r1y[iframe]-seg_mass[upper_arm.segnum]*G;
    upper_arm.m2[iframe]=torque[upper_arm.segnum][iframe]-
        upper_arm.r1x[iframe]*d1*
        (float)sin((double)phi[upper_arm.segnum][iframe])+
        upper_arm.r2x[iframe]*d2*
        (float)sin((double)phi[upper_arm.segnum][iframe])+
        upper_arm.r1y[iframe]*d1*
        (float)cos((double)phi[upper_arm.segnum][iframe])-
        upper_arm.r2y[iframe]*d2*
        (float)cos((double)phi[upper_arm.segnum][iframe])-
        upper_arm.m1[iframe];

    d1=seg_com[head.segnum]*seg_length[head.segnum];
    d2=(1-seg_com[head.segnum])*seg_length[head.segnum];
    head.r1x[iframe]=0.0;
    head.r1y[iframe]=0.0;
    head.m1[iframe]=0.0;
    head.r2x[iframe]=x_force[head.segnum][iframe]-head.r1x[iframe];
    head.r2y[iframe]=y_force[head.segnum][iframe]-head.r1y[iframe]-
        seg_mass[head.segnum]*G;
    head.m2[iframe]=torque[head.segnum][iframe]-
        head.r1x[iframe]*d1*
        (float)sin((double)phi[head.segnum][iframe])+
        head.r2x[iframe]*d2*
        (float)sin((double)phi[head.segnum][iframe])+
        head.r1y[iframe]*d1*
        (float)cos((double)phi[head.segnum][iframe])-
        head.r2y[iframe]*d2*
        (float)cos((double)phi[head.segnum][iframe])-
        head.m1[iframe];

    d1=seg_com[tail.segnum]*seg_length[tail.segnum];
    d2=(1-seg_com[tail.segnum])*seg_length[tail.segnum];
    tail.r2x[iframe]=0.0;
    tail.r2y[iframe]=0.0;
    tail.m2[iframe]=0.0;
    tail.r1x[iframe]=x_force[tail.segnum][iframe]-tail.r2x[iframe];
    tail.r1y[iframe]=y_force[tail.segnum][iframe]-tail.r2y[iframe]-
        seg_mass[tail.segnum]*G;
    tail.m1[iframe]=torque[tail.segnum][iframe]-
        tail.r1x[iframe]*d1*
        (float)sin((double)phi[tail.segnum][iframe])+
        tail.r2x[iframe]*d2*
        (float)sin((double)phi[tail.segnum][iframe])+
        tail.r1y[iframe]*d1*
        (float)cos((double)phi[tail.segnum][iframe])-
        tail.r2y[iframe]*d2*
        (float)cos((double)phi[tail.segnum][iframe])-
        tail.m2[iframe];

    d1=seg_com[torso.segnum]*seg_length[torso.segnum];
```

```
d2=(1-seg_com[torso.segnum])*seg_length[torso.segnum];
torso.r1x[iframe]=(-head.r2x[iframe])+(-upper_arm.r2x[iframe]);
torso.r1y[iframe]=(-head.r2y[iframe])+(-upper_arm.r2y[iframe]);
torso.m1[iframe]=(-head.m2[iframe])+(-upper_arm.m2[iframe]);
torso.r2x[iframe]=x_force[torso.segnum][iframe]-torso.r1x[iframe];
torso.r2y[iframe]=y_force[torso.segnum][iframe]-torso.r1y[iframe]-
    seg_mass[torso.segnum]*G;
torso.m2[iframe]=torque[torso.segnum][iframe]-
    torso.r1x[iframe]*d1*
    (float)sin((double)phi[torso.segnum][iframe])+
    torso.r2x[iframe]*d2*
    (float)sin((double)phi[torso.segnum][iframe])+
    torso.r1y[iframe]*d1*
    (float)cos((double)phi[torso.segnum][iframe])-
    torso.r2y[iframe]*d2*
    (float)cos((double)phi[torso.segnum][iframe])-
    torso.m1[iframe];

d1=seg_com[thigh.segnum]*seg_length[thigh.segnum];
d2=(1-seg_com[thigh.segnum])*seg_length[thigh.segnum];
thigh.r2x[iframe]=(-torso.r2x[iframe])+(-tail.r1x[iframe]);
thigh.r2y[iframe]=(-torso.r2y[iframe])+(-tail.r1y[iframe]);
thigh.m2[iframe]=(-torso.m2[iframe])+(-tail.m1[iframe]);
thigh.r1x[iframe]=x_force[thigh.segnum][iframe]-thigh.r2x[iframe];
thigh.r1y[iframe]=y_force[thigh.segnum][iframe]-thigh.r2y[iframe]-
    seg_mass[thigh.segnum]*G;
thigh.m1[iframe]=torque[thigh.segnum][iframe]-
    thigh.r1x[iframe]*d1*
    (float)sin((double)phi[thigh.segnum][iframe])+
    thigh.r2x[iframe]*d2*
    (float)sin((double)phi[thigh.segnum][iframe])+
    thigh.r1y[iframe]*d1*
    (float)cos((double)phi[thigh.segnum][iframe])-
    thigh.r2y[iframe]*d2*
    (float)cos((double)phi[thigh.segnum][iframe])-
    thigh.m2[iframe];

d1=seg_com[calf.segnum]*seg_length[calf.segnum];
d2=(1-seg_com[calf.segnum])*seg_length[calf.segnum];
calf.r2x[iframe]=(-thigh.r1x[iframe]);
calf.r2y[iframe]=(-thigh.r1y[iframe]);
calf.m2[iframe]=(-thigh.m1[iframe]);
calf.r1x[iframe]=x_force[calf.segnum][iframe]-calf.r2x[iframe];
calf.r1y[iframe]=y_force[calf.segnum][iframe]-calf.r2y[iframe]-
    seg_mass[calf.segnum]*G;
calf.m1[iframe]=torque[calf.segnum][iframe]-
    calf.r1x[iframe]*d1*
    (float)sin((double)phi[calf.segnum][iframe])+
    calf.r2x[iframe]*d2*
    (float)sin((double)phi[calf.segnum][iframe])+
    calf.r1y[iframe]*d1*
    (float)cos((double)phi[calf.segnum][iframe])-
    calf.r2y[iframe]*d2*
    (float)cos((double)phi[calf.segnum][iframe])-
    calf.m2[iframe];

d1=seg_com[hindfoot.segnum]*seg_length[hindfoot.segnum];
d2=(1-seg_com[hindfoot.segnum])*seg_length[hindfoot.segnum];
hindfoot.r2x[iframe]=(-calf.r1x[iframe]);
hindfoot.r2y[iframe]=(-calf.r1y[iframe]);
hindfoot.m2[iframe]=(-calf.m1[iframe]);
hindfoot.r1x[iframe]=x_force[hindfoot.segnum][iframe]-hindfoot.r2x[iframe];
```

```
hindfoot.r1y[iframe]=y_force[hindfoot.segnum][iframe]-hindfoot.r2y[iframe]-
    seg_mass[hindfoot.segnum]*G;
hindfoot.m1[iframe]=torque[hindfoot.segnum][iframe]-
    hindfoot.r1x[iframe]*d1*
    (float)sin((double)phi[hindfoot.segnum][iframe])+
    hindfoot.r2x[iframe]*d2*
    (float)sin((double)phi[hindfoot.segnum][iframe])+
    hindfoot.r1y[iframe]*d1*
    (float)cos((double)phi[hindfoot.segnum][iframe])-
    hindfoot.r2y[iframe]*d2*
    (float)cos((double)phi[hindfoot.segnum][iframe])-
    hindfoot.m2[iframe];

d1=seg_com[forefoot.segnum]*seg_length[forefoot.segnum];
d2=(1-seg_com[forefoot.segnum])*seg_length[forefoot.segnum];
forefoot.r2x[iframe]=(-hindfoot.r1x[iframe]);
forefoot.r2y[iframe]=(-hindfoot.r1y[iframe]);
forefoot.m2[iframe]=(-hindfoot.m1[iframe]);
forefoot.r1x[iframe]=x_force[forefoot.segnum][iframe]-forefoot.r2x[iframe];
forefoot.r1y[iframe]=y_force[forefoot.segnum][iframe]-forefoot.r2y[iframe]-
    seg_mass[forefoot.segnum]*G;
forefoot.m1[iframe]=torque[forefoot.segnum][iframe]-
    forefoot.r1x[iframe]*d1*
    (float)sin((double)phi[forefoot.segnum][iframe])+
    forefoot.r2x[iframe]*d2*
    (float)sin((double)phi[forefoot.segnum][iframe])+
    forefoot.r1y[iframe]*d1*
    (float)cos((double)phi[forefoot.segnum][iframe])-
    forefoot.r2y[iframe]*d2*
    (float)cos((double)phi[forefoot.segnum][iframe])-
    forefoot.m2[iframe];

/* calculate joint torques assumming zero torque at contact with */
/* ground */

forefoot.m1_0[iframe]=0.0;
forefoot.m2_0[iframe]=torque[forefoot.segnum][iframe]-
    forefoot.r1x[iframe]*d1*
    (float)sin((double)phi[forefoot.segnum][iframe])+
    forefoot.r2x[iframe]*d2*
    (float)sin((double)phi[forefoot.segnum][iframe])+
    forefoot.r1y[iframe]*d1*
    (float)cos((double)phi[forefoot.segnum][iframe])-
    forefoot.r2y[iframe]*d2*
    (float)cos((double)phi[forefoot.segnum][iframe])-
    forefoot.m1_0[iframe];

hindfoot.m1_0[iframe]=(-forefoot.m2_0[iframe]);
hindfoot.m2_0[iframe]=torque[hindfoot.segnum][iframe]-
    hindfoot.r1x[iframe]*d1*
    (float)sin((double)phi[hindfoot.segnum][iframe])+
    hindfoot.r2x[iframe]*d2*
    (float)sin((double)phi[hindfoot.segnum][iframe])+
    hindfoot.r1y[iframe]*d1*
    (float)cos((double)phi[hindfoot.segnum][iframe])-
    hindfoot.r2y[iframe]*d2*
    (float)cos((double)phi[hindfoot.segnum][iframe])-
    hindfoot.m1_0[iframe];

calf.m1_0[iframe]=(-hindfoot.m2_0[iframe]);
calf.m2_0[iframe]=torque[calf.segnum][iframe]-
    calf.r1x[iframe]*d1*
```

```
        (float)sin((double)phi[calf.segnum][iframe])+
        calf.r2x[iframe]*d2*
        (float)sin((double)phi[calf.segnum][iframe])+
        calf.r1y[iframe]*d1*
        (float)cos((double)phi[calf.segnum][iframe])-
        calf.r2y[iframe]*d2*
        (float)cos((double)phi[calf.segnum][iframe])-
        calf.m1_0[iframe];

    thigh.m1_0[iframe]=(-calf.m2_0[iframe]);
    thigh.m2_0[iframe]=torque[thigh.segnum][iframe]-
        thigh.r1x[iframe]*d1*
        (float)sin((double)phi[thigh.segnum][iframe])+
        thigh.r2x[iframe]*d2*
        (float)sin((double)phi[thigh.segnum][iframe])+
        thigh.r1y[iframe]*d1*
        (float)cos((double)phi[thigh.segnum][iframe])-
        thigh.r2y[iframe]*d2*
        (float)cos((double)phi[thigh.segnum][iframe])-
        thigh.m1_0[iframe];

    /* put into array for display */

    x_react[0][iframe]=(-forefoot.r1x[iframe]);    /* contact point */
    y_react[0][iframe]=(-forefoot.r1y[iframe]);
    j_torque[0][iframe]=(-forefoot.m1[iframe]);
    j_torque_0[0][iframe]=(-forefoot.m1_0[iframe]);
    angle=phi[forefoot.segnum][iframe+1]-phi[forefoot.segnum][iframe-1];
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[0][iframe]=j_torque[0][iframe]*0.5*angle;
    work_done_0[0][iframe]=j_torque_0[0][iframe]*0.5*angle;

    x_react[1][iframe]=forefoot.r2x[iframe];    /* mid-tarsal joint */
    y_react[1][iframe]=forefoot.r2y[iframe];
    j_torque[1][iframe]=forefoot.m2[iframe];
    j_torque_0[1][iframe]=forefoot.m2_0[iframe];
    angle=(phi[hindfoot.segnum][iframe+1]-phi[hindfoot.segnum][iframe-1]-
        phi[forefoot.segnum][iframe+1]+phi[forefoot.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[1][iframe]=j_torque[1][iframe]*0.5*angle;
    work_done_0[1][iframe]=j_torque_0[1][iframe]*0.5*angle;

    x_react[2][iframe]=hindfoot.r2x[iframe];    /* ankle */
    y_react[2][iframe]=hindfoot.r2y[iframe];
    j_torque[2][iframe]=hindfoot.m2[iframe];
    j_torque_0[2][iframe]=hindfoot.m2_0[iframe];
    angle=(phi[calf.segnum][iframe+1]-phi[calf.segnum][iframe-1]-
        phi[hindfoot.segnum][iframe+1]+phi[hindfoot.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
    if (angle<=(-M_PI)) angle+=(M_PI*2);
    work_done[2][iframe]=j_torque[2][iframe]*0.5*angle;
    work_done_0[2][iframe]=j_torque_0[2][iframe]*0.5*angle;

    x_react[3][iframe]=calf.r2x[iframe];        /* knee */
    y_react[3][iframe]=calf.r2y[iframe];
    j_torque[3][iframe]=calf.m2[iframe];
    j_torque_0[3][iframe]=calf.m2_0[iframe];
    angle=(phi[thigh.segnum][iframe+1]-phi[thigh.segnum][iframe-1]-
        phi[calf.segnum][iframe+1]+phi[calf.segnum][iframe-1]);
    if (angle>M_PI) angle-=(M_PI*2);
```

```
if (angle<=(-M_PI)) angle+=(M_PI*2);
work_done[3][iframe]=j_torque[3][iframe]*0.5*angle;
work_done_0[3][iframe]=j_torque_0[3][iframe]*0.5*angle;

x_react[4][iframe]=thigh.r2x[iframe];        /* hip */
y_react[4][iframe]=thigh.r2y[iframe];
j_torque[4][iframe]=thigh.m2[iframe];
j_torque_0[4][iframe]=thigh.m2_0[iframe];
angle=(phi[torso.segnum][iframe+1]-phi[torso.segnum][iframe-1]-
    phi[thigh.segnum][iframe+1]+phi[thigh.segnum][iframe-1]);
if (angle>M_PI) angle-=(M_PI*2);
if (angle<=(-M_PI)) angle+=(M_PI*2);
work_done[4][iframe]=j_torque[4][iframe]*0.5*angle;
work_done_0[4][iframe]=j_torque_0[4][iframe]*0.5*angle;

x_react[5][iframe]=tail.r1x[iframe];        /* tail base */
y_react[5][iframe]=tail.r1y[iframe];
j_torque[5][iframe]=tail.m1[iframe];
angle=(phi[tail.segnum][iframe+1]-phi[tail.segnum][iframe-1]-
    phi[torso.segnum][iframe+1]+phi[torso.segnum][iframe-1]);
if (angle>M_PI) angle-=(M_PI*2);
if (angle<=(-M_PI)) angle+=(M_PI*2);
work_done[5][iframe]=j_torque[5][iframe]*0.5*angle;

x_react[6][iframe]=head.r2x[iframe];        /* neck */
y_react[6][iframe]=head.r2y[iframe];
j_torque[6][iframe]=head.m2[iframe];
angle=(phi[torso.segnum][iframe+1]-phi[torso.segnum][iframe-1]-
    phi[head.segnum][iframe+1]+phi[head.segnum][iframe-1]);
if (angle>M_PI) angle-=(M_PI*2);
if (angle<=(-M_PI)) angle+=(M_PI*2);
work_done[6][iframe]=j_torque[6][iframe]*0.5*angle;

x_react[7][iframe]=upper_arm.r2x[iframe];    /* shoulder */
y_react[7][iframe]=upper_arm.r2y[iframe];
j_torque[7][iframe]=upper_arm.m2[iframe];
angle=(phi[torso.segnum][iframe+1]-phi[torso.segnum][iframe-1]-
    phi[upper_arm.segnum][iframe+1]+phi[upper_arm.segnum][iframe-1]);
if (angle>M_PI) angle-=(M_PI*2);
if (angle<=(-M_PI)) angle+=(M_PI*2);
work_done[7][iframe]=j_torque[7][iframe]*0.5*angle;

x_react[8][iframe]=lower_arm.r2x[iframe];    /* elbow */
y_react[8][iframe]=lower_arm.r2y[iframe];
j_torque[8][iframe]=lower_arm.m2[iframe];
angle=(phi[upper_arm.segnum][iframe+1]-phi[upper_arm.segnum][iframe-1]-
    phi[lower_arm.segnum][iframe+1]+phi[lower_arm.segnum][iframe-1]);
if (angle>M_PI) angle-=(M_PI*2);
if (angle<=(-M_PI)) angle+=(M_PI*2);
work_done[8][iframe]=j_torque[8][iframe]*0.5*angle;

}

while ((iret=menu("Select option:",menu1,nmenu1))!=nmenu1)
{
    switch (iret)
    {
    case 1:
        node_plot(joints,njoints,x_react,y_react,dummy,1,nframe-1,
            times,"Reaction Forces","Force (N)");
        break;
```

```
        case 2:
            seg_plot(dummy,dummy,j_torque,1,nframe-1,joints,njoints,
                times,"Joint Torques","Torques (Nm)");
            break;

        case 3:
            seg_plot(dummy,dummy,j_torque_0,1,nframe-1,joints,njoints_0,
                times,"Zero Contact Torque Joint Torques",
                "Torques (Nm)");
            break;

        case 4:
            seg_plot(dummy,dummy,work_done,1,nframe-1,joints,njoints,
                times,"Work Done per frame","Energy (J)");

        case 5:
            seg_plot(dummy,dummy,work_done_0,1,nframe-1,joints,njoints_0,
                times,"Work Done per frame - 0 contact work","Energy (J)");
        }
    }
}
```

smooth()

```
#include "params.h"

void smooth(nnodes,xpos,ypos,zpos,xpos_filt,ypos_filt,zpos_filt,nframe,fspeed)

/* this routine performs moving average type smoothing */

int nnodes;                         /* Number of nodes */
float xpos[MAX_NODES][MAX_FRAMES];      /* x world coordinates (m) */
float ypos[MAX_NODES][MAX_FRAMES];      /* y world coordinates (m) */
float zpos[MAX_NODES][MAX_FRAMES];      /* z world coordinates (m) */
float xpos_filt[MAX_NODES][MAX_FRAMES];     /* filtered x world coordinates (m) */
float ypos_filt[MAX_NODES][MAX_FRAMES];     /* filtered y world coordinates (m) */
float zpos_filt[MAX_NODES][MAX_FRAMES];     /* filtered z world coordinates (m) */
int nframe;                         /* number of frames */
float fspeed;                       /* Film frame interval (s) */

{
    int inodes;                     /* node counter */
    int iframe;                     /* frame counter */
    int ismooth;                    /* smooth counter */
    int start_frame,end_frame;          /* start and end frames */

    /* test for suitable number of frames */

    if (nframe<smooth_number)
    {
        printf("Too few frames for filtration\n");
        for (inodes=0;inodes<nnodes;inodes++)
        {
            for (iframe=0;iframe<nframe;iframe++)
            {
                xpos_filt[inodes][iframe]=xpos[inodes][iframe];
                ypos_filt[inodes][iframe]=ypos[inodes][iframe];
                zpos_filt[inodes][iframe]=zpos[inodes][iframe];
            }
        }
        return;
```

```
}

/* put in some sensible values for end points so as not to crash anything */

start_frame=smooth_number/2;
end_frame=nframe-smooth_number/2;
for (inodes=0;inodes<nnodes;inodes++)
{
    for (iframe=0;iframe<start_frame;iframe++)
    {
        xpos_filt[inodes][iframe]=xpos[inodes][iframe];
        ypos_filt[inodes][iframe]=ypos[inodes][iframe];
        zpos_filt[inodes][iframe]=zpos[inodes][iframe];
    }
    for (iframe=end_frame;iframe<nframe;iframe++)
    {
        xpos_filt[inodes][iframe]=xpos[inodes][iframe];
        ypos_filt[inodes][iframe]=ypos[inodes][iframe];
        zpos_filt[inodes][iframe]=zpos[inodes][iframe];
    }
}

/* loop over nodes */

for (inodes=0;inodes<nnodes;inodes++)
{

    for (iframe=start_frame;iframe<end_frame;iframe++)
    {
        xpos_filt[inodes][iframe]=0.0;
        ypos_filt[inodes][iframe]=0.0;
        zpos_filt[inodes][iframe]=0.0;
        for (ismooth=0;ismooth<smooth_number;ismooth++)
        {
            xpos_filt[inodes][iframe]+=
                xpos[inodes][iframe+ismooth-start_frame];
            ypos_filt[inodes][iframe]+=
                ypos[inodes][iframe+ismooth-start_frame];
            zpos_filt[inodes][iframe]+=
                zpos[inodes][iframe+ismooth-start_frame];
        }
        xpos_filt[inodes][iframe]=
            xpos_filt[inodes][iframe]/(float)smooth_number;
        ypos_filt[inodes][iframe]=
            ypos_filt[inodes][iframe]/(float)smooth_number;
        zpos_filt[inodes][iframe]=
            zpos_filt[inodes][iframe]/(float)smooth_number;
    }
}

/* finished */

printf("Smoothing successful\n");

}
```

stats()

```
#include "params.h"

void stats(title,nodes,nnodes,xpos,ypos,zpos,xpos_filt,ypos_filt,zpos_filt,nframe,
```

```
    segs,nsegs,nodes_per_seg,fspeed,seg_mass,seg_com,seg_moi)

/* this routine performs simple dynamics calculations on the raw node position data */

char title[STRING_SIZE];                 /* Title of node file */
int nnodes;                              /* Number of nodes */
char nodes[MAX_NODES][STRING_SIZE];      /* Names of nodes */
float xpos[MAX_NODES][MAX_FRAMES];       /* x world coordinates (m) */
float ypos[MAX_NODES][MAX_FRAMES];       /* y world coordinates (m) */
float zpos[MAX_NODES][MAX_FRAMES];       /* z world coordinates (m) */
float xpos_filt[MAX_NODES][MAX_FRAMES];/* filtered x world coordinates (m) */
float ypos_filt[MAX_NODES][MAX_FRAMES];/* filtered y world coordinates (m) */
float zpos_filt[MAX_NODES][MAX_FRAMES];/* filtered z world coordinates (m) */
int nframe;                              /* number of frames */
char segs[MAX_SEGS][STRING_SIZE];        /* Names of segments */
int nsegs;                               /* Number of segments */
int nodes_per_seg[NPS][MAX_SEGS];        /* Nodes in segment */
float fspeed;                            /* Film frame interval (s) */
float seg_mass[MAX_SEGS];                /* array of segment masses */
float seg_com[MAX_SEGS];                 /* array of segment relative COMs */
float seg_moi[MAX_SEGS];                 /* array of segment MOIs */

{
    float times[MAX_FRAMES];                          /* times (s) */
    float xvel[MAX_NODES+MAX_SEGS][MAX_FRAMES];       /* calculated velocities (m/s) */
    float yvel[MAX_NODES+MAX_SEGS][MAX_FRAMES];
    float zvel[MAX_NODES+MAX_SEGS][MAX_FRAMES];
    float xacc[MAX_NODES+MAX_SEGS][MAX_FRAMES];       /* calculated accn (m/s/s) */
    float yacc[MAX_NODES+MAX_SEGS][MAX_FRAMES];
    float zacc[MAX_NODES+MAX_SEGS][MAX_FRAMES];
    float xapos[MAX_SEGS][MAX_FRAMES];        /* angle of segment (radian) */
    float yapos[MAX_SEGS][MAX_FRAMES];
    float zapos[MAX_SEGS][MAX_FRAMES];
    float xavel[MAX_SEGS][MAX_FRAMES];        /* calculated angular velocities (rad/s) */
    float yavel[MAX_SEGS][MAX_FRAMES];
    float zavel[MAX_SEGS][MAX_FRAMES];
    float xaacc[MAX_SEGS][MAX_FRAMES];        /* calculated angular accln (rad/s/s) */
    float yaacc[MAX_SEGS][MAX_FRAMES];
    float zaacc[MAX_SEGS][MAX_FRAMES];
    float body_comx[MAX_FRAMES];      /* x centre of mass of animal (m) */
    float body_comy[MAX_FRAMES];      /* y centre of mass of animal (m) */
    float body_comz[MAX_FRAMES];      /* z centre of mass of animal (m) */
    float body_mass;                   /* mass of animal (kg) */
    float comx[MAX_SEGS][MAX_FRAMES];     /* x component of segment COM */
    float comy[MAX_SEGS][MAX_FRAMES];     /* y component of segment COM */
    float comz[MAX_SEGS][MAX_FRAMES];     /* z component of segment COM */
    float comxvel[MAX_SEGS][MAX_FRAMES];/* x component of segment COM vel */
    float comyvel[MAX_SEGS][MAX_FRAMES];/* y component of segment COM vel */
    float comzvel[MAX_SEGS][MAX_FRAMES];/* z component of segment COM vel */
    float comxacc[MAX_SEGS][MAX_FRAMES];/* x component of segment COM accln */
    float comyacc[MAX_SEGS][MAX_FRAMES];/* y component of segment COM accln */
    float comzacc[MAX_SEGS][MAX_FRAMES];/* z component of segment COM accln */
    float xp[MAX_NODES+MAX_SEGS][MAX_FRAMES];     /* x position data used */
    float yp[MAX_NODES+MAX_SEGS][MAX_FRAMES];     /* y position data used */
    float zp[MAX_NODES+MAX_SEGS][MAX_FRAMES];     /* z position data used */
    float x_force[MAX_SEGS][MAX_FRAMES];     /* x component of force */
    float y_force[MAX_SEGS][MAX_FRAMES];     /* y component of force */
    float z_force[MAX_SEGS][MAX_FRAMES];     /* z component of force */
    float x_torque[MAX_SEGS][MAX_FRAMES];     /* torque in x=0 plane */
    float y_torque[MAX_SEGS][MAX_FRAMES];     /* torque in y=0 plane */
    float z_torque[MAX_SEGS][MAX_FRAMES];     /* torque in z=0 plane */
    float seg_PE[MAX_SEGS][MAX_FRAMES];         /* segment potential energy */
    float seg_LKE[MAX_SEGS][MAX_FRAMES];     /* segment linear kinetic energy */
```

```
float seg_RKE[MAX_SEGS][MAX_FRAMES];    /* segment rotl kinetic energy */
float seg_length[MAX_SEGS];            /* mean segment lengths */
char positions[MAX_NODES+MAX_SEGS][STRING_SIZE];/* names for position data */
int npositions;                        /* number of position items */
int inodes;                            /* node counter */
int iframe;                            /* frame counter */
int isegs;                            /* segment counter */
static char menu1[][STRING_SIZE]=       /* function selector menu */
{
    "Node position",
    "Node velocity",
    "Node acceleration",
    "Segment angle",
    "Segment angular velocity",
    "Segment angular acceleration",
    "Segment lengths",
    "Node locus",
    "Forces",
    "Torques",
    "Energies",
    "Options",
    "User specific analysis",
    "Exit"
};
static char menu2[][STRING_SIZE]=    /* option selection menu */
{
    "Raw data",
    "Exit"
};
int ifunc;                          /* function */
int iopt;                              /* option */
int change_flag=TRUE;                /* flag to change values */
static int smoothed_flag=TRUE;     /* smoothed data flag */
FILE *unit;                          /* file pointer */
char fname[STRING_SIZE];              /* filename */
char filename[STRING_SIZE];          /* full pathname */
int frames_lost;                      /* frames lost at each end */
int useable_frames;                  /* nframe corrected by frames_lost */
float time;                          /* intermediate time value */

/* calculate number of frames to remove */

if (flag_filter==TRUE) frames_lost=2;
else frames_lost=smooth_number/2;

/* menu 1 */

while ((ifunc=menu("Analysis function",menu1,14))!=14)
{
    if (change_flag==TRUE)
    {

        /* copy over values */

        if (smoothed_flag==TRUE)
        {
            for (inodes=0;inodes<nnodes;inodes++)
            {
                for (iframe=frames_lost;
                    iframe<(nframe-frames_lost);iframe++)
                {
                    xp[inodes][iframe-frames_lost]=
```

```
                xpos_filt[inodes][iframe];
            yp[inodes][iframe-frames_lost]=
                ypos_filt[inodes][iframe];
            zp[inodes][iframe-frames_lost]=
                zpos_filt[inodes][iframe];
        }
    }
    useable_frames=nframe-2*frames_lost;

    /* calculate times */

    time=fspeed*(float)frames_lost;
    for (iframe=0;iframe<useable_frames;iframe++)
    {
        times[iframe]=time;
        time=time+fspeed;
    }

}
else
{
    for (inodes=0;inodes<nnodes;inodes++)
    {
        for (iframe=0;iframe<nframe;iframe++)
        {
            xp[inodes][iframe]=
                xpos[inodes][iframe];
            yp[inodes][iframe]=
                ypos[inodes][iframe];
            zp[inodes][iframe]=
                zpos[inodes][iframe];
        }
    }
    useable_frames=nframe;

    /* calculate times */

    time=0.0;
    for (iframe=0;iframe<useable_frames;iframe++)
    {
        times[iframe]=time;
        time=time+fspeed;
    }
}

/* copy over position names */

for (inodes=0;inodes<nnodes;inodes++)
{
    strcpy(positions[inodes],nodes[inodes]);
}

/* calculate centres of mass */

com(xp,yp,zp,nnodes,useable_frames,nodes_per_seg,nsegs,seg_com,
    seg_mass,body_comx,body_comy,body_comz,&body_mass,comx,
    comy,comz);

/* calculate segment lengths */

length(xp,yp,zp,nnodes,useable_frames,nodes_per_seg,nsegs,
    seg_length);
```

```
/* add data onto position data */

for (isegs=0;isegs<nsegs;isegs++)
{
    for (iframe=0;iframe<useable_frames;iframe++)
    {
        xp[nnodes+isegs][iframe]=comx[isegs][iframe];
        yp[nnodes+isegs][iframe]=comy[isegs][iframe];
        zp[nnodes+isegs][iframe]=comz[isegs][iframe];
    }
    strcpy(positions[nnodes+isegs],segs[isegs]);
    strcat(positions[nnodes+isegs]," COM");
}
for (iframe=0;iframe<useable_frames;iframe++)
{
    xp[nnodes+nsegs][iframe]=body_comx[iframe];
    yp[nnodes+nsegs][iframe]=body_comy[iframe];
    zp[nnodes+nsegs][iframe]=body_comz[iframe];
}
strcpy(positions[nnodes+nsegs],"Whole body COM");
npositions=nnodes+nsegs+1;

/* calculate kinematics */

l_kinem(npositions,xp,yp,zp,useable_frames,fspeed,times,
    xvel,yvel,zvel,xacc,yacc,zacc);
r_kinem(xp,yp,zp,useable_frames,fspeed,nsegs,nodes_per_seg,times,
    xapos,yapos,zapos,xavel,yavel,zavel,xaacc,yaacc,zaacc);

/* put COM accel and vel into separate array */

for (isegs=0;isegs<nsegs;isegs++)
{
    for (iframe=0;iframe<useable_frames;iframe++)
    {
        comxvel[isegs][iframe]=xvel[nnodes+isegs][iframe];
        comyvel[isegs][iframe]=yvel[nnodes+isegs][iframe];
        comzvel[isegs][iframe]=zvel[nnodes+isegs][iframe];
        comxacc[isegs][iframe]=xacc[nnodes+isegs][iframe];
        comyacc[isegs][iframe]=yacc[nnodes+isegs][iframe];
        comzacc[isegs][iframe]=zacc[nnodes+isegs][iframe];
    }
}

for (iframe=0;iframe<useable_frames;iframe++)
{
    comxvel[nsegs][iframe]=xvel[nnodes+isegs][iframe];
    comyvel[nsegs][iframe]=yvel[nnodes+isegs][iframe];
    comzvel[nsegs][iframe]=zvel[nnodes+isegs][iframe];
    comxacc[nsegs][iframe]=xacc[nnodes+nsegs][iframe];
    comyacc[nsegs][iframe]=yacc[nnodes+nsegs][iframe];
    comzacc[nsegs][iframe]=zacc[nnodes+nsegs][iframe];
}
seg_mass[nsegs]=body_mass;
strcpy(segs[nsegs],"Whole body");

/* calculate dynamics */

l_dynam(comxacc,comyacc,comzacc,nsegs+1,useable_frames,seg_mass,
    x_force,y_force,z_force);
```

```
        r_dynam(xaacc,yaacc,zaacc,nsegs,useable_frames,seg_moi,
            x_torque,y_torque,z_torque);

        /* calculate energetics */

        energetics(comy,comxvel,comyvel,comzvel,xavel,yavel,zavel,
            seg_mass,seg_moi,nsegs,useable_frames,
            seg_PE,seg_LKE,seg_RKE);

        change_flag=FALSE;
    }

    switch (ifunc)
    {

        /* node position */

    case 1:
        node_plot(positions,npositions,xp,yp,zp,0,useable_frames,times,
            "Node Position","Distance (m)");
        break;

        /* node velocity */

    case 2:
        node_plot(positions,npositions,xvel,yvel,zvel,1,useable_frames-1,
            times,"Node Velocity","Velocity (m/s)");
        break;

        /* node acceleration */

    case 3:
        node_plot(positions,npositions,xacc,yacc,zacc,1,useable_frames-1,
            times,"Node Acceleration","Acceleration (m/s/s)");
        break;

        /* segment angle */

    case 4:
        seg_plot(xapos,yapos,zapos,0,useable_frames,segs,nsegs,times,
            "Segment Angle","Angle (r)");
        break;

        /* segment angular velocity */

    case 5:
        seg_plot(xavel,yavel,zavel,1,useable_frames-1,segs,nsegs,times,
            "Segment Angular Velocity","Angular Velocity (r/s)");
        break;

        /* segment angular acceleration */

    case 6:
        seg_plot(xaacc,yaacc,zaacc,1,useable_frames-1,segs,nsegs,times,
            "Segment Angular Acceleration","Angular Acceleration (r/s/s)");
        break;

        /* segment lengths */

    case 7:
        seg_lengths(segs,nsegs,seg_length);
        break;
```

```
        /* node locus */

    case 8:
        locus(positions,npositions,xp,yp,zp,useable_frames);
        break;

        /* forces */

    case 9:
        node_plot(segs,nsegs+1,x_force,y_force,z_force,1,useable_frames-1,
            times, "Resultant Force on COM","Force (N)");
        break;

        /* torques */

    case 10:
        seg_plot(x_torque,y_torque,z_torque,1,useable_frames-1,segs,nsegs,
            times,"Torque about COM","Torque (Nm)");
        break;

        /* energies */

    case 11:

        energy_plot(segs,nsegs,seg_PE,seg_LKE,seg_RKE,1,useable_frames-1,
            times);
        break;

        /* options */

    case 12:
        change_flag=TRUE;
        if (smoothed_flag==TRUE) strcpy(menu2[0],"Raw data");
        else strcpy(menu2[0],"Smoothed data");
        while ((iopt=menu("Options:",menu2,2))!=2)
        {
            switch (iopt)
            {
            case 1:
                if (smoothed_flag==TRUE)
                {
                    strcpy(menu2[0],"Smoothed data");
                    smoothed_flag=FALSE;
                }
                else
                {
                    strcpy(menu2[0],"Raw data");
                    smoothed_flag=TRUE;
                }
                break;
            }
        }
        break;

        /* do user specific analysis */

    case 13:

        user_specific_analysis(xp,yp,zp,xvel,yvel,zvel,xacc,yacc,zacc,
            xapos,yapos,zapos,xavel,yavel,zavel,xaacc,yaacc,zaacc,
            x_force,y_force,z_force,x_torque,y_torque,z_torque,
```

```
                comx,comy,comz,comxacc,comyacc,comzacc,
                body_comx,body_comy,body_comz,body_mass,
                times,seg_length,seg_mass,seg_com,seg_moi,
                positions,npositions,nodes,nnodes,segs,nsegs,useable_frames,
                seg_PE,seg_LKE,seg_RKE);
            break;
        }
    }

    /* finished */

}
```

toe_tip_and_body_COM_output()

```
/* routine to output toe tip and body COM information */

#include "params.h"

void toe_tip_and_body_COM_output(positions,xp,yp,times,npositions,nframe)

char positions[MAX_NODES+MAX_SEGS][STRING_SIZE];/* position names */
float xp[MAX_NODES+MAX_SEGS][MAX_FRAMES];    /* x coordinates */
float yp[MAX_NODES+MAX_SEGS][MAX_FRAMES];    /* y coordinates */
float times[MAX_FRAMES];              /* times */
int npositions;                      /* number of positions */
int nframe;                          /* number of frames */

{
    int iposition;                  /* position counter */
    int toe_tip;                     /* toe tip index */
    int body_COM;                    /* body COM index */
    int iframe;                     /* frame counter */
    FILE *unit;                     /* file pointer */
    char fname[STRING_SIZE];         /* intermediate filename */
    char filename[STRING_SIZE];     /* full filename */

    /* look for required positions */

    for (iposition=0;iposition<npositions;iposition++)
    {
        if (strcmp("Toe tip",positions[iposition])==0) toe_tip=iposition;
        if (strcmp("Whole body COM",positions[iposition])==0) body_COM=iposition;
    }

    /* write out data */

    do
    {

        /* get filename */

        printf("Input EXCEL data file name : ");
        scanf("%s",fname);
        strcpy(filename,ANALYSIS_DIRECTORY);
        strcat(filename,ANALYSIS_PREFIX);
        strcat(filename,fname);
        strcat(filename,ANALYSIS_SUFFIX_EXCEL);

        /* open file */
```

```
    unit=fopen(filename,"w");
} while (unit==NULL);

/* write out data in ASCII form suitable for EXCEL import */

fprintf(unit,"Time\011Toe Tip x\011Toe Tip y\011CM x\011CM y\015\012");
fprintf(unit,"\015\012");
for (iframe=0;iframe<nframe;iframe++)
{
    fprintf(unit,"%12.5e\011%12.5e\011%12.5e\011%12.5e\011%12.5e\015\012",
        times[iframe],xp[toe_tip][iframe],yp[toe_tip][iframe],
        xp[body_COM][iframe],yp[body_COM][iframe]);
}

    fclose(unit);

    printf("File %s written successfully\n",filename);
}
```

translate3d()

```
/* Subroutine produces a translation 4*4 matrix from the transformation vector */

#include "params.h"

void translate3d(tx,ty,tz,m)
double tx,ty,tz;
float m[4][4];
{
    identity3d(m);
    m[3][0]=(float)tx;
    m[3][1]=(float)ty;
    m[3][2]=(float)tz;
}
```

user_specific_analysis()

```
#include "params.h"

user_specific_analysis(xp,yp,zp,xvel,yvel,zvel,xacc,yacc,zacc,
    xapos,yapos,zapos,xavel,yavel,zavel,xaacc,yaacc,zaacc,
    x_force,y_force,z_force,x_torque,y_torque,z_torque,
    comx,comy,comz,comxacc,comyacc,comzacc,
    body_comx,body_comy,body_comz,body_mass,
    times,seg_length,seg_mass,seg_com,seg_moi,
    positions,npositions,nodes,nnodes,segs,nsegs,nframe,
    seg_PE,seg_LKE,seg_RKE)

int nnodes;                                  /* Number of nodes */
char nodes[MAX_NODES][STRING_SIZE];          /* Names of nodes */
int nframe;                                  /* number of frames */
char segs[MAX_SEGS][STRING_SIZE];           /* Names of segments */
int nsegs;                                   /* Number of segments */
float seg_mass[MAX_SEGS];                   /* array of segment masses */
float seg_com[MAX_SEGS];                     /* array of segment relative COMs */
float seg_moi[MAX_SEGS];                     /* array of segment MOIs */
float times[MAX_FRAMES];                     /* times (s) */
float xvel[MAX_NODES+MAX_SEGS][MAX_FRAMES];    /* calculated velocities (m/s) */
float yvel[MAX_NODES+MAX_SEGS][MAX_FRAMES];
float zvel[MAX_NODES+MAX_SEGS][MAX_FRAMES];
```

```
float xacc[MAX_NODES+MAX_SEGS][MAX_FRAMES];    /* calculated accelerations (m/s/s) */
float yacc[MAX_NODES+MAX_SEGS][MAX_FRAMES];
float zacc[MAX_NODES+MAX_SEGS][MAX_FRAMES];
float xapos[MAX_SEGS][MAX_FRAMES];          /* angle of segment (radian) */
float yapos[MAX_SEGS][MAX_FRAMES];
float zapos[MAX_SEGS][MAX_FRAMES];
float xavel[MAX_SEGS][MAX_FRAMES];          /* calculated ang vels (rad/s) */
float yavel[MAX_SEGS][MAX_FRAMES];
float zavel[MAX_SEGS][MAX_FRAMES];
float xaacc[MAX_SEGS][MAX_FRAMES];          /* calculated ang accln (rad/s/s) */
float yaacc[MAX_SEGS][MAX_FRAMES];
float zaacc[MAX_SEGS][MAX_FRAMES];
float body_comx[MAX_FRAMES];              /* x centre of mass of animal (m) */
float body_comy[MAX_FRAMES];              /* y centre of mass of animal (m) */
float body_comz[MAX_FRAMES];              /* z centre of mass of animal (m) */
float body_mass;                             /* mass of animal (kg) */
float comx[MAX_SEGS][MAX_FRAMES];          /* x component of segment COM */
float comy[MAX_SEGS][MAX_FRAMES];          /* y component of segment COM */
float comz[MAX_SEGS][MAX_FRAMES];          /* z component of segment COM */
float comxacc[MAX_SEGS][MAX_FRAMES];          /* x component of seg COM accln */
float comyacc[MAX_SEGS][MAX_FRAMES];          /* y component of seg COM accln */
float comzacc[MAX_SEGS][MAX_FRAMES];          /* z component of seg COM accln */
float xp[MAX_NODES+MAX_SEGS][MAX_FRAMES];     /* x position data used */
float yp[MAX_NODES+MAX_SEGS][MAX_FRAMES];     /* y position data used */
float zp[MAX_NODES+MAX_SEGS][MAX_FRAMES];     /* z position data used */
float x_force[MAX_SEGS][MAX_FRAMES];          /* x component of force */
float y_force[MAX_SEGS][MAX_FRAMES];          /* y component of force */
float z_force[MAX_SEGS][MAX_FRAMES];          /* z component of force */
float x_torque[MAX_SEGS][MAX_FRAMES];          /* torque in x=0 plane */
float y_torque[MAX_SEGS][MAX_FRAMES];          /* torque in y=0 plane */
float z_torque[MAX_SEGS][MAX_FRAMES];          /* torque in z=0 plane */
float seg_length[MAX_SEGS];                   /* mean segment lengths */
char positions[MAX_NODES+MAX_SEGS][STRING_SIZE];/* names for position data */
int npositions;                               /* number of position items */
float seg_PE[MAX_SEGS][MAX_FRAMES];           /* segment potential energy */
float seg_LKE[MAX_SEGS][MAX_FRAMES];          /* segment linear kinetic energy */
float seg_RKE[MAX_SEGS][MAX_FRAMES];          /* segment rotational kinetic energy */

/* put user specified analysis routines in this function */

{
    static char menu1[][STRING_SIZE]=    /* menu prompts */
    {
        "Simplified Quadrupedal Analysis",
        "Predictive Model Analysis",
        "Toe tip and Body COM output",
        "Exit"
    };
    int nmenu=4;                          /* number of menu items */
    int iret;                              /* menu return value */

    /* menu */

    while((iret=menu("User Specific Analysis",menu1,nmenu))!=nmenu)
    {
        switch (iret)
        {
        case 1:
            simplified_quadrupedal(seg_length,seg_mass,seg_com,
                zapos,x_force,y_force,z_torque,segs,nsegs,nframe,times);
            break;
```

```
        case 2:
            predictive_analysis(seg_length,seg_mass,seg_com,
                zapos,x_force,y_force,z_torque,segs,nsegs,nframe,times);
            break;

        case 3:
            toe_tip_and_body_COM_output(positions,xp,yp,times,npositions,
                nframe);
            break;
        }
    }
}
```

view()

```
#include "params.h"

void view()

/* this routine allows the user to view the video frames */

{
    int iret;                   /* menu selection */
    char fname[STRING_SIZE];          /* picture file name */
    char filename[STRING_SIZE];         /* full picture file name */
    static char menu1[][STRING_SIZE]=    /* menu */
    {
        "Read frame file",
        "Write picture file",
        "Exit"
};

/* get into right graphics mode */

CLEAR_GRAPH;

iret=1;
do
{
    switch (iret)
    {

    case 1:

        /* read in picture file */

        readpic();
        break;

case 2:

        /* write out picture file */

        printf("Input picture file name : ");
        scanf("%s",fname);
        strcpy(filename,PICTURE_DIRECTORY);
        strcat(filename,PICTURE_PREFIX);
        strcat(filename,fname);
        strcat(filename,PICTURE_SUFFIX);

        bitmap_to_file(display,TRUE,0,0,filename,TRUE,0.0,0.0,0,0,TRUE);
```

```
            break;
        }

    } while ((iret=menu("Select option:",menu1,3))!=3);

    /* terminate graphics */

    CLEAR_GRAPH;

    /* finished */

}
```

wrnode()

```
#include "params.h"

void wrnode(title,nodes,nnodes,xpos,ypos,zpos,nframe,fspeed)

/* this routine writes out the node position file */

char title[STRING_SIZE];                /* file title line */
char nodes[MAX_NODES][STRING_SIZE];     /* names of the nodes of the model */
int nnodes;                             /* the number of nodes */
float xpos[MAX_NODES][MAX_FRAMES];      /* the x world coordinates */
float ypos[MAX_NODES][MAX_FRAMES];      /* the y world coordinates */
float zpos[MAX_NODES][MAX_FRAMES];      /* the z world coordinates */
int nframe;                             /* the number of frames */
int fspeed;                             /* the interval between frames */


{

    FILE *unit;                         /* file pointer */
    char fname[STRING_SIZE];            /* filename */
    char filename[STRING_SIZE];         /* full filename */
    int iframes;                        /* counter frame number */
    int inodes;                         /* counter node number */

    do
    {

        /* get filename */

        printf("Input node data file name : ");
        scanf("%s",fname);
        strcpy(filename,NODE_DIRECTORY);
        strcat(filename,NODE_PREFIX);
        strcat(filename,fname);
        strcat(filename,NODE_SUFFIX);

        /* open file */

        unit=fopen(filename,"w");
    } while (unit==NULL);

    /* write data */

    fprintf(unit,"%s\n",title);
    fprintf(unit,"%e\n",fspeed);
    fprintf(unit,"%d\n",nframe);
    for (iframes=0;iframes<nframe;iframes++)
```

```
{
    fprintf(unit,"%d\n",iframes);
    fprintf(unit,"%d\n",nnodes);
    for (inodes=0;inodes<nnodes;inodes++)
    {
        fprintf(unit,"%d %e %e %e\n",inodes,xpos[inodes][iframes],
            ypos[inodes][iframes],zpos[inodes][iframes]);
        fprintf(unit,"%s\n",nodes[inodes]);
    }
}

    /* close file */

    fclose(unit);

    /* print success message */

    printf("File %s written successfully\n",filename);

    /* finished */

}
```

yesno()

```
#include "params.h"

int yesno(prompt)

/* this routine waits for the user to enter y (TRUE) or n (FALSE) via a mouse driven
menu */

char prompt[];                          /* This is the menu title string */

{
    int reply;                      /* the value returned TRUE/FALSE */
    int iret=0;                      /* the value returned by the menu */

    static char menu1[][STRING_SIZE]=    /* the menu */
    {
        "Yes",
        "No"
    };

    while (iret==0)
        iret=menu(prompt,menu1,2);

    reply= iret==1 ? TRUE:FALSE;
    return (reply);

}
```

FORTRAN Glue Routines

fparams.h

```
      implicit none

c     max size of euqtion matrix
```

```
      integer MAX_MATRIX
      parameter (MAX_MATRIX=50)

c     number of DLT parameters

      integer NDLT
      parameter (NDLT=11)
```

dlt_parameters()

```
c     This subroutine solves the DLT reconstruction equations
c     calculating the parameters L1 to L11

c     solved using minimax algorithm

      subroutine dlt_parameters(xw,yw,zw,q,r,nrefs,l)

      include 'fparams.h'

      real xw(nrefs),yw(nrefs),zw(nrefs),
     1    q(nrefs),r(nrefs)
      integer nrefs
      real l(nrefs)

c     xw,yw,zw    world coordinates of reerence points
c     q,r    screen coordinates of reference points
c     nrefs    number of reference points
c     l        dlt parameters

      double precision a(MAX_MATRIX,MAX_MATRIX),b(MAX_MATRIX),
     1                 x(MAX_MATRIX),ta(MAX_MATRIX,MAX_MATRIX)
      integer m,n,ndim,mdim,irank,iter,ifail
      double precision tol,relerr,resmax

c     m        number of equations
c     n         number of unknowns
c     ndim    1st dimension of a
c     mdim    2nd dimension of a
c     a        ndim * mdim matrix for equations
c     ta    transpose of a
c     b        m matrix of RHS of equations
c     tol    tolerance
c     relerr    max acceptable error
c     resmax    largest residual
c     irank    rank of matrix a
c     iter    number of iterations
c     ifail     failure flag

      integer iref,i,im,in

c     iref    reference value counter
c     i        DLT parameter counter
c     im,in    counters to transpose matrix

c     setup arrays for equation solution

            do 10 iref=1,nrefs

            a(iref*2-1,1)=-xw(iref)
            a(iref*2-1,2)=-yw(iref)
```

```
            a(iref*2-1,3)=-zw(iref)
            a(iref*2-1,4)=-1.0
            a(iref*2-1,5)=0.0
            a(iref*2-1,6)=0.0
            a(iref*2-1,7)=0.0
            a(iref*2-1,8)=0.0
            a(iref*2-1,9)=q(iref)*xw(iref)
            a(iref*2-1,10)=q(iref)*yw(iref)
            a(iref*2-1,11)=q(iref)*zw(iref)

            a(iref*2,1)=0.0
            a(iref*2,2)=0.0
            a(iref*2,3)=0.0
            a(iref*2,4)=0.0
            a(iref*2,5)=-xw(iref)
            a(iref*2,6)=-yw(iref)
            a(iref*2,7)=-zw(iref)
            a(iref*2,8)=-1.0
            a(iref*2,9)=r(iref)*xw(iref)
            a(iref*2,10)=r(iref)*yw(iref)
            a(iref*2,11)=r(iref)*zw(iref)

          b(iref*2-1)=-q(iref)
          b(iref*2)=-r(iref)

10            continue

c     get values into variables

      m=2*nrefs
c     m=NDLT
      n=NDLT
      mdim=MAX_MATRIX
      ndim=MAX_MATRIX
      tol=0.0
      relerr=0.0
      ifail=1

c     transpose a

          do 20 im=1,m
                do 30 in=1,n
                    ta(in,im)=a(im,in)
30                  continue
20            continue


c     call nag routine to perform calculation

      call e02gcf(m,n,mdim,ndim,ta,b,tol,relerr,x,
     1            resmax,irank,iter,ifail)

      write(6,98)ifail,resmax,iter
98    format(' IFAIL = ',i2,' RESMAX = ',1pe12.4,' ITER = ',i10)

c     put solution into correct array

            do 40 i=1,NDLT
            l(i)=x(i)
40          continue

      return
```

```
      end
```

dlt_recon()

```
c     This routine uses the DLT parameters to reconstruct
c     the (x,y,z) coordinates from two sets of screen coordinates

c     calculates the minimax solution of the equations

      subroutine dlt_recon(la,lb,qa,ra,qb,rb,xxw,yyw,zzw)

      include 'fparams.h'

      real la(11),lb(11)
      real qa,ra
      real qb,rb
      real xxw,yyw,zzw

c     la,lb        DLT reconstruction parameters
c     qa,ra         screen 1 (q,r) coordinates
c     qb,rb         screen 2 (q,r) coordinates
c     xxw,yyw,zzw    reconstructed world coordinates

      double precision a(MAX_MATRIX,MAX_MATRIX),b(MAX_MATRIX),
     1                 x(MAX_MATRIX),ta(MAX_MATRIX,MAX_MATRIX)
      integer m,n,ndim,mdim,irank,iter,ifail
      double precision tol,relerr,resmax

c     m         number of equations
c     n          number of unknowns
c     ndim     1st dimension of a
c     mdim     2nd dimension of a
c     a         ndim * mdim matrix for equations
c     ta     transpose of a
c     b         m matrix of RHS of equations
c     tol     tolerance
c     relerr     max acceptable error
c     resmax     largest residual
c     irank     rank of matrix a
c     iter     number of iterations
c     ifail      failure flag

      integer im,in

c     im,in     counters to transpose matrix

c     get values into array

      a(1,1)=qa*la(9)-la(1)
      a(1,2)=qa*la(10)-la(2)
      a(1,3)=qa*la(11)-la(3)

      b(1)=la(4)-qa

      a(2,1)=ra*la(9)-la(5)
      a(2,2)=ra*la(10)-la(6)
      a(2,3)=ra*la(11)-la(7)

      b(2)=la(8)-ra

      a(3,1)=qb*lb(9)-lb(1)
```

```
      a(3,2)=qb*lb(10)-lb(2)
      a(3,3)=qb*lb(11)-lb(3)

      b(3)=lb(4)-qb

      a(4,1)=rb*lb(9)-lb(5)
      a(4,2)=rb*lb(10)-lb(6)
      a(4,3)=rb*lb(11)-lb(7)

      b(4)=lb(8)-rb

c     get values into variables

      m=4
      n=3
      mdim=MAX_MATRIX
      ndim=MAX_MATRIX
      tol=0.0
      relerr=0.0
      ifail=1

c     transpose a

          do 20 im=1,m
                  do 30 in=1,n
                      ta(in,im)=a(im,in)
30                    continue
20            continue

c     call nag routine to perform calculation

      call e02gcf(m,n,mdim,ndim,ta,b,tol,relerr,x,
     1            resmax,irank,iter,ifail)

c     put solution into correct variables

      xxw=x(1)
      yyw=x(2)
      zzw=x(3)

      return
      end
```

general.c

```
C     A general set of routines to pass values along to the NAG fortran routines

c     string cleanup routine

      subroutine cleanup(string)
      include 'fparams.h'

      character *80 string
      integer i,f

      f=0
          do 10 i=1,80
          if (f.ne.0) then
          string(i:i)=char(0)
          else
                  if (string(i:i).eq.char(0)) then
```

```
                    f=1
                endif
                    endif
10              continue
            return
            end

            subroutine cnagsti(path,device,pawse)
            include 'fparams.h'

            character *80 path,device
            logical pawse

            call nagsti(path,device,pawse)
            return
            end


            subroutine cj06ahf(title)
            include 'fparams.h'

            character *80 title

            call cleanup(title)
            call j06ahf(title)
            return
            end


            subroutine cj06ajf(iaxis,title)
            include 'fparams.h'

            character *80 title
            integer iaxis

            call cleanup(title)
            call j06ajf(iaxis,title)
            return
            end


            subroutine cj06zaf(string)
            include 'fparams.h'

            character *80 string

            call cleanup(string)
            call j06zaf(string)
            return
            end

            program start

            include 'fparams.h'

            call gap
            end
```

digit.exe

C Routines

visilog.h

```
/* visilog.h - visilog information */

/* this structure uses high to low byte order for its long integers */
/* this is NOT normal and hence the function to reverse the order of */
/* the bytes */

struct visilogImageHeader
{
    long int magicNumber;
    long int pixelsPerLine;
    long int numberOfLines;
    long int res1;
    long int res2;
    long int res3;
    long int gridType;                  /* rectangular grid */
    long int res4;
    long int arithmaticType;            /* long integer image */
    long int bitsPerPixel;
    long int res5;
    long int xOrigin;
    long int yOrigin;
    long int res6;
    long int res7;
    long int visilogHeaderSize;
    long int userHeaderSize;
    long int res8;
    long int totalHeaderSize;
};

long int VisilogConvert(int);

/* routine to cenvert ordinary integers to visilog style reversed 4 byte integers */

long int VisilogConvert(value)
int value;
{
    long int returnValue=0L;
    unsigned char *pointer1,*pointer2;
    unsigned char byte;

    pointer1=(unsigned char *)&value;
    byte=*pointer1;
    pointer2=(unsigned char *)&returnValue+3;
    *pointer2=byte;

    pointer1++;
    pointer2--;
    byte=*pointer1;
    *pointer2=byte;

    return(returnValue);
}
```

digit.c

```
#include <stdio.h>
#include <time.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <conio.h>
#include <bios.h>
#include <graph.h>
#include <string.h>
#include <dos.h>
#include "matrox.h"
#include "visilog.h"

#define VBLANK 0x126c                    /* non-zero when in vertical blanking
period */
#define STATUS 0x026c                    /* status register */
#define CONTROL2 0x0a6c                  /* control register 2 */
#define ODDFIELD 0x20                    /* odd field flag */
#define FGRACT 0x08                      /* frame grab active flag */
#define INTERVAL 8                       /* frame interval between pulses */
#define STRINGSIZE 80                    /* default size of strings */

void main(void);
void SetupMatrox(void);
void SetupSerial(void);
void MainMenu(void);
void WriteSoundtrack(void);
void ReadSoundtrack(void);
void GrabSingleFrame(void);
void DigitizeFrames(void);
void SynchronizePulses(void);
void Controls(void);
void FrameSave(char *,int,int);
void FillIn(void);
void FileName(char *,int,char *);
int Menu(char *,char[][STRINGSIZE],int);

extern void grab4(void);                 /* machine code routine to grab 4 successive
frames */


void main()
{

    /* initialize matrox card */

    SetupMatrox();

    /* initialize serial port */

    SetupSerial();

    /* go do main menu loop */

    MainMenu();
}

void MainMenu()
{
    static char menu1[][STRINGSIZE]=
```

```
    {
        "Write Soundtrack",
        "Read and Display Soundtrack",
        "Grab Single Frame",
        "Digitize Frames",
        "Adjust Brightness and Contrast",
        "Exit"
    };
    int iret;               /* menu return value */

    while ((iret=Menu("Main Menu",menu1,6))!=6)
    {
        switch (iret)
        {
        case 1:
            WriteSoundtrack();
            break;

        case 2:
            ReadSoundtrack();
            break;

        case 3:
            GrabSingleFrame();
            break;

        case 4:
            DigitizeFrames();
            break;

        case 5:
            Controls();
            break;
        }
    }
}

/* write the timed soundtrack onto the video tape */

void WriteSoundtrack()
{
    unsigned int frameCounter;              /* frame counter */
    unsigned int pulseCounter;              /* pulse counter */
    struct timeb time1,time2;               /* time stores */
    float elapsedTime;                      /* elapsed time */
    unsigned char *lowByte,*highByte;       /* pointers to individual bytes */

    SetupSerial();

    _clearscreen(_GCLEARSCREEN);
    printf("Start dubbing on video recorder and press any key\n\n");
    while (_bios_keybrd(_KEYBRD_READY)==0);
    _bios_keybrd(_KEYBRD_READ);

    printf("Writing soundtrack\n\n\nPress any key when finished\n");

    lowByte=(unsigned char *)&pulseCounter;
    highByte=lowByte+1;
    pulseCounter=0;
    ftime(&time1);
    while (_bios_keybrd(_KEYBRD_READY)==0)
    {
```

```
    /* wait for change from odd to even field  4 times */

    for (frameCounter=0;frameCounter<INTERVAL;frameCounter++)
    {
        while ((inp(STATUS)&ODDFIELD)!=0);
        while ((inp(STATUS)&ODDFIELD)==0);
    }
    _bios_serialcom(_COM_SEND,0,(int)*lowByte);
    _bios_serialcom(_COM_SEND,0,(int)*highByte);
    pulseCounter++;
}
ftime(&time2);
elapsedTime=(float)time2.time-(float)time1.time+
    (float)time2.millitm/1000.0-(float)time1.millitm/1000.0;
_bios_keybrd(_KEYBRD_READ);

printf("Time elapsed is %f\n",elapsedTime);
printf("Pulses written  %u\n\n",pulseCounter);
printf("Expected elapsed time is %f\n",
    (float)pulseCounter*(float)INTERVAL/25.0);

printf("\nPress any key to continue\n");
while (_bios_keybrd(_KEYBRD_READY)==0);
_bios_keybrd(_KEYBRD_READ);
}

/* read soundtrack and display the pulse number */

void ReadSoundtrack()
{
    unsigned int pulseCounter;              /* pulse counter */
    unsigned int byte1,byte2;               /* data bytes */
    struct timeb time1,time2;               /* time stores */
    float elapsedTime;                      /* elapsed time */
    unsigned char *lowByte,*highByte;       /* pointers to individual bytes */

    _clearscreen(_GCLEARSCREEN);

    lowByte=(unsigned char *)&pulseCounter;
    highByte=lowByte+1;

    /* synchronize to pulses */

    SynchronizePulses();

    /* all synchronized now - next character will be first of a pair */

    ftime(&time1);
    while (_bios_keybrd(_KEYBRD_READY)==0)
    {
        byte1=_bios_serialcom(_COM_RECEIVE,0,0);
        byte1=byte1&0x00ff;
        byte2=_bios_serialcom(_COM_RECEIVE,0,0);
        byte2=byte2&0x00ff;

        *lowByte=(unsigned char)byte1;
        *highByte=(unsigned char)byte2;

        _settextposition(0,0);
        printf("%5u",pulseCounter);
    }
```

```
    ftime(&time2);
    fg_snap(1);
    fg_sbuf(1);
    fg_sync(0);
    elapsedTime=(float)time2.time-(float)time1.time+
        (float)time2.millitm/1000.0-(float)time1.millitm/1000.0;
    _bios_keybrd(_KEYBRD_READ);

    printf("\nTime elapsed is %f\n",elapsedTime);
    printf("Pulses read  %u\n\n",pulseCounter);
    printf("Expected elapsed time is %f\n",
        (float)pulseCounter*(float)INTERVAL/25.0);

    printf("\nPress any key to continue\n");
    while (_bios_keybrd(_KEYBRD_READY)==0);
    _bios_keybrd(_KEYBRD_READ);
    fg_sync(1);
    fg_sbuf(0);
}

void GrabSingleFrame()
{
    unsigned int byte1,byte2;                   /* data bytes */
    unsigned int pulseCounter;                  /* pulse counter */
    unsigned char *lowByte,*highByte;           /* pointers to individual bytes */
    unsigned int pulse;                         /* pulse counter */

    _clearscreen(_GCLEARSCREEN);

    lowByte=(unsigned char *)&pulseCounter;
    highByte=lowByte+1;

    printf("Input counter value :");
    scanf("%u",&pulse);

    /* loop through numbered pulses */

    _clearscreen(_GCLEARSCREEN);
    printf("Searching for pulse %u\n",pulse);

    /* synchronize */

    SynchronizePulses();

    /* all synchronized now - next character will be first of a pair */

    while (1)
    {
        byte1=_bios_serialcom(_COM_RECEIVE,0,0);
        byte1=byte1&0x00ff;
        byte2=_bios_serialcom(_COM_RECEIVE,0,0);
        byte2=byte2&0x00ff;

        *lowByte=(unsigned char)byte1;
        *highByte=(unsigned char)byte2;

        if (pulse==pulseCounter) break;
        _settextposition(2,2);
        printf("%5u",pulseCounter);
    }

    /* at right place now, so grab frame */
```

```
    fg_dquad(0);                    /* grab quadrant 0 */
    fg_snap(1);
    fg_sbuf(1);                     /* show buffer */
    fg_sync(0);                     /* internal sync */

    printf("\nPress any key to continue\n");
    while (_bios_keybrd(_KEYBRD_READY)==0);
    _bios_keybrd(_KEYBRD_READ);

    fg_sync(1);                     /* external sync */
    fg_sbuf(0);                     /* show incoming signal */

}

void DigitizeFrames()
{
    unsigned int pulseCounter;              /* pulse counter */
    unsigned int byte1,byte2;               /* data bytes */
    unsigned char *lowByte,*highByte;       /* pointers to individual bytes */
    unsigned int startPulse;                /* first pulse to start digitizing on */
    unsigned int numberOfPulses;            /* number of pulses to digitize over */
    unsigned int endPulse;                  /* end pulse for digitizing */
    unsigned int pulse;                     /* pulse counter */
    char sequenceName[STRINGSIZE];          /* sequence name */
    char fileName[STRINGSIZE];              /* file name */
    int frameCounter=0;                     /* frame counter */
    int temporaryFrameCounter;              /* temporary frame counter */
    int quadrant;                           /* quadrant counter */
    int field;                              /* field counter */

    _clearscreen(_GCLEARSCREEN);

    lowByte=(unsigned char *)&pulseCounter;
    highByte=lowByte+1;

    printf("Input sequence name:");
    scanf("%s",sequenceName);

    printf("Input start pulse number:");
    scanf("%u",&startPulse);
    printf("Input number of pulses  :");
    scanf("%u",&numberOfPulses);
    endPulse=startPulse+numberOfPulses;

    /* loop through numbered pulses */

    for (pulse=startPulse;pulse<endPulse;pulse++)
    {
        /* get even numbered frames first */

        temporaryFrameCounter=frameCounter;

        _clearscreen(_GCLEARSCREEN);
        printf("Searching for pulse %u\n",pulse);

        /* synchronize */

        SynchronizePulses();

        /* all synchronized now - next character will be first of a pair */
```

```
        while (1)
        {
            byte1=_bios_serialcom(_COM_RECEIVE,0,0);
            byte1=byte1&0x00ff;
            byte2=_bios_serialcom(_COM_RECEIVE,0,0);
            byte2=byte2&0x00ff;

            *lowByte=(unsigned char)byte1;
            *highByte=(unsigned char)byte2;

            if (pulse==pulseCounter) break;
            _settextposition(2,2);
            printf("%5u",pulseCounter);
        }

        /* at right place now, so grab next 4 frames */
        /* making sure every other frame grabbed */

        _disable();                     /* interrupts off */
        fg_dquad(0);                    /* grab quadrant 0 */
        fg_snap(1);
        while(inp(VBLANK)==0);
        while(inp(VBLANK)!=0);
        while(inp(VBLANK)==0);
        fg_dquad(1);                    /* grab quadrant 1 */
        fg_snap(1);
        while(inp(VBLANK)==0);
        while(inp(VBLANK)!=0);
        while(inp(VBLANK)==0);
        fg_dquad(2);                    /* grab quadrant 2 */
        fg_snap(1);
        while(inp(VBLANK)==0);
        while(inp(VBLANK)!=0);
        while(inp(VBLANK)==0);
        fg_dquad(3);                    /* grab quadrant 3 */
        fg_snap(1);
        _enable();                      /* interrupts back on */

        /* write all eight field out to files */

        _settextposition(2,2);
        printf("%5u\n\n",pulseCounter);
        fg_sbuf(1);                     /* show buffer */
        fg_sync(0);                     /* internal sync */
        for (quadrant=0;quadrant<4;quadrant++)
        {
            fg_dquad(quadrant);
            for (field=0;field<2;field++)
            {
                FileName(sequenceName,frameCounter,fileName);
                FrameSave(fileName,quadrant,field);
                frameCounter++;
            }
            frameCounter+=2;
        }
        fg_sync(1);                     /* external sync */
        fg_sbuf(0);                     /* show incoming signal */

        /* now get odd numbered frames */

        frameCounter=temporaryFrameCounter+2;
```

```
        _clearscreen(_GCLEARSCREEN);
        printf("Searching for pulse %u\n",pulse);

        /* synchronize */

        SynchronizePulses();

        /* all synchronized now - next character will be first of a pair */

        while (1)
        {
            byte1=_bios_serialcom(_COM_RECEIVE,0,0);
            byte1=byte1&0x00ff;
            byte2=_bios_serialcom(_COM_RECEIVE,0,0);
            byte2=byte2&0x00ff;

            *lowByte=(unsigned char)byte1;
            *highByte=(unsigned char)byte2;

            if (pulse==pulseCounter) break;
            _settextposition(2,2);
            printf("%5u",pulseCounter);
        }

        /* this time through skip a frame here */

        while ((inp(STATUS)&ODDFIELD)!=0);
        while ((inp(STATUS)&ODDFIELD)==0);
        while ((inp(STATUS)&ODDFIELD)!=0);

        /* at right place now, so grab next 4 frames */
        /* making sure every other frame grabbed */

        _disable();                     /* interrupts off */
        fg_dquad(0);                    /* grab quadrant 0 */
        fg_snap(1);
        while(inp(VBLANK)==0);
        while(inp(VBLANK)!=0);
        while(inp(VBLANK)==0);
        fg_dquad(1);                    /* grab quadrant 1 */
        fg_snap(1);
        while(inp(VBLANK)==0);
        while(inp(VBLANK)!=0);
        while(inp(VBLANK)==0);
        fg_dquad(2);                    /* grab quadrant 2 */
        fg_snap(1);
        while(inp(VBLANK)==0);
        while(inp(VBLANK)!=0);
        while(inp(VBLANK)==0);
        fg_dquad(3);                    /* grab quadrant 3 */
        fg_snap(1);
        _enable();                      /* interrupts back on */

        /* write all eight field out to files */

        _settextposition(2,2);
        printf("%5u\n\n",pulseCounter);
        fg_sbuf(1);                     /* show buffer */
        fg_sync(0);                     /* internal sync */
        for (quadrant=0;quadrant<4;quadrant++)
        {
            fg_dquad(quadrant);
```

```
            for (field=0;field<2;field++)
            {
                FileName(sequenceName,frameCounter,fileName);
                FrameSave(fileName,quadrant,field);
                frameCounter++;
            }
            frameCounter+=2;
        }
        frameCounter-=2;                    /* correct frame count */
        fg_sync(1);                         /* external sync */
        fg_sbuf(0);                         /* show incoming signal */
    }
}

/* synchronize to sound pulses by checking for incrementing */

void SynchronizePulses()
{
    unsigned char *lowFirst,*highFirst;     /* pointers to first counter */
    unsigned char *lowSecond,*highSecond;   /* pointers to second counter */
    unsigned int first,second;              /* startup pulse counters */
    unsigned int byte1,byte2;               /* bytes read serially */

    SetupSerial();
    lowFirst=(unsigned char *)&first;
    highFirst=lowFirst+1;
    lowSecond=(unsigned char *)&second;
    highSecond=lowSecond+1;

    /* synchronize for data pairs */

    while (1)
    {

        /* get two bytes and see if incrementing normally */

        byte1=_bios_serialcom(_COM_RECEIVE,0,0);
        byte1=byte1&0x00ff;
        byte2=_bios_serialcom(_COM_RECEIVE,0,0);
        byte2=byte2&0x00ff;
        *lowFirst=(unsigned char)byte1;
        *highFirst=(unsigned char)byte2;

        byte1=_bios_serialcom(_COM_RECEIVE,0,0);
        byte1=byte1&0x00ff;
        byte2=_bios_serialcom(_COM_RECEIVE,0,0);
        byte2=byte2&0x00ff;
        *lowSecond=(unsigned char)byte1;
        *highSecond=(unsigned char)byte2;

        /* if not incrementing, skip one byte and try again */

        if (second!=first+1) _bios_serialcom(_COM_RECEIVE,0,0);
        else break;
    }
}

/* setup and initialize the matrox card */

void SetupMatrox()
{
    fg_inifmt(0x26c,1,0,0,1,0);             /* initialize card */
```

```
    fg_dquad(0);                                /* display quadrant 0 */
    fg_chan(2);                                 /* input channel 2 */
    fg_sync(1);                                 /* external synch */
    fg_quadm(1);                                /* set 4 quadrant mode */
    fg_autoset();                               /* set gain and offset */
    fg_sbuf(0);                                 /* display signal */
}

/* setup and initialize the serial port */

void SetupSerial()
{
    /* call bios routine */

    _bios_serialcom(_COM_INIT,0,_COM_CHR8|_COM_STOP1|_COM_EVENPARITY|
        _COM_600);
}

/***** NB Routines after this point may need tidying up *****/

/* produce general menu */

int Menu(title,prompts,items)
char title[];                                   /* title string */
char prompts[][STRINGSIZE];                    /* prompt strings */
int items;                      /* number of items */
{
    int i;                      /* item counter */
    int key;                  /* key pressed */
    int iret;                                   /* return value */
    int row;                                    /* row numbers */
    char buffer[STRINGSIZE];                   /* string buffer */
    int tab,len,max_len=0;              /* menu positioning */

    _clearscreen(_GCLEARSCREEN);
    _settextposition(0,(80-strlen(title))/2);
    _outtext(title);       /* print title */

    for (i=0;i<items;i++)
    {
        len=strlen(prompts[i]);
        if (len>max_len) max_len=len;
    }
    tab=(80-(max_len+4))/2;

    row=2+(23-items*2)/2;
    for (i=0;i<items;i++)               /* print out prompts */
    {
        _settextposition(row,tab);
        sprintf(buffer,"(%1i) %s",i+1,prompts[i]);
        _outtext(buffer);
        row+=2;
    }

    _settextposition(24,0);

    do                          /* get key press */
    {
        key=getch();
        iret=key-(int)'0';          /* convert to integer */
    } while (iret<1 || iret>items);
```

```
    return(iret);                   /* return value */
}

/* routine to set up user defined gain and offset */

void Controls()
{
    int iret;                   /* menu selection */
    static int offset=100;              /* offset */
    static int gain=155;               /* gain */
    static char menu1[][STRINGSIZE]=        /* menu 1 prompts */
    {
        "Increase Gain",
        "Decrease Gain",
        "Increase Offset",
        "Decrease Offset",
        "Draw Histogram",
        "Exit"
    };
    char buffer[STRINGSIZE];                /* character buffer */
    long max_value;
    long hist_buffer[256];

    strcpy(menu1[2],"Increase Offset ");
    sprintf(buffer,"(%3d)",offset);
    strcat(menu1[2],buffer);
    strcpy(menu1[0],"Increase Gain   ");
    sprintf(buffer,"(%3d)",255-gain);
    strcat(menu1[0],buffer);

    /* set offset/gain and grab picture */

    fg_offset(offset);
    fg_gain(gain);
    fg_dquad(0);                                /* quadrant 0 */
    fg_cgrab(1);                                /* continuous grabbing */
    fg_sbuf(1);                                 /* show grab buffer */

    do
    {
        iret=Menu("Control Options:",menu1,6);
        switch (iret)
        {
        case 1:
            gain-=5;
            if (gain<0) gain=0;
            fg_gain(gain);
            break;

        case 2:
            gain+=5;
            if (gain>255) gain=255;
            fg_gain(gain);
            break;

        case 3:
            offset+=5;
            if (offset>255) offset=255;
            fg_offset(offset);
            break;

        case 4:
```

```
        offset-=5;
        if (offset<0) offset=0;
        fg_offset(offset);
        break;

    case 5:
        fg_cgrab(0);
        fg_sync(0);
        FillIn();
        max_value=fg_histo(hist_buffer);
        fg_setind(0);
        fg_rectf(0,0,512,512);
        fg_dhisto(max_value,100,500,350,255,255,hist_buffer);
        _outtext("Press a key when ready");
        while (getch()==0);
        fg_sync(1);
        fg_cgrab(1);
        break;
    }
    strcpy(menu1[2],"Increase Offset ");
    sprintf(buffer,"(%3d)",offset);
    strcat(menu1[2],buffer);
    strcpy(menu1[0],"Increase Gain   ");
    sprintf(buffer,"(%3d)",255-gain);
    strcat(menu1[0],buffer);

} while (iret!=6);

    /* back to default */

    fg_sbuf(0);                                 /* external signal */
    fg_cgrab(0);                                /* continuous grab off */
}

/* frame save routine - heavily personalized for pip-1024 card */

void FrameSave(filename,quadrant,field)
char filename[];                                /* file name */
int quadrant;                                   /* sector number */
int field;                                      /* field number */
{
    FILE *unit;                 /* file unit */
    int rows=256;                               /* number of rows in picture */
    int columns=512;                            /* number of columns in picture */
    int irow;                   /* row counter */
    int ycount;                 /* y coordinate counter */
    char buffer[512];                           /* row buffer */
    struct visilogImageHeader fileHeader;       /* visilog file header */

    printf("Saving %s ....\n",filename);

    unit=fopen(filename,"wb");          /* open file */

    /* write out file header */

    fileHeader.magicNumber=VisilogConvert(0x6931);
    fileHeader.pixelsPerLine=VisilogConvert(columns);
    fileHeader.numberOfLines=VisilogConvert(rows);
    fileHeader.res1=VisilogConvert(1);
    fileHeader.res2=VisilogConvert(0);
    fileHeader.res3=VisilogConvert(0);
    fileHeader.gridType=VisilogConvert(1);                  /* rectangular */
```

```
    fileHeader.res4=VisilogConvert(0);
    fileHeader.arithmaticType=VisilogConvert(0x14);          /* integer */
    fileHeader.bitsPerPixel=VisilogConvert(8);
    fileHeader.res5=VisilogConvert(0);
    fileHeader.xOrigin=VisilogConvert(1);
    fileHeader.yOrigin=VisilogConvert(1);
    fileHeader.res6=VisilogConvert(1);
    fileHeader.res7=VisilogConvert(0);
    fileHeader.visilogHeaderSize=VisilogConvert(76);
    fileHeader.userHeaderSize=VisilogConvert(0);
    fileHeader.res8=VisilogConvert(0);
    fileHeader.totalHeaderSize=VisilogConvert(76);

    fwrite(&fileHeader,sizeof(fileHeader),1,unit);

    /* set up to read from right part of screen */

    ycount=field;

    for (irow=0;irow<rows;irow++)
    {
        fg_rowr(ycount,quadrant,buffer);/* read row into memory */
        fwrite(buffer,512,1,unit);    /* and write it to disk */
        ycount++;             /* increment ycount twice */
        ycount++;
    }

    fclose(unit);                 /* close file */
}

/* fill in second field lines with first field lines */

void FillIn()
{
    int rows=256;                  /* number of rows in picture */
    int columns=512;              /* number of columns in picture */
    int irow;                   /* row counter */
    int ycount;                  /* y coordinate counter */
    char buffer[512];             /* row buffer */

    ycount=0;
    for (irow=0;irow<rows;irow++)
    {
        fg_rowr(ycount,0,buffer);       /* read row into memory */
        ycount++;             /* increment ycount */
        fg_roww(ycount,0,buffer);    /* write over blank row */
        ycount++;             /* increment ycount */
    }
}

/* set up file name */

void FileName(seq_name,iframe,filename)
char seq_name[];                /* sequence name */
int iframe;                                  /* frame number */
char filename[];                /* file name */
{
    char string[4];                /* number string */

    strcpy(filename,seq_name);                /* copy to file name */
    if (strlen(filename)>8) filename[8]=0;  /* truncate if required */
```

```
    sprintf(string,"%3.3i",iframe);          /* convert number to string */

    strcat(filename,".");                    /* add number as extension */
    strcat(filename,string);
}
```

stretchpic

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* this program reads in a 512 by 256 pixel file and stetches it to 1024 by 768 */
/* visilog format */

#define XIN 512
#define YIN 256
#define XOUT 1024
#define YOUT 768
#define XFACT 2
#define YFACT 3
#define STRINGSIZE 128

main(nparms,parms)
int nparms;
char *parms[];
{
    FILE *unit;                          /* file pointer */
    char filename[STRINGSIZE];          /* filename */
    char buffer[YIN][XIN];              /* buffer for file input */
    char stretchBuffer[YOUT][XOUT];     /* buffer for file output */
    int x,y;                             /* counters */
    int nbytes;                          /* block counter */
    struct visilogImageHeader
    {
        long int magicNumber;
        long int pixelsPerLine;
        long int numberOfLines;
        long int res1;
        long int res2;
        long int res3;
        long int gridType;                      /* rectangular grid */
        long int res4;
        long int arithmaticType;                /* long integer image */
        long int bitsPerPixel;
        long int res5;
        long int xOrigin;
        long int yOrigin;
        long int res6;
        long int res7;
        long int visilogHeaderSize;
        long int userHeaderSize;
        long int res8;
        long int totalHeaderSize;
    } imageHeader;
    char tempFile[STRINGSIZE];           /* temporary file name */
    char command[STRINGSIZE];           /* command string */

    /* Open file and read image data */

    strcpy(filename,parms[1]);
    unit=fopen(filename,"r");
    if (unit==NULL)
    {
        printf("File %s not found\n",filename);
        exit(-1);
    }
```

```
/* get x and y range of input file */

fread(&imageHeader,sizeof(imageHeader),1,unit);
if (imageHeader.pixelsPerLine!=XIN || imageHeader.numberOfLines!=YIN)
{
    printf("Wrong picture size\n");
    exit(-1);
}

/* read data in */

nbytes=fread(buffer,XIN*YIN,1,unit);
fclose(unit);
if (nbytes!=1)
{
    printf("Error reading file %s\n",filename);
    exit(-1);
}

/* stretch file */

for (y=0;y<YIN;y++)
{
    for (x=0;x<XIN;x++)
    {
        stretchBuffer[y*YFACT][x*XFACT]=buffer[y][x];
        stretchBuffer[y*YFACT][x*XFACT+1]=buffer[y][x];
        stretchBuffer[y*YFACT+1][x*XFACT]=buffer[y][x];
        stretchBuffer[y*YFACT+1][x*XFACT+1]=buffer[y][x];
        stretchBuffer[y*YFACT+2][x*XFACT]=buffer[y][x];
        stretchBuffer[y*YFACT+2][x*XFACT+1]=buffer[y][x];
    }
}
imageHeader.pixelsPerLine=XOUT;
imageHeader.numberOfLines=YOUT;

strcpy(tempFile,tempnam(NULL,"pic"));
unit=fopen(tempFile,"w");
fwrite(&imageHeader,sizeof(imageHeader),1,unit);
fwrite(stretchBuffer,sizeof(stretchBuffer),1,unit);
fclose(unit);
strcpy(command,"mv ");
strcat(command,tempFile);
strcat(command," ");
strcat(command,filename);
strcat(command," &");
system(command);
}
```

Leaping Model

C Routines

Params.h

```
/* parameters file */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "MathConstants.h"

/* resource IDs */

#define MENU_BAR_ID 128                    /* main menu bar */

#define APPLE_MENU_ID 128                  /* apple menu */
#define FILE_MENU_ID 129                   /* file menu */
#define EDIT_MENU_ID 130                   /* edit menu */
#define MODEL_MENU_ID 131                  /* model menu */

#define ABOUT_ID 128                       /* about alert */
#define MESSAGE_ID 129                     /* general message alert */

#define MODEL_DEFINITION_DIALOG_ID 128     /* model definition dialog */
#define UNSAVED_DATA_DIALOG_ID 129         /* unsaved data dialog */
#define OPTIONS_DIALOG_ID 130              /* options dialog */
#define SEGMENTS_DIALOG_ID 131             /* segments dialog */
#define PROGRESS_DIALOG_ID 132             /* progress dialog */

/* menu items */

#define ABOUT_ITEM 1                       /* apple menu */

#define NEW_ITEM 1                         /* file menu */
#define OPEN_ITEM 2
#define SAVE_ITEM 4
#define SAVE_AS_ITEM 5
#define WRITE_ITEM 7
#define QUIT_ITEM 9

#define UNDO_ITEM 1                        /* edit menu */
#define CUT_ITEM 2
#define COPY_ITEM 3
#define PASTE_ITEM 4
#define CLEAR_ITEM 6

#define DEFINE_ITEM 1                      /* model menu */
#define OPTIONS_ITEM 2
#define SEGMENTS_ITEM 3

/* dialog items */

#define OK 1
#define CANCEL 2
```

```
#define MODEL_DEFINITION_MTJ_X 3    /* model definition dialog */
#define MODEL_DEFINITION_MTJ_Y 4
#define MODEL_DEFINITION_ANKLE_X 5
#define MODEL_DEFINITION_ANKLE_Y 6
#define MODEL_DEFINITION_KNEE_X 7
#define MODEL_DEFINITION_KNEE_Y 8
#define MODEL_DEFINITION_HIP_X 9
#define MODEL_DEFINITION_HIP_Y 10
#define MODEL_DEFINITION_NOSETIP_X 11
#define MODEL_DEFINITION_NOSETIP_Y 12

#define UNSAVED_DATA_NO_SAVE 3        /* unsaved data dialog */

#define OPTIONS_MASS 3                /* options dialog box */
#define OPTIONS_G 4
#define OPTIONS_TIME_TOLERANCE 5
#define OPTIONS_RANGE 6
#define OPTIONS_TIMES 7
#define OPTIONS_ITERATIONS 8
#define OPTIONS_EXTENSION_FRACTION 9

#define SEGMENTS_FOREFOOT_MASS 3    /* segments dialog box */
#define SEGMENTS_HINDFOOT_MASS 4
#define SEGMENTS_CALF_MASS 5
#define SEGMENTS_THIGH_MASS 6
#define SEGMENTS_TORSO_MASS 7
#define SEGMENTS_FOREFOOT_COM 8
#define SEGMENTS_HINDFOOT_COM 9
#define SEGMENTS_CALF_COM 10
#define SEGMENTS_THIGH_COM 11
#define SEGMENTS_TORSO_COM 12

#define PROGRESS_INDICATOR 3        /* progress dialog */

/* apple constants */

#define MIN_SLEEP 0L
#define NIL_MOUSE_REGION 0L
#define REMOVE_ALL_EVENTS 0
#define NULL_STRING "\p"
#define MOVE_TO_FRONT (WindowPtr)-1L

#define DIALOG_NULL_EVENT 1000       /* null event returned from dialog */

/* program parameters */

#define STRING_SIZE 128               /* string size */
#define MAX_RESULTS 100               /* maximum number of time intervals */
#define FILE_OWNER 'LM!!'            /* created file ownership */
#define FILE_TYPE 'LM!!'             /* binary file type */
#define FILE_DIALOG_X 75             /* position of file dialog box */
#define FILE_DIALOG_Y 75

/* type definitions */

typedef struct
{
    double x;
    double y;
} Coordinate;

typedef struct
```

```
{
    double x;
    double y;
    double theta;
    double r;
} Vector;

typedef struct
{
    Coordinate toeTip;
    Coordinate midTarsalJoint;
    Coordinate ankle;
    Coordinate knee;
    Coordinate hip;
    Coordinate noseTip;
    Coordinate foreFootCOM;
    Coordinate hindFootCOM;
    Coordinate calfCOM;
    Coordinate thighCOM;
    Coordinate torsoCOM;
    Coordinate bodyCOM;
} ModelCoordinates;

typedef struct
{
    double foreFoot;
    double hindFoot;
    double calf;
    double thigh;
    double torso;
} ModelCOMs;

typedef struct
{
    double foreFoot;
    double hindFoot;
    double calf;
    double thigh;
    double torso;
} ModelMass;

typedef struct
{
    Vector foreFoot;
    Vector hindFoot;
    Vector calf;
    Vector thigh;
    Vector torso;
} ModelVectors;

/* prototypes */

void AdjustMenus(void);
void HandleAppleChoice(short);
void HandleEditChoice(short);
void HandleFileChoice(short);
void HandleModelChoice(short);
void HandleMenuChoice(long);
void HandleMouseDown(void);
void HiLiteOK(DialogPtr);
short IsDAWindow(WindowPtr);
void main(void);
```

```
void MainLoop(void);
void MenuBarInit(void);
void ToolBoxInit(void);
void DialogInit(void);
void DialogEnd(void);
void OpenFile(AppFile *);
Boolean DefineModel(void);
Boolean Options(void);
Boolean Segments(void);
void GetModelSettings(void);
void SetModelSettings(void);
void GetOptionsSettings(void);
void SetOptionsSettings(void);
void GetSegmentsSettings(void);
void SetSegmentsSettings(void);
void New(void);
void SaveFile(void);
void SaveAs(void);
Boolean UnsavedData(void);
void Write(void);
Boolean Calculate(void);
double VectorLength(Vector);
double VectorAngle(Vector);
double VectorX(Vector);
double VectorY(Vector);
Boolean LengthFunction(double,int,double);
void WriteResults(char *,double);
void Rotate(Coordinate *,double);
void XAxisize(ModelCoordinates *);
void CentresOfMass(ModelCoordinates *);
void OpenFromDocument(void);
pascal Boolean NullEventFilter(DialogPtr,EventRecord *,short *);

/* globals */

extern Boolean gDone;                   /* done flag */
extern EventRecord gTheEvent;           /* event structure */
extern MenuHandle gAppleMenu;           /* menu handles */
extern MenuHandle gFileMenu;
extern MenuHandle gEditMenu;
extern MenuHandle gModelMenu;
extern DialogPtr gModelDefinitionDialog;    /* dialog pointer */
extern DialogPtr gUnsavedDataDialog;
extern DialogPtr gOptionsDialogBox;
extern DialogPtr gSegmentsDialog;
extern DialogPtr gProgressDialog;

extern Boolean gDefinitionOK;           /* model definition in memory */
extern Boolean gDefinitionToSave;       /* stuff to save flag */
extern SFReply gDefinitionFile;         /* definition file stuff */


extern ModelCoordinates gModel;         /* transformed model parameters */
extern ModelCoordinates gUserModel;     /* user input model parameters */
extern ModelCoordinates gResults[MAX_RESULTS];    /* model results */
extern ModelVectors gVectors;           /* model vectors */
extern ModelCOMs gCOMs;                 /* model COM's */
extern ModelMass gSegmentMass;          /* segment masses */
extern double gMass;                    /* animal mass */
extern double g;                        /* acceleration due to gravity */
extern double gTimeTolerance;           /* time tolerance */
extern int gNumberOfTimes;              /* number of time values */
```

```
extern int gMaxIterations;                  /* maximum number of iterations */
extern double gTimes[MAX_RESULTS];          /* times calculates */
extern double gRange;                      /* leap range */
extern double gFMax;                      /* maximum force */
extern double gSMax;                      /* maximum extension */
extern double gTMax;                      /* maximum time */
extern double gExtensionFraction;          /* fraction of max extension */
```

MathConstants.h

```
/* some useful constants */

#  define M_E            2.7182818284590452354
#  define M_LOG2E        1.4426950408889634074
#  define M_LOG10E    0.43429448190325182765
#  define M_LN2         0.69314718055994530942
#  define M_LN10         2.3025850929940456840
#  define M_PI        3.14159265358979323846
#  define M_PI_2         1.5707963267948966192
#  define M_PI_4         0.7853981633974483096
#  define M_1_PI         0.3183098861837906715
#  define M_2_PI         0.6366197723675813430
#  define M_2_SQRTPI     1.1283791670955125739
#  define M_SQRT2         1.4142135623730950488
#  define M_SQRT1_2     0.70710678118654752440
#  define MAXFLOAT     ((float)3.40282346638528860e+38)
```

AdjustMenus()

```
/* routine to set which menu options are available */

#include "Params.h"

void AdjustMenus()
{
    /* check for desk accessories */

    if (IsDAWindow(FrontWindow()))
    {
        EnableItem(gEditMenu,UNDO_ITEM);
        EnableItem(gEditMenu,CUT_ITEM);
        EnableItem(gEditMenu,COPY_ITEM);
        EnableItem(gEditMenu,PASTE_ITEM);
        EnableItem(gEditMenu,CLEAR_ITEM);
    }
    else
    {
        DisableItem(gEditMenu,UNDO_ITEM);
        DisableItem(gEditMenu,CUT_ITEM);
        DisableItem(gEditMenu,COPY_ITEM);
        DisableItem(gEditMenu,PASTE_ITEM);
        DisableItem(gEditMenu,CLEAR_ITEM);
    }

    /* check for data in memory */

    if (gDefinitionOK)
    {
        EnableItem(gFileMenu,WRITE_ITEM);
        DisableItem(gFileMenu,OPEN_ITEM);
```

```
    }
    else
    {
        DisableItem(gFileMenu,WRITE_ITEM);
        EnableItem(gFileMenu,OPEN_ITEM);
    }
}
```

Calculate()

```
#include "Params.h"

/* routine to perform the actual modelling calculations */

Boolean Calculate()
{
    short itemHit;                  /* item hit */
    short itemType;                  /* dummy item type */
    Rect itemRect;                  /* dummy item rect */
    Handle itemHandle;              /* item handle */
    double pValue;                  /* loop parameter */
    int pCounter;                     /* loop counter */
    double time;                    /* current time */
    double timeInterval;            /* time interval for results */
    double tolerance;                /* absolute tolerance value */
    char buffer[STRING_SIZE];       /* string buffer */

    /* put up percentage completed window */

    ShowWindow(gProgressDialog);

    /* correct for Body COM on x axis and calculate COMs */

    gModel=gUserModel;
    XAxisize(&gModel);

    /* calculate start vectors */

    gVectors.foreFoot.x=gModel.midTarsalJoint.x;
    gVectors.foreFoot.y=gModel.midTarsalJoint.y;
    gVectors.hindFoot.x=gModel.ankle.x-gModel.midTarsalJoint.x;
    gVectors.hindFoot.y=gModel.ankle.y-gModel.midTarsalJoint.y;
    gVectors.calf.x=gModel.knee.x-gModel.ankle.x;
    gVectors.calf.y=gModel.knee.y-gModel.ankle.y;
    gVectors.thigh.x=gModel.hip.x-gModel.knee.x;
    gVectors.thigh.y=gModel.hip.y-gModel.knee.y;
    gVectors.torso.x=gModel.noseTip.x-gModel.hip.x;
    gVectors.torso.y=gModel.noseTip.y-gModel.hip.y;

    /* calculate lengths */

    gVectors.foreFoot.r=VectorLength(gVectors.foreFoot);
    gVectors.hindFoot.r=VectorLength(gVectors.hindFoot);
    gVectors.calf.r=VectorLength(gVectors.calf);
    gVectors.thigh.r=VectorLength(gVectors.thigh);
    gVectors.torso.r=VectorLength(gVectors.torso);

    /* and angles */

    gVectors.foreFoot.theta=VectorAngle(gVectors.foreFoot);
    gVectors.hindFoot.theta=VectorAngle(gVectors.hindFoot);
```

```
gVectors.calf.theta=VectorAngle(gVectors.calf);
gVectors.thigh.theta=VectorAngle(gVectors.thigh);
gVectors.torso.theta=VectorAngle(gVectors.torso);

/* calculate start joint positions */

gResults[0].toeTip.x=0.0;
gResults[0].toeTip.y=0.0;
gResults[0].midTarsalJoint.x=gModel.midTarsalJoint.x;
gResults[0].midTarsalJoint.y=gModel.midTarsalJoint.y;
gResults[0].ankle.x=gModel.ankle.x;
gResults[0].ankle.y=gModel.ankle.y;
gResults[0].knee.x=gModel.knee.x;
gResults[0].knee.y=gModel.knee.y;
gResults[0].hip.x=gModel.hip.x;
gResults[0].hip.y=gModel.hip.y;
gResults[0].noseTip.x=gModel.noseTip.x;
gResults[0].noseTip.y=gModel.noseTip.y;

/* rotate to position Body COM on X axis and calculate COMs */

XAxisize(&gResults[0]);

/* now end joint positions */

gResults[gNumberOfTimes-1].toeTip.x=0.0;
gResults[gNumberOfTimes-1].toeTip.y=0.0;
gResults[gNumberOfTimes-1].midTarsalJoint.x=gVectors.foreFoot.r;
gResults[gNumberOfTimes-1].midTarsalJoint.y=0.0;
gResults[gNumberOfTimes-1].ankle.x=
    gResults[gNumberOfTimes-1].midTarsalJoint.x + gVectors.hindFoot.r;
gResults[gNumberOfTimes-1].ankle.y=0.0;
gResults[gNumberOfTimes-1].knee.x=
    gResults[gNumberOfTimes-1].ankle.x + gVectors.calf.r;
gResults[gNumberOfTimes-1].knee.y=0.0;
gResults[gNumberOfTimes-1].hip.x=
    gResults[gNumberOfTimes-1].knee.x + gVectors.thigh.r;
gResults[gNumberOfTimes-1].hip.y=0.0;
gResults[gNumberOfTimes-1].noseTip.x=
    gResults[gNumberOfTimes-1].hip.x + gVectors.torso.r;
gResults[gNumberOfTimes-1].noseTip.y=0.0;

/* rotate to position Body COM on X axis */

XAxisize(&gResults[gNumberOfTimes-1]);

/* calculate gFMax */

gSMax = gExtensionFraction * ( gResults[gNumberOfTimes-1].bodyCOM.x -
    gResults[0].bodyCOM.x );
gFMax = ( gMass * gRange * g ) / ( 2.0 * gSMax );
gTMax = sqrt( 2.0 * gMass * gSMax / gFMax );

timeInterval = gTMax / ( double ) ( gNumberOfTimes );
tolerance = timeInterval * gTimeTolerance;

/* time loop */

time=timeInterval;
for (pCounter=1;pCounter<gNumberOfTimes;pCounter++)
{
    GetDItem(gProgressDialog,PROGRESS_INDICATOR,&itemType,
```

```
        &itemHandle,&itemRect);
    sprintf(buffer,"%d out of %d times calculated",
        pCounter,gNumberOfTimes);
    SetIText(itemHandle,CtoPstr(buffer));

    ModalDialog(NullEventFilter,&itemHit);
    if (itemHit==CANCEL)
    {
        SysBeep(10);
        HideWindow(gProgressDialog);
        return(TRUE);
    }

    if (LengthFunction(time,pCounter,gTimeTolerance)==FALSE)
    {
        ParamText("\pLength function iteration failed to converge",
            NULL_STRING,NULL_STRING,NULL_STRING);
        NoteAlert(MESSAGE_ID,NULL);
    }
    time+=timeInterval;
}

GetDItem(gProgressDialog,PROGRESS_INDICATOR,&itemType,
    &itemHandle,&itemRect);
sprintf(buffer,"%d out of %d times calculated",
    pCounter,gNumberOfTimes);
SetIText(itemHandle,CtoPstr(buffer));

ModalDialog(NullEventFilter,&itemHit);
if (itemHit==CANCEL)
{
    SysBeep(10);
    HideWindow(gProgressDialog);
    return(TRUE);
}

/* rotate 45° */

for (pCounter=0;pCounter<gNumberOfTimes;pCounter++)
{
    Rotate(&gResults[pCounter].midTarsalJoint,M_PI_4);
    Rotate(&gResults[pCounter].ankle,M_PI_4);
    Rotate(&gResults[pCounter].knee,M_PI_4);
    Rotate(&gResults[pCounter].hip,M_PI_4);
    Rotate(&gResults[pCounter].noseTip,M_PI_4);

    /* and do times while we're at it */

    gTimes[pCounter]=timeInterval*(double)pCounter;

    }

    SysBeep(10);
    WriteResults("Test Result File",timeInterval);

    HideWindow(gProgressDialog);

    return(TRUE);
}
```

CentreOfMass()

```
/* routine to calculate centres of mass */

#include "Params.h"

void CentresOfMass(model)
ModelCoordinates *model;                    /* model */
{
    /* segment centres of mass */

model->foreFootCOM.x = gCOMs.foreFoot * ( model->midTarsalJoint.x -
    model->toeTip.x ) + model->toeTip.x;
model->foreFootCOM.y = gCOMs.foreFoot * ( model->midTarsalJoint.y -
    model->toeTip.y ) + model->toeTip.y;

model->hindFootCOM.x = gCOMs.hindFoot * ( model->ankle.x -
    model->midTarsalJoint.x ) + model->midTarsalJoint.x;
model->hindFootCOM.y = gCOMs.hindFoot * ( model->ankle.y -
    model->midTarsalJoint.y ) + model->midTarsalJoint.y;

model->calfCOM.x = gCOMs.calf * ( model->knee.x -
    model->ankle.x ) + model->ankle.x;
model->calfCOM.y = gCOMs.calf * ( model->knee.y -
    model->ankle.y ) + model->ankle.y;

model->thighCOM.x = gCOMs.thigh * ( model->hip.x -
    model->knee.x ) + model->knee.x;
model->thighCOM.y = gCOMs.thigh * ( model->hip.y -
    model->knee.y ) + model->knee.y;

model->torsoCOM.x = gCOMs.torso * ( model->noseTip.x -
    model->hip.x ) + model->hip.x;
model->torsoCOM.y = gCOMs.torso * ( model->noseTip.y -
    model->hip.y ) + model->hip.y;

/* overall centre of mass */

model->bodyCOM.x = ( model->foreFootCOM.x * gSegmentMass.foreFoot +
    model->hindFootCOM.x * gSegmentMass.hindFoot +
    model->calfCOM.x * gSegmentMass.calf +
    model->thighCOM.x * gSegmentMass.thigh +
    model->torsoCOM.x * gSegmentMass.torso ) /
    ( gSegmentMass.foreFoot + gSegmentMass.hindFoot +
    gSegmentMass.calf + gSegmentMass.thigh +
    gSegmentMass.torso );

model->bodyCOM.y = ( model->foreFootCOM.y * gSegmentMass.foreFoot +
    model->hindFootCOM.y * gSegmentMass.hindFoot +
    model->calfCOM.y * gSegmentMass.calf +
    model->thighCOM.y * gSegmentMass.thigh +
    model->torsoCOM.y * gSegmentMass.torso ) /
    ( gSegmentMass.foreFoot + gSegmentMass.hindFoot +
    gSegmentMass.calf + gSegmentMass.thigh +
    gSegmentMass.torso );

}
```

DefineModel()

```
/* routine to define the parameters of the leaping model */

#include "Params.h"
```

```
Boolean DefineModel()
{
    short itemHit;                          /* item hit value */

    /* show dialog window */

    SetModelSettings();
    ShowWindow(gModelDefinitionDialog);

    HiLiteOK(gModelDefinitionDialog);

  while (TRUE)
  {
      ModalDialog(NULL,&itemHit);
      switch(itemHit)
      {

      /* OK button */

      case (OK):
          HideWindow(gModelDefinitionDialog);
          GetModelSettings();
          gDefinitionToSave=TRUE;
          gDefinitionOK=TRUE;
          return(TRUE);
          break;

      /* Cancel button */


          case (CANCEL):
              HideWindow(gModelDefinitionDialog);
              return(FALSE);
              break;
          }
      }
}
```

DialogEnd()

```
/* routine to initialise dialog windows */

#include "Params.h"

void DialogEnd()
{
    DisposeDialog(gModelDefinitionDialog);
    DisposeDialog(gUnsavedDataDialog);
    DisposeDialog(gOptionsDialogBox);
    DisposeDialog(gSegmentsDialog);
    DisposeDialog(gProgressDialog);
}
```

DialogInit()

```
/* routine to initialise dialog windows */

#include "Params.h"

void DialogInit()
```

```
{
    gModelDefinitionDialog=GetNewDialog(MODEL_DEFINITION_DIALOG_ID,NULL,
        MOVE_TO_FRONT);
    gUnsavedDataDialog=GetNewDialog(UNSAVED_DATA_DIALOG_ID,NULL,
        MOVE_TO_FRONT);
    gOptionsDialogBox=GetNewDialog(OPTIONS_DIALOG_ID,NULL,
        MOVE_TO_FRONT);
    gSegmentsDialog=GetNewDialog(SEGMENTS_DIALOG_ID,NULL,
        MOVE_TO_FRONT);
    gProgressDialog=GetNewDialog(PROGRESS_DIALOG_ID,NULL,
        MOVE_TO_FRONT);
}
```

GetModelSettings()

```
/* routine to get the model parameters settings from the dialog box */

#include "Params.h"

void GetModelSettings()
{
    short itemType;                 /* dummy item type */
    Rect itemRect;                  /* dummy item rect */
    Handle itemHandle;              /* item handle */
    char buffer[STRING_SIZE];       /* string buffer */

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_MTJ_X,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.midTarsalJoint.x));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_MTJ_Y,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.midTarsalJoint.y));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_ANKLE_X,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.ankle.x));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_ANKLE_Y,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.ankle.y));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_KNEE_X,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.knee.x));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_KNEE_Y,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.knee.y));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_HIP_X,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.hip.x));
```

```
    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_HIP_Y,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.hip.y));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_NOSETIP_X,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.noseTip.x));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_NOSETIP_Y,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&(gUserModel.noseTip.y));

    /* calculate centres of mass */

    CentresOfMass(&gUserModel);
}
```

GetOptionSettings()

```
/* routine to get the options parameters settings from the dialog box */

#include "Params.h"

void GetOptionsSettings()
{
    short itemType;                     /* dummy item type */
    Rect itemRect;                      /* dummy item rect */
    Handle itemHandle;                  /* item handle */
    char buffer[STRING_SIZE];           /* string buffer */

    GetDItem(gOptionsDialogBox,OPTIONS_MASS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gMass);

    GetDItem(gOptionsDialogBox,OPTIONS_G,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&g);

    GetDItem(gOptionsDialogBox,OPTIONS_TIME_TOLERANCE,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gTimeTolerance);

    GetDItem(gOptionsDialogBox,OPTIONS_RANGE,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gRange);

    GetDItem(gOptionsDialogBox,OPTIONS_TIMES,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%d",&gNumberOfTimes);

    GetDItem(gOptionsDialogBox,OPTIONS_ITERATIONS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
```

```
    sscanf(PtoCstr(buffer),"%d",&gMaxIterations);

    GetDItem(gOptionsDialogBox,OPTIONS_EXTENSION_FRACTION,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gExtensionFraction);
}
```

GetSegmentSettings()

```
/* routine to get the model parameters settings from the dialog box */

#include "Params.h"

void GetSegmentsSettings()
{
    short itemType;                          /* dummy item type */
    Rect itemRect;                          /* dummy item rect */
    Handle itemHandle;                      /* item handle */
    char buffer[STRING_SIZE];              /* string buffer */

    GetDItem(gSegmentsDialog,SEGMENTS_FOREFOOT_MASS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gSegmentMass.foreFoot);

    GetDItem(gSegmentsDialog,SEGMENTS_HINDFOOT_MASS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gSegmentMass.hindFoot);

    GetDItem(gSegmentsDialog,SEGMENTS_CALF_MASS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gSegmentMass.calf);

    GetDItem(gSegmentsDialog,SEGMENTS_THIGH_MASS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gSegmentMass.thigh);

    GetDItem(gSegmentsDialog,SEGMENTS_TORSO_MASS,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gSegmentMass.torso);

    GetDItem(gSegmentsDialog,SEGMENTS_FOREFOOT_COM,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gCOMs.foreFoot);

    GetDItem(gSegmentsDialog,SEGMENTS_HINDFOOT_COM,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gCOMs.hindFoot);

    GetDItem(gSegmentsDialog,SEGMENTS_CALF_COM,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gCOMs.calf);
```

```
    GetDItem(gSegmentsDialog,SEGMENTS_THIGH_COM,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gCOMs.thigh);

    GetDItem(gSegmentsDialog,SEGMENTS_TORSO_COM,&itemType,
        &itemHandle,&itemRect);
    GetIText(itemHandle,buffer);
    sscanf(PtoCstr(buffer),"%lf",&gCOMs.torso);

}
```

HandleAppleChoice()

```
/* routine to handle choices in the apple menu */

#include "Params.h"

void HandleAppleChoice(theItem)
short theItem;
{
    Str255 accName;
    short accNumber;

    switch (theItem)
    {
    case ABOUT_ITEM:
        NoteAlert(ABOUT_ID,NULL);
        break;

    default:
        GetItem(gAppleMenu,theItem,accName);
        accNumber=OpenDeskAcc(accName);
        break;
    }
}
```

HandleEditChoice()

```
/* routine to handle edit menu choice */

#include "Params.h"

void HandleEditChoice(theItem)
short theItem;
{
    SystemEdit(theItem-1);
}


/* routine to handle choices from the file menu */

#include "Params.h"

void HandleFileChoice(theItem)
short theItem;
{
    switch (theItem)
    {
    case NEW_ITEM:
        if (UnsavedData())
```

```
        {
            New();
        }
        break;

    case OPEN_ITEM:
        OpenFile(NULL);
        break;

    case SAVE_ITEM:
        SaveFile();
        break;

    case SAVE_AS_ITEM:
        SaveAs();
        break;

    case WRITE_ITEM:
        Write();
        break;

    case QUIT_ITEM:
        if (UnsavedData())
        {
            gDone=TRUE;
        }
        break;
    }
}
```

HandleMenuChoice()

```
/* routine to handle menu choices */

#include "Params.h"

void HandleMenuChoice(menuChoice)
long menuChoice;
{
    short theMenu;
    short theItem;

    if (menuChoice!=0)
    {
        theMenu=HiWord(menuChoice);
        theItem=LoWord(menuChoice);
        switch (theMenu)
        {
        case APPLE_MENU_ID:
            HandleAppleChoice(theItem);
            break;

        case FILE_MENU_ID:
            HandleFileChoice(theItem);
            break;

        case EDIT_MENU_ID:
            HandleEditChoice(theItem);
            break;

        case MODEL_MENU_ID:
```

```
            HandleModelChoice(theItem);
            break;
        }
        HiliteMenu(0);
    }
}
HandleModelChoice()
/* routine to handle choices from the model menu */

#include "Params.h"

void HandleModelChoice(theItem)
short theItem;
{
    switch (theItem)
    {
    case DEFINE_ITEM:
        DefineModel();
        break;

    case OPTIONS_ITEM:
        Options();
        break;

    case SEGMENTS_ITEM:
        Segments();
        break;

    }
}
```

HandleMouseDown()

```
/* routine to handle mouse downs */

#include "Params.h"

void HandleMouseDown()
{
    WindowPtr whichWindow;
    short thePart;
    long menuChoice,windSize;

    thePart=FindWindow(gTheEvent.where,&whichWindow);
    switch (thePart)
    {
    case inMenuBar:
        AdjustMenus();
        menuChoice=MenuSelect(gTheEvent.where);
        HandleMenuChoice(menuChoice);
        break;

    case inSysWindow:
        SystemClick(&gTheEvent,whichWindow);
        break;
    }
}
```

HILiteOK()

```
/* routine to hilite the OK button on a standard modal dialog box */

#include "Params.h"

void HiLiteOK(dialogPointer)
DialogPtr dialogPointer;            /* pointer to dialog box for hilite */
{
    short itemType;                    /* item type */
    Handle item;                    /* item */
    Rect box;                        /* rect enclosing item */
    GrafPtr oldPort;

    /* get old drawing port */

    GetPort(&oldPort);

    /* set new port */

    SetPort(dialogPointer);

    /* get details about the OK button */

    GetDItem(dialogPointer,OK,&itemType,&item,&box);

    /* and do the hiliting */

    PenSize(3,3);
    InsetRect(&box,-4,-4);
    FrameRoundRect(&box,16,16);

    /* back to old port */

    SetPort(&oldPort);
}
```

IsDAWindow()

```
/* routine to test for DA Window */

#include "Params.h"

short IsDAWindow(whichWindow)
WindowPtr whichWindow;
{
    if (whichWindow==NULL) return (FALSE);
    else /* DA windows have negative windowKinds */
      return(((WindowPeek)whichWindow)->windowKind<0);
}
```

LengthFunction()

```
/* routine to calculate the positions of the joints at a given time */

#include "Params.h"
#define DEBUG 0

Boolean LengthFunction(goalTime,pCounter,tolerance)
double goalTime;                /* time required for distance calculation */
int pCounter;                    /* result counter */
double tolerance;                /* absolute time tolerance */
```

```
{
    double pValue=0.5;              /* initial value of p */
    double deltaP=0.25;              /* initial p change value */
    int loopCounter=0;              /* iteration counter */
    double time;                   /* calculated time from p */
    ModelVectors vectors;          /* intermediate vectors */

    /* range checking */

    if (goalTime<=0.0)
    {
        goalTime=0.0;
        pValue=1.0;
    }

    if (goalTime>=gTMax)
    {
        goalTime=gTMax;
        pValue=0.0;
    }

    /* loop till goalTime is within tolerance of actual time */

    while (TRUE)
    {

        /* count interation */

        loopCounter++;

        /* calculate vectors */

        vectors.foreFoot.theta = pValue * gVectors.foreFoot.theta;
        vectors.foreFoot.r = gVectors.foreFoot.r;
        vectors.foreFoot.x = VectorX(vectors.foreFoot);
        vectors.foreFoot.y = VectorY(vectors.foreFoot);

        vectors.hindFoot.theta = pValue * gVectors.hindFoot.theta;
        vectors.hindFoot.r = gVectors.hindFoot.r;
        vectors.hindFoot.x = VectorX(vectors.hindFoot);
        vectors.hindFoot.y = VectorY(vectors.hindFoot);

        vectors.calf.theta = pValue * gVectors.calf.theta;
        vectors.calf.r = gVectors.calf.r;
        vectors.calf.x = VectorX(vectors.calf);
        vectors.calf.y = VectorY(vectors.calf);

        vectors.thigh.theta = pValue * gVectors.thigh.theta;
        vectors.thigh.r = gVectors.thigh.r;
        vectors.thigh.x = VectorX(vectors.thigh);
        vectors.thigh.y = VectorY(vectors.thigh);

        vectors.torso.theta = pValue * gVectors.torso.theta;
        vectors.torso.r = gVectors.torso.r;
        vectors.torso.x = VectorX(vectors.torso);
        vectors.torso.y = VectorY(vectors.torso);

        /* calculate joint positions */

        gResults[pCounter].toeTip.x=0.0;
        gResults[pCounter].toeTip.y=0.0;
```

```
        gResults[pCounter].midTarsalJoint.x=
            gResults[pCounter].toeTip.x+
            vectors.foreFoot.x;
        gResults[pCounter].midTarsalJoint.y=
            gResults[pCounter].toeTip.y+
            vectors.foreFoot.y;

        gResults[pCounter].ankle.x=
            gResults[pCounter].midTarsalJoint.x+
            vectors.hindFoot.x;
        gResults[pCounter].ankle.y=
            gResults[pCounter].midTarsalJoint.y+
            vectors.hindFoot.y;

        gResults[pCounter].knee.x=
            gResults[pCounter].ankle.x+
            vectors.calf.x;
        gResults[pCounter].knee.y=
            gResults[pCounter].ankle.y+
            vectors.calf.y;

        gResults[pCounter].hip.x=
            gResults[pCounter].knee.x+
            vectors.thigh.x;
        gResults[pCounter].hip.y=
            gResults[pCounter].knee.y+
            vectors.thigh.y;

        gResults[pCounter].noseTip.x=
            gResults[pCounter].hip.x+
            vectors.torso.x;
        gResults[pCounter].noseTip.y=
            gResults[pCounter].hip.y+
            vectors.torso.y;

        /* rotate and align with x axis */

        XAxisize(&gResults[pCounter]);

        /* calculate time from position of COM */

        time=sqrt(2.0 * gMass * (gResults[pCounter].bodyCOM.x -
            gResults[0].bodyCOM.x) / gFMax);

#if DEBUG
        printf("p = %.12lf  time = %.12lf\n",pValue,time);
#endif

        /* check to see if close enough to required value to do */

        if (fabs(time-goalTime)<tolerance) return(TRUE);
        else
        {
            if (time>goalTime) pValue+=deltaP;
            else pValue-=deltaP;

            if (pValue>1.0) pValue=1.0;
            if (pValue<0.0) pValue=0.0;

            deltaP*=0.5;
        }
```

```
        /* check iteration count */

        if (loopCounter>gMaxIterations) return(FALSE);
    }
}
```

main()

```
/* main routine */

#include "Params.h"

/* globals */

Boolean gDone;                  /* done flag */
EventRecord gTheEvent;          /* event structure */
MenuHandle gAppleMenu;          /* menu handles */
MenuHandle gFileMenu;
MenuHandle gEditMenu;
MenuHandle gModelMenu;
DialogPtr gModelDefinitionDialog;    /* dialog pointer */
DialogPtr gUnsavedDataDialog;
DialogPtr gOptionsDialogBox;
DialogPtr gSegmentsDialog;
DialogPtr gProgressDialog;

Boolean gDefinitionOK=FALSE;        /* model definition in memory */
Boolean gDefinitionToSave=FALSE;    /* stuff to save flag */
SFReply gDefinitionFile;            /* definition file stuff */

ModelCoordinates gModel;                    /* transformed model parameters */
ModelCoordinates gUserModel;                /* user input model parameters */
ModelCoordinates gResults[MAX_RESULTS];      /* model results */
ModelVectors gVectors;                       /* model vectors */
ModelCOMs gCOMs;                            /* model COM's */
ModelMass gSegmentMass;                      /* segment masses */
double gMass;                               /* animal mass */
int gNumberOfTimes;                          /* time increment */
int gMaxIterations;                          /* maximum number of iterations */
double g;                                   /* acceleration due to gravity */
double gTimeTolerance;                       /* fractional time tolerance */
double gTimes[MAX_RESULTS];                  /* times calculates */
double gRange;                               /* leap range */
double gFMax;                               /* maximum force */
double gSMax;                               /* maximum extension */
double gTMax;                               /* maximum time */
double gExtensionFraction;                   /* fraction of max extension */

void main()
{
    ToolBoxInit();
    MenuBarInit();
    DialogInit();
    New();
    OpenFromDocument();

    MainLoop();

    exit(0);
}
```

```
/* main program loop */

#include "Params.h"

void MainLoop()
{
    char theChar;                       /* key character */

    gDone=FALSE;
    while(gDone==FALSE)
    {

        /* get events */

        WaitNextEvent(everyEvent,&gTheEvent,MIN_SLEEP,NIL_MOUSE_REGION);

        /* and act on it */

        switch (gTheEvent.what)
        {
        case mouseDown:
            HandleMouseDown();
            break;

        case keyDown:
        case autoKey:
            theChar=gTheEvent.message & charCodeMask;
            if ((gTheEvent.modifiers & cmdKey)!=0)
            {
                AdjustMenus();
                HandleMenuChoice(MenuKey(theChar));
            }
            break;

        case updateEvt:
            break;
        }
    }
}
```

MenuBarInit()

```
/* routine to set up menu bar */

#include "Params.h"

void MenuBarInit()
{
    Handle myMenuBar;

    myMenuBar=GetNewMBar(MENU_BAR_ID);
    SetMenuBar(myMenuBar);

    gAppleMenu=GetMHandle(APPLE_MENU_ID);
    AddResMenu(gAppleMenu,'DRVR');

    gFileMenu=GetMHandle(FILE_MENU_ID);
    gEditMenu=GetMHandle(EDIT_MENU_ID);
    gModelMenu=GetMHandle(MODEL_MENU_ID);

    DrawMenuBar();
```

```
}
```

New()

```
/* routine to reset globals to their startup values */

#include "Params.h"

void New()
{
    /* reset the dialog boxes */

    DialogEnd();
    DialogInit();

    /* model */

    GetModelSettings();

    /* definition file status */

    gDefinitionOK=FALSE;
    gDefinitionToSave=FALSE;
    gDefinitionFile.fName[0]=0;

    /* options */

    GetOptionsSettings();

    /* segments */

    GetSegmentsSettings();
}
```

NullEventsFilter()

```
/* dialog filter routine to return NULL EVENTS */

#include "Params.h"

pascal Boolean NullEventFilter(theDialog,theEvent,itemHit)
DialogPtr theDialog;                /* calling dialog */
EventRecord *theEvent;                /* event returned */
short *itemHit;                        /* item hit return code */
{
    if (theEvent->what==nullEvent)
    {
        *itemHit=DIALOG_NULL_EVENT;
        return(TRUE);
    }

    return(FALSE);
}
```

OpenFile()

```
/* routine to open an existing model definition file */

#include "Params.h"
```

```
void OpenFile(theFile)
AppFile *theFile;                          /* pointer to file */
{
    Point myPoint;                          /* position of dialog */
    SFTypeList typeList;                   /* type selection */
    short numTypes;                         /* number of acceptable filetypes */
    short refNum;                          /* file reference number */
    long numBytes;                          /* number of bytes to read */

    /* check to see if passed a file */

    if (theFile==NULL)
    {
    myPoint.h=FILE_DIALOG_X;
    myPoint.v=FILE_DIALOG_Y;
    numTypes=1;
    typeList[0]=FILE_TYPE;
    SFGetFile(myPoint,NULL_STRING,NULL,numTypes,typeList,NULL,
        &gDefinitionFile);
    }
    else
    {
        gDefinitionFile.good=TRUE;
        strcpy((char *)gDefinitionFile.fName,(char *)theFile->fName);
        gDefinitionFile.vRefNum=theFile->vRefNum;
    }

    /* process gDefinitionFile */

    if (gDefinitionFile.good==TRUE)
    {
        gDefinitionToSave=FALSE;
        gDefinitionOK=TRUE;

        /* open and read file */

        FSOpen(gDefinitionFile.fName,gDefinitionFile.vRefNum,&refNum);

        numBytes=(long)sizeof(gUserModel);
        FSRead(refNum,&numBytes,&gUserModel);

        numBytes=(long)sizeof(gSegmentMass);
        FSRead(refNum,&numBytes,(char *)&gSegmentMass);

        numBytes=(long)sizeof(gCOMs);
        FSRead(refNum,&numBytes,(char *)&gCOMs);

        numBytes=(long)sizeof(gMass);
        FSRead(refNum,&numBytes,(char *)&gMass);

        numBytes=(long)sizeof(g);
        FSRead(refNum,&numBytes,(char *)&g);

        numBytes=(long)sizeof(gTimeTolerance);
        FSRead(refNum,&numBytes,(char *)&gTimeTolerance);

        numBytes=(long)sizeof(gRange);
        FSRead(refNum,&numBytes,(char *)&gRange);

        numBytes=(long)sizeof(gNumberOfTimes);
        FSRead(refNum,&numBytes,(char *)&gNumberOfTimes);
```

```
        numBytes=(long)sizeof(gMaxIterations);
        FSRead(refNum,&numBytes,(char *)&gMaxIterations);

        numBytes=(long)sizeof(gExtensionFraction);
        FSRead(refNum,&numBytes,(char *)&gExtensionFraction);

        /* close file */

        FSClose(refNum);
    }
}
```

OpenFromDocument()

```
/* opens a file that has been selected by the finder */

#include "Params.h"

void OpenFromDocument()
{
    short message;          /* document count message */
    short count;            /* document count number */
    AppFile theFile;        /* file selected */

    /* count documents */

    CountAppFiles(&message,&count);

    /* check count, and whether print requested (not available) */

    if (count==0) return;
    if (message==appPrint) exit(0);

    /* get first file only (and then only if type FILE_TYPE) */

    GetAppFiles(1,&theFile);
    ClrAppFiles(1);

    if (theFile.fType!=FILE_TYPE) return;

    OpenFile(&theFile);
}
```

Options()

```
/* routine to set the modelling options */

#include "Params.h"

Boolean Options()
{
    short itemHit;                          /* item hit value */

    /* show dialog window */

    SetOptionsSettings();
    ShowWindow(gOptionsDialogBox);

    HiLiteOK(gOptionsDialogBox);
```

```
while (TRUE)
{
    ModalDialog(NULL,&itemHit);
    switch(itemHit)
    {

    /* OK button */

    case (OK):
        HideWindow(gOptionsDialogBox);
        GetOptionsSettings();
        gDefinitionToSave=TRUE;
        return(TRUE);
        break;

    /* Cancel button */

    case (CANCEL):
        HideWindow(gOptionsDialogBox);
        return(FALSE);
        break;
    }
  }
}
```

Rotate()

```
/* routine to do a 2D rotation by an arbitrary amount to a coordinate */

#include "Params.h"

void Rotate(point,angle)
Coordinate *point;                  /* point to be rotated */
double angle;                      /* angle to rotate (radians) */
{
    double x,y;                     /* temporary storage for new angle */
    double sinAngle,cosAngle;      /* temporary storage of the trig stuff */

    /* do trig */

    sinAngle=sin(angle);
    cosAngle=cos(angle);

    /* do matrix rotation stuff */

    x=point->x*cosAngle-point->y*sinAngle;
    y=point->x*sinAngle+point->y*cosAngle;

    point->x=x;
    point->y=y;

}
```

SaveAs()

```
/* routine to perform named save function */

#include "Params.h"

void SaveAs()
```

```
{
    SFReply reply;                          /* reply from file dialog */
    Point myPoint;                          /* position of dialog */
    short refNum;                          /* file reference number */
    char prompt[STRING_SIZE];              /* prompt file name */
    long numBytes;                          /* number of bytes to write */

    if (gDefinitionFile.fName[0]==0)
    {
        strcpy(prompt,(char *)"\pUntitled");
    }
    else
    {
        strcpy(prompt,(char *)gDefinitionFile.fName);
    }

    myPoint.h=FILE_DIALOG_X;
    myPoint.v=FILE_DIALOG_Y;
    SFPutFile(myPoint,NULL_STRING,prompt,NULL,&reply);

    if (reply.good==TRUE)
    {
        strcpy((char *)gDefinitionFile.fName,(char *)reply.fName);
        gDefinitionFile.vRefNum=reply.vRefNum;

        /* create and open file */

        FSDelete(gDefinitionFile.fName,gDefinitionFile.vRefNum);
        Create(gDefinitionFile.fName,gDefinitionFile.vRefNum,
            FILE_OWNER,FILE_TYPE);
        FSOpen(gDefinitionFile.fName,gDefinitionFile.vRefNum,&refNum);

        /* write out data */

        numBytes=(long)sizeof(gUserModel);
        FSWrite(refNum,&numBytes,(char *)&gUserModel);

        numBytes=(long)sizeof(gSegmentMass);
        FSWrite(refNum,&numBytes,(char *)&gSegmentMass);

        numBytes=(long)sizeof(gCOMs);
        FSWrite(refNum,&numBytes,(char *)&gCOMs);

        numBytes=(long)sizeof(gMass);
        FSWrite(refNum,&numBytes,(char *)&gMass);

        numBytes=(long)sizeof(g);
        FSWrite(refNum,&numBytes,(char *)&g);

        numBytes=(long)sizeof(gTimeTolerance);
        FSWrite(refNum,&numBytes,(char *)&gTimeTolerance);

        numBytes=(long)sizeof(gRange);
        FSWrite(refNum,&numBytes,(char *)&gRange);

        numBytes=(long)sizeof(gNumberOfTimes);
        FSWrite(refNum,&numBytes,(char *)&gNumberOfTimes);

        numBytes=(long)sizeof(gMaxIterations);
        FSWrite(refNum,&numBytes,(char *)&gMaxIterations);

        numBytes=(long)sizeof(gExtensionFraction);
```

```
        FSWrite(refNum,&numBytes,(char *)&gExtensionFraction);

        /* close file */

        FSClose(refNum);
        gDefinitionToSave=FALSE;
    }
}
```

SaveFile()

```
/* routine to perform un-named save function */

#include "Params.h"

void SaveFile()
{
    short refNum;                       /* file reference number */
    long numBytes;                       /* number of bytes to write */
    char buffer[STRING_SIZE*5];           /* buffer space */

    if (gDefinitionFile.fName[0]==0)
    {
        SaveAs();
    }
    else
    {
        FSOpen(gDefinitionFile.fName,gDefinitionFile.vRefNum,&refNum);

        /* write out data */

        numBytes=(long)sizeof(gUserModel);
        FSWrite(refNum,&numBytes,(char *)&gUserModel);

        numBytes=(long)sizeof(gSegmentMass);
        FSWrite(refNum,&numBytes,(char *)&gSegmentMass);

        numBytes=(long)sizeof(gCOMs);
        FSWrite(refNum,&numBytes,(char *)&gCOMs);

        numBytes=(long)sizeof(gMass);
        FSWrite(refNum,&numBytes,(char *)&gMass);

        numBytes=(long)sizeof(g);
        FSWrite(refNum,&numBytes,(char *)&g);

        numBytes=(long)sizeof(gTimeTolerance);
        FSWrite(refNum,&numBytes,(char *)&gTimeTolerance);

        numBytes=(long)sizeof(gRange);
        FSWrite(refNum,&numBytes,(char *)&gRange);

        numBytes=(long)sizeof(gNumberOfTimes);
        FSWrite(refNum,&numBytes,(char *)&gNumberOfTimes);

        numBytes=(long)sizeof(gMaxIterations);
        FSWrite(refNum,&numBytes,(char *)&gMaxIterations);

        numBytes=(long)sizeof(gExtensionFraction);
        FSWrite(refNum,&numBytes,(char *)&gExtensionFraction);
```

```
        /* close file */

        FSClose(refNum);
        gDefinitionToSave=FALSE;
    }
}
```

Segments()

```
/* routine to set the modelling options */

#include "Params.h"

Boolean Segments()
{
    short itemHit;                          /* item hit value */

    /* show dialog window */

    SetSegmentsSettings();
    ShowWindow(gSegmentsDialog);

    HiLiteOK(gSegmentsDialog);

    while (TRUE)
    {
        ModalDialog(NULL,&itemHit);
        switch(itemHit)
        {

        /* OK button */

        case (OK):
            HideWindow(gSegmentsDialog);
            GetSegmentsSettings();
            gDefinitionToSave=TRUE;
            return(TRUE);
            break;

        /* Cancel button */

        case (CANCEL):
            HideWindow(gSegmentsDialog);
            return(FALSE);
            break;
        }
    }
}
```

SetModelSettings()

```
/* routine to set the model parameters settings for the dialog box */

#include "Params.h"

void SetModelSettings()
{
    short itemType;                         /* dummy item type */
    Rect itemRect;                          /* dummy item rect */
    Handle itemHandle;                      /* item handle */
```

```
    char buffer[STRING_SIZE];              /* character buffer */

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_MTJ_X,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.midTarsalJoint.x);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_MTJ_Y,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.midTarsalJoint.y);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_ANKLE_X,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.ankle.x);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_ANKLE_Y,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.ankle.y);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_KNEE_X,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.knee.x);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_KNEE_Y,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.knee.y);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_HIP_X,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.hip.x);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_HIP_Y,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.hip.y);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_NOSETIP_X,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.noseTip.x);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gModelDefinitionDialog,MODEL_DEFINITION_NOSETIP_Y,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gUserModel.noseTip.y);
    SetIText(itemHandle,CtoPstr(buffer));

}
```

SetOptionsSettings()

```
/* routine to set the options parameters settings for the dialog box */

#include "Params.h"

void SetOptionsSettings()
```

```
{
    short itemType;                          /* dummy item type */
    Rect itemRect;                           /* dummy item rect */
    Handle itemHandle;                       /* item handle */

    char buffer[STRING_SIZE];                /* character buffer */

    GetDItem(gOptionsDialogBox,OPTIONS_MASS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gMass);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gOptionsDialogBox,OPTIONS_G,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",g);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gOptionsDialogBox,OPTIONS_TIME_TOLERANCE,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gTimeTolerance);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gOptionsDialogBox,OPTIONS_RANGE,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gRange);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gOptionsDialogBox,OPTIONS_TIMES,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%d",gNumberOfTimes);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gOptionsDialogBox,OPTIONS_ITERATIONS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%d",gMaxIterations);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gOptionsDialogBox,OPTIONS_EXTENSION_FRACTION,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gExtensionFraction);
    SetIText(itemHandle,CtoPstr(buffer));
}
```

SetSegmentsSettings()

```
/* routine to set the model parameters settings for the dialog box */

#include "Params.h"

void SetSegmentsSettings()
{
    short itemType;                          /* dummy item type */
    Rect itemRect;                           /* dummy item rect */
    Handle itemHandle;                       /* item handle */
    char buffer[STRING_SIZE];                /* character buffer */

    GetDItem(gSegmentsDialog,SEGMENTS_FOREFOOT_MASS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gSegmentMass.foreFoot);
    SetIText(itemHandle,CtoPstr(buffer));
```

```
    GetDItem(gSegmentsDialog,SEGMENTS_HINDFOOT_MASS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gSegmentMass.hindFoot);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_CALF_MASS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gSegmentMass.calf);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_THIGH_MASS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gSegmentMass.thigh);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_TORSO_MASS,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gSegmentMass.torso);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_FOREFOOT_COM,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gCOMs.foreFoot);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_HINDFOOT_COM,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gCOMs.hindFoot);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_CALF_COM,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gCOMs.calf);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_THIGH_COM,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gCOMs.thigh);
    SetIText(itemHandle,CtoPstr(buffer));

    GetDItem(gSegmentsDialog,SEGMENTS_TORSO_COM,&itemType,
        &itemHandle,&itemRect);
    sprintf(buffer,"%lg",gCOMs.torso);
    SetIText(itemHandle,CtoPstr(buffer));
}
```

ToolBoxInit()

```
/* Routine to initialize various toolbox managers */

#include "Params.h"

void ToolBoxInit()
{
    InitGraf(&thePort);
    InitFonts();
    FlushEvents(everyEvent,REMOVE_ALL_EVENTS);
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NULL);
```

```
    InitCursor();
}
```

UnsavedData()

```
/* routine to check for and save any unsaved model data file */

#include "Params.h"

Boolean UnsavedData()
{
    Boolean dialogDone=FALSE;              /* dialog done flag */
    short itemHit;                          /* item hit value */
    Boolean returnCode=TRUE;                  /* return code */

    /* test for unsaved data */

    if (gDefinitionToSave!=TRUE) return(returnCode);

    /* show dialog window */

    ShowWindow(gUnsavedDataDialog);
    HiLiteOK(gUnsavedDataDialog);

    while (dialogDone==FALSE)
    {
        ModalDialog(NULL,&itemHit);
        switch(itemHit)
        {

        /* Save button */

        case (OK):
            HideWindow(gUnsavedDataDialog);
            SaveFile();
            dialogDone=TRUE;
            break;

        /* Don't Save button */

        case (UNSAVED_DATA_NO_SAVE):
            HideWindow(gUnsavedDataDialog);
            dialogDone=TRUE;
            break;

        /* Cancel button */

        case (CANCEL):
            HideWindow(gUnsavedDataDialog);
            dialogDone=TRUE;
            returnCode=FALSE;
            break;
        }
    }

    return (returnCode);
}
```

VectorAngle()

```
/* routine to calculate angles */

#include "Params.h"

double VectorAngle(vector)
Vector vector;
{
    double angle;

    angle=atan2(vector.y,vector.x);
    return(angle);
}
```

VectorLength()

```
/* routine to calculate lengths */

#include "Params.h"

double VectorLength(vector)
Vector vector;
{
    double length;

    length=sqrt(vector.x*vector.x+vector.y*vector.y);
    return(length);
}
```

VectorX()

```
/* routine to calculate x part of vector */

#include "Params.h"

double VectorX(vector)
Vector vector;
{
    double x;

    x = vector.r * cos(vector.theta);
    return(x);
}
```

VectorY()

```
/* routine to calculate y part of vector */

#include "Params.h"

double VectorY(vector)
Vector vector;
{
    double y;

    y = vector.r * sin(vector.theta);
    return(y);
}
```

Write()

```
/* routine to write out data */

#include "Params.h"

void Write()
{
    Calculate();
}
```

WriteResults()

```
/* this routine writes out the node position file */

#include"Params.h"

void WriteResults(title,frameInterval)

char title[STRING_SIZE];                    /* file title line */
double frameInterval;                        /* the interval between frames */

{
    int frameCounter;                       /* counter frame number */
    int nodeCounter;                        /* counter node number */
    SFReply reply;                          /* reply from file dialog */
    Point myPoint;                          /* position of dialog */
    short refNum;                           /* file reference number */
    char prompt[STRING_SIZE];               /* prompt file name */
    char buffer[STRING_SIZE];               /* file buffer */
    long numBytes;                          /* number of bytes to write */

    if (gDefinitionFile.fName[0]==0)
    {
        strcpy(prompt,(char *)"\pUntitled.node");
    }
    else
    {
        strcpy(prompt,(char *)gDefinitionFile.fName);
        PtoCstr(prompt);
        strcat(prompt,".node");
        CtoPstr(prompt);
    }

    myPoint.h=FILE_DIALOG_X;
    myPoint.v=FILE_DIALOG_X;
    SFPutFile(myPoint,NULL_STRING,prompt,NULL,&reply);

    if (reply.good==TRUE)
    {
        /* create and open file */

        FSDelete(reply.fName,reply.vRefNum);
        Create(reply.fName,reply.vRefNum,FILE_OWNER,'TEXT');
        FSOpen(reply.fName,reply.vRefNum,&refNum);

        /* write data */

        sprintf(buffer,"%s\r",title);
        numBytes=(long)strlen(buffer);
```

```
FSWrite(refNum,&numBytes,buffer);
sprintf(buffer,"%.12le\r",frameInterval);
numBytes=(long)strlen(buffer);
FSWrite(refNum,&numBytes,buffer);
sprintf(buffer,"%d\r",gNumberOfTimes);
numBytes=(long)strlen(buffer);
FSWrite(refNum,&numBytes,buffer);

for (frameCounter=0;frameCounter<gNumberOfTimes;frameCounter++)
{
    sprintf(buffer,"%d\r",frameCounter);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);
    sprintf(buffer,"%d\r",6);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);

    sprintf(buffer,"%d %.12le %.12le %.12le\r",0,0.0,0.0,0.0);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);
    sprintf(buffer,"Toe Tip\r");
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);

    sprintf(buffer,"%d %.12le %.12le %.12le\r",1,
        gResults[frameCounter].midTarsalJoint.x,
        gResults[frameCounter].midTarsalJoint.y,
        0.0);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);
    sprintf(buffer,"Mid-tarsal Joint\r");
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);

    sprintf(buffer,"%d %.12le %.12le %.12le\r",2,
        gResults[frameCounter].ankle.x,
        gResults[frameCounter].ankle.y,
        0.0);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);
    sprintf(buffer,"Ankle\r");
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);

    sprintf(buffer,"%d %.12le %.12le %.12le\r",3,
        gResults[frameCounter].knee.x,
        gResults[frameCounter].knee.y,
        0.0);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);
    sprintf(buffer,"Knee\r");
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);

    sprintf(buffer,"%d %.12le %.12le %.12le\r",4,
        gResults[frameCounter].hip.x,
        gResults[frameCounter].hip.y,
        0.0);
    numBytes=(long)strlen(buffer);
    FSWrite(refNum,&numBytes,buffer);
    sprintf(buffer,"Hip\r");
    numBytes=(long)strlen(buffer);
```

```
            FSWrite(refNum,&numBytes,buffer);

            sprintf(buffer,"%d %.12le %.12le %.12le\r",5,
                gResults[frameCounter].noseTip.x,
                gResults[frameCounter].noseTip.y,
                0.0);
            numBytes=(long)strlen(buffer);
            FSWrite(refNum,&numBytes,buffer);
            sprintf(buffer,"Nose Tip\r");
            numBytes=(long)strlen(buffer);
            FSWrite(refNum,&numBytes,buffer);
        }
    }

    /* close file */

    FSClose(refNum);

    /* finished */

}
```

XAxisize()

```
/* routine to rotate model so that body COM is on the x axis */

#include "Params.h"

void XAxisize(model)
ModelCoordinates *model;            /* model */
{
    double angle;                    /* rotation angle */

    /* calculate centres of mass */

    CentresOfMass(model);

    /* find angle of body COM off centre */

    angle=atan2(model->bodyCOM.y,model->bodyCOM.x);
    angle*=(-1);

    /* now perform all the rotations */

    Rotate(&model->midTarsalJoint,angle);
    Rotate(&model->ankle,angle);
    Rotate(&model->knee,angle);
    Rotate(&model->hip,angle);
    Rotate(&model->noseTip,angle);

    /* and redo centres of mass */

    CentresOfMass(model);

}
```

References

1. Abbott B. C.; Bigland B. The effects of force and speed changes on the rate of oxygen consumption during negative work. J. Physiol., Lond. 1953; 120: 319-325.

2. ADAMS. 1.1. : Mechanical Dynamics Inc.; 1990.

3. Alexander R. McN. The mechanics of jumping by a dog *(Canis familiaris)*. J. Zool.. 1974; 173: 549-573.

4. Alexander R. McN. Animal mechanics. Oxford: Blackwell Scientific; 1983.

5. Alexander R. McN. Gibbons swing stress away. Nature. 1989; 342: 229.

6. Alexander R. McN.; Jayes A. S.; Maloiy G. M. O.; Wathuta E. M. Allometry of the limb bones of mammals from shrews (Sorex) to elephant (Loxodonta). J. Zool. Lond.. 1979; 189: 305-314.

7. Alexander R. McN.; Jayes A. S. A dynamic similarity hypothesis for gaits of quadrupedal mammals. J. Zool., Lond.. 1983; 201: 135-152.

8. Alexander R. McN.; Vernon A. The mechanics of hopping by kangeroos *(Macropodidae)*. J. Zool.. 1975; 177: 265-303.

9. Apkarian J.; Naumann S.; Cairns B. A three dimensional kinematic and dynamic model of the lower limb. J. Biomechanics. 1989; 22: 143-155.

10. Armstrong W. W.; Green M. W. The dynamics of articulated rigid bodies for purposes of animation. The Visual Computer. 1985; 1: 231-240.

11. Bearder S. K.; Doyle G. A. Ecology of bushbabies, *Galago senegalensis* and *Galago crassicaudatus*, with some notes about their behaviour in the field. Martin R. D.; Doyle G. A.; Walker A. C., Editors. Prosimian biology. London: Duckworth; 1974: 109-130.

12. Bearder S. K.; Martin R. D. Acacia gum and its use by bushbabies, *Galago senegalensis* (Primates: *Lorisidae)*. Int. J. Primatol.. 1980; 1: 103-128.

13. Bennet-Clark H. C. Scale effects in jumping animals. Pedley T. J., Editor. Scale effects in animal locomotion. London: Academic Press; 1977.

14. Biewener A. A.; Alexander R. McN.; Heglund N. C. Elastic energy storage in the hopping of kangaroo rats *(Dipodomys spectablis)*. J. Zool., Lond.. 1981; 195: 369-383.

15. Bresler B.; Frankel J. P. The forces and moments in the leg during level walking. Trans. ASME. 1950; 72: 27-36.

16. Calow L.; Alexander R. McN. A mechanical analysis of a hind leg of a frog *(Rana temporaria)*. J. Zool.. 1973; 171: 293-321.

17. Cartmill M. Climbing. Hildebrand M.; Bramble D. M.; Liem K. F.; Wake D B., Editors. Functional vertebrate morphology. Cambridge, MA: Harvard University Press; 1985: 71-88.

18. Cavagna G. A.; Heglund N. C.; Taylor C. R. Mechanical work in terrestrial locomotion: two basic mechanisms for minimizing energy expenditure. Am. J. Physiol.:Regulatory, Integrative and Comparative Physiol.. 1977; 2: R243-R261.

19. Charles-Dominique P. Ecology and behaviour of nocturnal primates. London: Duckworth; 1977.

20. Cheney D. L.; Wrangham R. W. Predation. Smuts B. B.; Cheney D. L.; Seyfarth R. M; Wrangham R. W.; Struhsaker T. T., Editors. Primate societies. Chicago: University of Chicago Press; 1986: 227-239.

21. Clutton-Brock T. H.; Harvey P. H. Primate ecology and social resources. J. Zool. Lond.. 1977; 183: 1-39.

22. Crompton R. H. Foraging, habitat structure, and locomotion in two species of *Galago*. Rodman P. S.; Cant J. G. H., Editors. Adaptations

for foraging in nonhuman primates. New York: Columbia University Press; 1984: 73-111.

23. Crompton R. H. A leap in the dark: locomotor behaviour and ecology in *Galago senegalensis* and *G. crassicaudatus*. Cambridge, MA: Harvard University; 1980.

24. Crompton R. H.; Andau P. M. Locomotion and habitat utilization in free-ranging *Tarsius bancanus* : a preliminary report. Primates. 1986; 27: 337-355.

25. Currey J. D. The mechanical adaptations of bones. New Jersey: Princeton University Press; 1984.

26. DADS. 6.0. : Computer Aided Design Software Inc.; 1989.

27. Darwin C. On the origin of species by natural selection or the preservation of favoured races in the struggle for life. London: John Murray; 1859.

28. Dickinson S. The efficiency of bicycle-pedalling, as affected by speed and load. J. Physiol., Lond.. 1929; 67: 242-255.

29. Dunbar D. C. Aerial maneuvers of leaping lemurs: the physics of whole body rotations while airborne. A. J. Primatol.. 1988; 16: 291-303.

30. Emmerson S. Allometry and jumping in frogs: helping the twain to meet. Evol.. 1978; 32: 551-564.

31. Emmerson S. B. Jumping and leaping. Hildebrand M.; Bramble D. M.; Liem K. F.; Wake D. B., Editors. Functional vertebrate morphology. Cambridge, MA: Harvard University Press; 1985: 58-72.

32. Fleagle J. G. Locomotor behaviour and muscular anatomy in sympatruc Malysian leafmonkeys *(Presbytis obscura* and *Presbytis melalophos)*. Amer. J. Phys. Anthro.. 1976; 46: 297-308.

33. Gibbs K. Advanced physics. Cambridge: Cambridge University Press; 1990.

34. Gill F. B.; Wolf L. L. Economics of feeding territoriality in the golden-winged sunbird. Ecology. 1975; 56: 333-345.

35. Grafen A. How not to measure inclusive fitness. Nature. 1982; 298: 425-426.

36. Günther M. M. Funktionsmorphologische Untersuchungen zum Sprungverhalten an mehrenen Halfaffenarten *(Galago moholi, Galago (Otolemur) garnettii, Lemur catta)*. Berlin: Freien Universität; 1989.

37. Günther M. M.; Ishida H.; Nakano Y. The jump as a fast mode of locomotion in arborial and terrestrial biotopes. Z. Morph. Anthrop.. 1991; 78: 341-372.

38. Hall-Craggs E. C. B. An analysis of the jump of the lesser galago *(Galago senegalensis)*. J. Zool.. 1965; 147: 20-29.

39. Hall-Craggs E. C. B. The jump of the bushbaby - a photographic analysis. Med. Biol. Ill.. 1964; 14: 170-174.

40. Hall-Craggs E. C. B. Physiological and histochemical parameters in comparative locomotor studies. Martin R. D.; Doyle G. A.; Walker A. C., Editors. Prosimian anatomy, biochemistry and evolution. London: Duckworth; 1977: 829-845.

41. Hamilton W. D. The genetical theory of social behaviour (I and II). J. theor. Biol.. 1964; 7: 1-16, 17-32.

42. Harvey P.; Martin R. D.; Clutton-Brock T. H. Life histories in comparative perspective. Smuts B. B.; Cheney D. L.; Seyfarth R. M; Wrangham R. W.; Struhsaker T. T., Editors. Primate societies. Chicago: University of Chicago Press; 1987: 181-196.

43. Hatze H. The meaning of the term 'biomechanics'. J. Biomechanics. 1974; 7: 189-190.

44. Heglund N. C. Comparative energetics and mechanics of locomotion: How do primates fit in? Jungers W. L., Editor. Size and scaling in primate biology. New York: Plenum Press; 1985: 319-335.

45. Heinrich B. Bumblebee economics. Cambridge, MA: Harvard University Press; 1979.

46. Hildebrand M. Walking and running. Hildebrand M.; Bramble D. M.; Liem K. F.; Wake D. B., Editors. Vertebrate functional morphology. Cambridge, MA: Harvard University Press; 1985: 38-57.

47. Hill A. V. The dimensions of animals and their muscular dynamics. Science Progress. 1950; 38: 209-230.

48. HP 9000 series 300 computers. HP-UX system administrator manual. Fort Collins, Colorado: Hewlett-Packard Company; 1988.

49. HP 9000 series 300/800 computers. Programming with the Xlib user interface toolbox. Fort Collins, Colorado: Hewlett-Packard Company; 1988.

50. HP 9000 series 300/800 computers. Programming with the Xrlib user interface toolbox. Fort Collins, Colorado: Hewlett-PAckard Company; 1988.

51. HP 9000 series 300/800 computers. Starbase programming with X11. Fort Collins, Colorado: Hewlett-Packard Company; 1988.

52. HP 9000 series 300/800 computers. Starbase device drivers library manual. Fort Collins, Colorado: Hewlett-Packard Company; 1988.

53. HP 9000 series 800/300 computers. Starbase graphics techniques: HP-UX concepts and tutorials. Fort Collins, Colorado: Hewlett-Packard Company; 1988.

54. Hunt K. H. Kinematic geometry of mechanisms. Oxford: Clarendon Press; 1978.

55. Jouffroy F. K.; Gasc J. P. A cineradiographical analysis of leaping in an African prosimian (Galago alleni). Jenkins F. A., Editor. Primate locomotion. New York: Academic Press; 1974: 117-142.

56. Kleppner D.; Kolenkow R. J. An introduction to mechanics. London: McGraw-Hill; 1973.

57. Lanyon L. E.; Bourn S. The influence of mechanical function on the development and remodelling of the tibia. An experimental study in sheep. J. Bone Jt. Surg.. 1979; 61A: 263-273.

58. Laurig W. Methodological and physiological aspects of electromyographic investigations. Komi P. V., Editor. Biomechanics V-A. Baltimore: Univ. Park Press; 1976: 219-230.

59. Lima S.; Valone T. J.; Caraco T. Foraging efficiency - predation risk tradeoff in the grey squirrel. Anim. Behav.. 1985; 33: 155-165.

60. Marey E. J. Animal mechanism: a treatise on terrestrial and aerial locomotion. New York: Appleton; 1874.

61. Martin R. D.; Chivers D. J.; MacClarnon A. M.; Hladik C. M. Gastrointestinal allometry in primates and other animals. Jungers W. L. Size and scaling in primate biology. London: Plenum; 1985: 61-89.

62. McFarland D. Animal behaviour: psychology, ethology and evolution. Harlow: Longman Scientific & Technical; 1985.

63. McFarland D. Decision making in animals. Nature. 1977; 269: 15-21.

64. McGhee R. B. Finite state control of quadrupedal locomotion. Proceedings of the second international symposium on external control of human extremities; Dubrovnik, Yugoslavia. ; 1966.

65. McMahon T. A. Muscles, reflexes, and locomotion. Princeton, New Jersey: Princeton University Press; 1984.

66. McMahon T. A. Size and shape in biology. Science. 1973; 179: 1201-1204.

67. Miller N. R.; Shapiro R.; McLaughlin T. M. A technique for obtaining spatial kinematic parameters of segments of biomechanical systems from cinematographic data. J. Biomechanics. 1980; 13: 535-547.

68. Mollinson T. Die Körperproportionen der Primaten. Morphol. Jahrb.. 1911; 42: 79-304.

69. Muybridge E. Animals in motion. London: Chapman & Hall; 1899.

70. NAG. The NAG FORTRAN library manual mark 13. Oxford: The Numerical Algorithms Group Ltd.; 1988.

71. NAG. The NAG graphical supplement manual mark 2. Oxford: The Numerical Algorithms Group Ltd.; 1985.

72. Napier J. R.; Napier P. H. A handbook of living primates. London: Academic Press; 1967.

73. Napier J. R.; Napier P. H. The natural history of the primates. London: British Museum (Natural History); 1985.

74. Napier J. R.; Walker A. C. Vertical clinging and leaping - a newly recognized category of primate locomotion. Folia Primatol.. 1967; 6: 204-219.

75. Nash L. T.; Harcourt C. S. Social organization of galagos in Kenyan coastal forests: II. *Galago garnettii*. Am. J. Primatol.. 1988; 10: 357-369.

76. Norton F. G. J. Advanced mathematics. London: Pan Books; 1987.

77. Oxnard C. E.; Crompton R. H.; Lieberman S. S. Animal lifestyles and anatomies: the case of the prosimian primates. Seattle: University of Washington Press; 1990.

78. Oxnard C. E.; German R. Z.; McArdle J. E. The functional morphometrics of the hip and thigh in leaping prosimians. Amer. J. Phys. Anthro.. 1981; 54: 481-498.

Appendix

79. Oxnard C. E.; German R. Z.; Jouffroy F. K.; Lessertisseur J. The morphometrics of limb proportions in leaping prosimians. Amer. J. Phys. Anthro.. 1981; 54: 421-430.

80. Paul P. Robot manipulators: mathematics, programming and motor control. Cambridge, MA: MIT Press; 1981.

81. Pezzack J. C.; Norman R. W.; Winter D. A. An assessment of derivative determining techniques used for motion analysis. J. Biomech.. 1977; 10: 377-382.

82. Prost J. H. A definitional system for the classification of primate locomotion. Am. J. Phys. Anthro.. 1967; 26: 149-170.

83. Radar C. M.; Gold B. Digital filtering design techniques in the frequency domain. Proc. IEEE. 1967; 55: 149-171.

84. Rasmussen D. T.; Izard M. K. Scaling of growth and life history traits relative to body size, brain size, and metabolic rate in lorises and galagos *(Lorisidae, Primates)*. Am. J. Phys. Anthrop.. 1988; 75: 357-367.

85. Riermsersa D. J.; Bogart A. J. van den; Schamhardt H. C.; Hartman W. Kinetics and kinematics of the equine hind limb: *in vivo* tendon strain and joint kinematics. Am. J. Vet. Res.. 1988; 49: 1353-1359.

86. Ripley S. The leaping of langurs, a problem in the study of locomotor adaptation. Amer. J. Phys. Anthro.. 1967; 26: 149-170.

87. Rothschild M.; Schlein Y.; Parker K.; Sternberg S. Jump of the oriental rat flea *Xenopsylla cheopsis* (Roths). Nature. 1972; 239: 45-48.

88. Saint-Exupéry A. de. Le petit prince. Paris: Gallimard; 1946.

89. Schmidt-Nielsen K. Animal physiology. Cambridge: Cambridge University Press; 1983.

90. Schultz A. H. The skeleton of the trunk and limbs of higher primates. Hum. Biol.. 1930; 2: 303-438.

91. Shapiro R. Direct linear transformation method for three-dimensional cinematography. Research Quarterly. 1978; 49: 197-205.

92. Shaw G. A. A few notes upon four species of lemurs, specimens of which were brought alive to England in 1878. Proc. Zool. Soc. London. 1879: 132-136.

93. Smith R. M. Biomechanics of the locomotion of *Galago Senegalensis*. Arizona: Arizona State University; 1987.

94. Sperry D. Fiber type composition and post-metamorphic growth of anuran hindlimb muscles. J. Morph.. 1981; 170: 321-345.

95. Strickberger M. W. Evolution. Boston: Jones and Bartlett; 1990.

96. Tattersall I. The primates of Madagascar. New York: Colombia University Press; 1982.

97. Thomas O. On the mammals obtained by Mr. John Whitehead during his recent expedition to the Philippines with field notes by the collector. Trans. Zool. Soc. London. 1896; 54(2): 387-398.

98. Tinbergen N. On aims and methods in ethology. Z. Tierpsychol.. 1963; 20: 410-433.

99. Townsend M. A.; Seireg A. The synthesis of bipedal locomotion. J. Biomech.. 1981; 14: 727-738.

100. Walker A. Locomotor adaptations in past and present prosimian primates. Jenkins F. A., Editor. Primate locomotion. New York: Academic Press; 1974: 349-381.

101. Walton M.; Anderson B. D. The aerobic cost of saltatory locomotion in the Fowler's toad *(Bufo woodhousei fowleri)*. J. exp. Biol.. 1988; 136: 273-288.

102. Weis-Fogh T. A rubber-like protein in insect cuticle. J. Exp. Biol.. 1960; 37: 889-907.

Appendix

103. Wells J. P.; DeMenthon D. F. Measurement of body segment mass, center of gravity, and determination of moments of inertia by double pendulum in *Lemur fulvus*. Am. J. Primatol.. 1987; 12: 299-308.

104. Wells R. P. Mechanical energy costs of human movement: an approach to evaluating the transfer possibilities of two-joint muscles. J. Biomech.. 1988; 21: 955-964.

105. Winter D. A. Biomechanics and motor control of human movement. New York: John Wiley & Sons; 1990.

106. Winter D. A.; Sidwall H. G.; Hobson D. A. Measurement and reduction of noise in kinematics of locomotion. J. Biomech.. 1974; 7: 157-159.

107. Wood G. A.; Marshal R. N. The accuracy of DLT extrapolation in three-dimensional film analysis. J. Biomech.. 1986; 19: 781-785.

108. Yu L. Ontogeny of children's limbs - with particular reference to inertial characteristics. Liverpool: University of Liverpool; 1991.