12.010 Homework #1 **Due Thursday September 29, 2011**

Solutions

Question (1): (10-points) Express the following numbers in base 2, 8, 10, and 16 as appropriate (subscript denotes the base of the input number). (See notes on web page and power point)

65261_{10}

15_{16}

5655_{8}

6013_{8}

Answer: Since each of the columns in a number represents the multiplier of the base to power of the (column –1), we can solve this problem by direct calculation. The easiest problems of this type are conversion to base10 since this is the system most of us are familiar with. The conversion from base10 to some other is the most difficult, however in a question such as this (conversion to multiple systems), having done one the rest should be easy. One technique, I will show below for the second question—the conversion of 65261_{10}. In base 16, we know that the number will look like

$$e\acute{}16^4+d\acute{}16^3+c\acute{}16^2+b\acute{}16^1+a\acute{}16^0= e\acute{}65536+d\acute{}4096+c\acute{}256+b\acute{}16+a.$$

Since our value is less than $65536(=16^4)$, we know that e and above must zero. To compute d, we divide the number by 4096 and take the integer part (=d), then subtract d´4096, from the original number and obtain a remainder (d=15 which is Hex F, the remainder R=3821). We repeat the same process on the remainder, this time dividing 256 to obtain c. This results in c=14 which hexadecimal digit E. The remainder is divided by 16 to obtain b, and the final remainder a. To compute the other terms, we convert the hexadecimal to binary, and convert the binary. To do this, we use the fact that a 4-bit binary number can represent each digit in the hex number. The table below shows the concept

Hexadecimal	F				E				E				D			
Binary	1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
Octal	1	7			7				3				5			
Notice how the binary digits can be grouped and mapped to the digits in the hexadecimal and octal numbers.																

Final answers to all the values are given below. Values in bold are the original numbers given.

Base 2	Base 8	Base 10	Base 16
1111111011101101	177355	**65621**	FEED
10101	25	21	**15**
101110101101	**5655**	2989	BAD
110000001011	**6013**	3083	C0B

Question (2): (10-points) How long will it take on a 56K modem to transfer a 56 Mbytes file? How long on 45 Mbps T1 Ethernet line? Calculation should be accurate to 3-significant digits.

Answer:

(a) There are two answers to this problem. Strictly kbs is 1024 bits-per-second and k bs is 1000 bits-per-seconds. However in common use today is kbs to mean either 1024 or 1000 bits-per-second. Hence a 56K modem could transfer data at 56*1024 bits-per-sec, and a 56 Mbyte file contains 56*1024*1024*8 bits, the transfer will take 8192 seconds, assuming that the full transfer rate is achieved, or the rate could be 56*1000bps, in which case the time will be 8388.6 seconds. In practice these modems rarely achieve rates higher than 45 kbps. A standard that seems to be adhered to is b means bit and B means byte.

(b) For a 45 Mbps T1 Ethernet line the transfer rate is 45*1024*1024 bits-per-second, and again assuming full transfer rate, it would take 9.96 seconds or if 1000 bps is assumed 10.44 seconds.

Question (3): (10-points) In a computer with 1 Gbytes of memory, what is the maximum size matrix that can be stored with 8-bytes per number in (a) full storage i.e., NxN, (b) lower triangular form. What are the values if the numbers are stored in 4-byte numbers (assume all of the memory can be used for storage)? **The numbers here should be exact, not approximations.**

Answer:

(a) In 1Gbytes, 1024*1024*1024 bytes of data can be saved. An NxN matrix stored in 8-byte floating point will require N*N*8 bytes of storage. Therefore, the size of matrix that can be stored is $N = \sqrt{1024*1024*1024/8} = 11585$ i.e., a 11585x11585 matrix (with 5503 real*8 bytes remaining).

(b) If lower triangle form is used, then the storage of the matrix requires Nx(N+1)/2 elements, each of 8 bytes, and therefore the maximum sized matrix is $N = \frac{-1 \pm \sqrt{1+8M}}{2}$ where M is the total memory in real*8 words (use only positive root). The exact positive solution to this equation is N=16383.5, but we must use the integer value for this. Therefore N=16383, leaving us with 65536 bytes to use for any program that manipulates this matrix.

If 4-byte floating point is used, then we can store twice the number of values but since the matrix size goes as N^2 the size of matrix will only go up by √2 for the full storage case and ~√2 for the lower diagonal case. Therefore for full storage, N=16384, and for lower triangular form, N=23169.

Question (4): (20-points)) In class we gave the precision and range for IEEE 4-byte floating point numbers. What is the precision and range for IEEE 8-byte floating-point numbers? (IEEE 8-byte floating point uses 11 bits for the exponent and 53 bits for the mantissa (don't forget about the sign bits). (see Notes on web page and power point)

Answer: Since 10 bits are allocated to the exponent; the largest number that can be represented is 2^{10}-1=1023. Therefore the total range of the exponent is $2^{1023}=9\acute{}10^{307}$ or approximately 10 to the powers of –308 to 308. In the mantissa we have 53 bits, leaving 52 bits to represent the value of the mantissa (one sign bit). The maximum number that can be represented in $2^{52}=5\acute{}10^{15}$, and hence we have approximately 15 significant digits.

Question (5): (10-points) Develop an algorithm to sum the squares of integers between two values n and m i.e. $\sum_{i=n}^{m} i^2$**.** The solution is not computer code but rather the method to be used written out in English sentences.

Answer: There are several ways of solving this problem in terms of algorithm design. We examined this type of problem in lecture 1 and there are two practical solutions to the problem.
(1) The easiest type algorithm is to directly compute the sum and so this would look like:
- Get the input values of n and m
- Initialize a variable that will contain the sum to zero
- Loop values of i between n and m
  Add to the sum the value if i*i
- End loop
- Output the sum.

(2) A faster method that requires more thought is to derive an expression for the sum that can be directly computed. This initially takes more of the programmer's time but will execute faster later. There are formulas for the sums of powers of integer and one example is
CRC Standard Mathematical Tables and Formulae, 30th edition, which is available on-line at
http://www.mathnetbase.com/ejournals/books/book_summary/summary.asp?id=614
http://www.mathnetbase.com/books/614/cr2479_01.pdf section 1.2.12, Page 21, Sums of powers of integers, gives the expression we need.
The sum of i2 from 1 to n is n(n+1)(2n+1)/6 and the formula we need is [m(m+1)(2m+1)-n(n+1)(2n+1)]/6. This algorithm will look like.
- Get the input values of n and m
- Set sum to [m(m+1)(2m+1)-n(n+1)(2n+1)]/6
- Output the sum.

Issues that can be encountered:

(a) We have seen that integers can only reach finite value: 4-byte integers are the most common and have a maximum value of 2147483647. When m=1860 and n=1, the final result is within the range limit, but the intermediate calculation of n(n+1)(2n+1) would exceed the range of integers. There are two types of solutions to this: (a) divide one the terms by 6 first and then multiple by the others (care is needed here in that integer calculations will round down unless the result is integer or (b) do the calculation in floating point can convert the final answer to the nearest integer (because there will be some rounding. Integer overflows are often a problem when factorials are used and these calculations are often better down in floating point.
(b) There can problems with the second method when m and n are close and large. If m=n+d and d is small compared to n, then the sum is approximately n^2 d. For d=1, n and m as large as 46340 can be computed without integer overflow. In these cases, the simple algorithm (1) would be fast and easy to implement. Some analytic manipulation of the expression in (2) could yield a closed form expression using just n and d that might be less prone to integer overflow. This will take more time to develop and errors could be made in the derivation. Sometime the straightforward "brute force" algorithm is the best.

Question (6): (50-points) Design an algorithm to predict how long it will take a bicyclist to travel a specified course given the characteristics of the bicycle and rider and the amount of power the bicyclist can output as a function of time. This problem will lead to later questions where you will write a program to simulate the motion of the bicyclist. In this question you are not writing computer code: You are finding the equations you will need to use and thinking about how to implement those equations into an algorithm to solve this problem.

You will write (in English) a description of
(1) Equations of motions of the bicycle. Concentrate here on basic energy and force equations. (Final problems will be 2-D motion i.e., no corners)
(2) Converting forces and accelerations: Given the forces acting on the bicycle, how do you calculate the motion of the bicycle and the power requirements.
(3) Input and outputs for the program. What types of information will the program need for input and what might it output?
(4) List a set of methods that you will use to validate that your program outputs the correct values and how you will judge "correct".
(5) Given the nature of this problem, describe what you think will be the issues that you will need to consider carefully when programming this problem.

(Your answer should address each of the items above. Your answer should be equations and written description.

Answer:

(1) Forces and equations of motion

This is a basic force balance analysis, which can be affected by many different processes. The basic motion must satisfy **F**=*m***a** where **F** is the (vector) sum of the forces acting on the bike, *m* is the mass of the rider and bike and **a** in the (vector) acceleration of the body. Consider the forces:

(a) Gravity: Since this is bike we can take it to be constant (i.e., no great height changes). In a very general solution we would treat gravity as height dependent.

(b) Centripetal force that will be used to keep the bike on the path to be followed. This force is need even when motion is just in 2-D (ie., height and horizontal distance).

(c) Drag: Normally drag will act along a direction opposite to the velocity vector of the body. On a bike there are two types: Wind drag which will depend on velocity squared, and rolling drag which just depends on the mass of the bike and rider and an effective friction coefficient.

(d) Force added by rider. For a bicycle this can complicated. The rider applies a force to the pedals and the force is applied over a distance as the pedal moves. The force times the distance traveled is the work done by the rider. This work results in a force at the tires on the ground and so we can think of the work being done as the force at the tires on the ground times the distance the bike moves. The complications here are that there is energy lose going from the pedals to tires. This lose could be quantified or we could just use the work at the tires (knowing then that this is minimum amount of work the rider will need to do).

The next important concept is that the rate of doing work is power and hence the power used by the rider will be

$$P_r = d(F_r D)/dt = F_r V_b \quad \Rightarrow F_r = P_r / V_b$$

Hence if we know the power supplied by the rider and the velocity of the bike, we can compute the force supplied by the rider the wheels. This force acts along the direction of the road surface. Notice here that a zero velocity, the force is infinite and this means at low velocities the rider cannot supply all the power possible because the force would be too large. Therefore we need to have an upper limit on the rider force at low velocities.

We need to worry about the accuracy of the calculation. This will affect the details of the numerical techniques that we use.

How to solve the problem? The basic choice is always between analytic solutions and numerical solutions. For steady state solutions over flat topography a force balance equation can be written in which the force supplied by the rider matches the drag and rolling forces. However, our problem is more complicated and so a numerical solution is needed. Note, however, we can partially test our code with the steady state solutions.

Numerical Solution: Basic equation of motion:

$$\ddot{\mathbf{x}}(t) = \mathbf{F}(t)/m$$

$$\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}_o t + \int_o^t \ddot{\mathbf{x}}(s)ds$$

$$\mathbf{x}(t) = \mathbf{x}_o + \int_o^t \dot{\mathbf{x}}(u)du = \mathbf{x}_o + \int_o^t \left(\dot{\mathbf{x}}_o u + \int_o^u \ddot{\mathbf{x}}(s)ds \right) du$$

where **x** is the position and the dots above **x** denote derivatives, **F** is the vector sum of the forces applied to be body, and t, u and s are times.
There are two methods to solve the problem:

(a) Solve in 2-D with say x and z coordinate directions (i.e., x above is a 2-D vector) and include forces that keep the bike on the path. This approach is the only one that would work for a 3-D problem.
(b) Solve in 1-D by projecting all forces into the direction of travel and then assuming that normal forces balance and keep the bike on the path. This is basically the roller coaster solution and should give the same results as (1) if the bike does not become airborne. Solution (2) is easier but not as flexible as solution (1). It provides a means of testing solution (1).

Force calculations (equations that will be needed).
Gravity: **F** will be a vector pointing down which may depend on the height of the body (or could be constant for small displacements of the body). The formulas for gravity are

$$F = \frac{GMm}{r^2} = \frac{GMm}{(R+h)^2} \approx \frac{GMm}{R^2}(1 - 2h/R) = (9.806 - 3.0784 \times 10^{-6} h)m$$

where GM is the gravitational constant times the mass of the Earth (GM=3.98x10^{14} m^3/sec^{-2}), R is the radius of the Earth 6.3708x10^6 m, and h is the height of the bike above the surface. (The numerical values above are m/s^{-2} and h is in meters). Height dependence will only be needed for a very mountainous terrain. In our problem we can keep gravity constant at 9.8 m/s^2.

Centripetal Force: Curvature effects on the path followed by the bike require that the centripetal forces be included even when the trajectory is 2-D (i.e, along the path and up/down direction.

$$\boldsymbol{F}_c = -mV^2 \frac{d^2z}{dx^2}$$

where x and z are the horizontal and vertical coordinates and *V* is the velocity of the bike. Correctly accounting for the centripetal force will keep the bike on the specified path. If the curvature is too large in a convex sense (i.e., top of a hill) then the bike could leave the path. This case is one that needs to be considered when the solution to the problem is programmed.

Drag Force: The drag force is modeled by

$$\mathbf{F}_d = -\frac{1}{2} C_d \rho \mathbf{V}^2$$

where C_d is the drag coefficient again depending on wing area and aspect ratio, *r* is the air density, **V** is the velocity Drag acts along the wing shape.

Rolling Force:

$$\text{Rolling Drag} \quad F_r = -M_r g C_r$$

where Cr is a rolling coefficient and represents the effects of friction. It acts opposite to the direction of motion but its magnitude is constant.

The motion of the bike can be computed by vector summing all the forces, including the reaction force from the surface the bike sits on (unless the bike becomes airborne) and computing the accelerations. The accelerations are integrated to compute velocities and positions. The power supplied by the rider may be integrated in time to compute the total energy used (and thus the number of calories expended by the rider).

(2) Converting forces and motion

The forces are converted into accelerations by dividing by the mass of the bike and rider. The mass is needed when the energy used by the rider is computed. To compute the motion of the bike, the accelerations need to be integrated to yield velocities and velocities integrated to get position. The position along the track determines the slope and curvature of the path, which are needed to compute the accelerations. The integration is not simple time integration but rather the system is coupled. We need to solve a 2nd order differential equation.

Method for solving equations

Simple integration: Euler's method is simplest and least accurate method. In numerical integration, small time steps are taken from the initial conditions and at each step; the acceleration and current velocity are used to compute the velocity and value at the end of the time step. In our case, using the θ_1" and θ_2" second derivatives we can compute updated values of θ_1' and θ_2' and θ_1 and θ_2 after each time step. The so-called boundary conditions or initial conditions, give the initial values of θ_1' and θ_2' and θ_1 and θ_2 at time zero.

Mathematically Euler's method is written below where the y will be θ_1 or θ_2 and the function *f* is the expressions above and Δt is the time step. So here, the value of y and y' at time t_n is stepped to time t_{n+1}.

Solution to $y'' = f(t, y, y')$ for a step in t (time) of Δt

$$y_{n+1} = y_n + \Delta t[y'_n + \Delta t f(t_n, y_n, y'_n)/2] + O(\Delta t^2)$$

$$y'_{n+1} = y'_n + \Delta t f(t_n, y_n, y'_n)$$

The term $O(\Delta t^2)$ means the error is "of-order-of" Δt^2. This means that if the time step is halved, the error should reduce by $(1/2)^2$ or a quarter. Other methods have much better error performance.

Euler's method is very inaccurate and a better approach tries to take into account that y'' is changing over the time step Δt. The Runge-Kutta integration is one of the most common methods for improved performance. The equations below are a 4th order

Runge-Kutta algorithm given at
http://www.convertit.com/Go/Convertit/Reference/AMS55.ASP?Res=150&Page=897
equation 25.5.20 and is from Abramowitz and Stegun, Handbook of Mathematical functions. The function for the accelerations need to be called more frequently with this method but the accuracy is O(Δt^5) which means that if the time step is halved, the error should reduce by $(1/2)^5$ or 1/32. This method should run much faster than Euler's method because a much larger step size can be taken.

Solution to $y'' = f(t, y, y')$ for a step in t (time) of Δt

$$y_{n+1} = y_n + \Delta t[y'_n + \frac{1}{6}(k_1 + k_2 + k_3)] + O(\Delta t^5)$$

$$y'_{n+1} = y'_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

$$k_1 = \Delta t f(t_n, y_n, y'_n)$$

$$k_2 = \Delta t f(t_n + \Delta t/2, y_n + \Delta t y'_n/2 + \Delta t k_1/8, y'_n + k_1/2)$$

$$k_3 = \Delta t f(t_n + \Delta t/2, y_n + \Delta t y'_n/2 + \Delta t k_1/8, y'_n + k_2/2)$$

$$k_4 = \Delta t f(t_n + \Delta t, y_n + \Delta t y'_n + \Delta t k_3/2, y'_n + k_3)$$

There are equivalent Runge-Kutta formulas when first derivatives are given such as in the second set of equations above that use the momenta.

(3) Input/output for program.

The user inputs we will need are:

(a) Information about the path followed by bike. There are many ways this information could be entered such as heights along the path or simple functions. Because we need to compute the curvature, constructing the surface as the sum of a set of sine and cosine functions of different wavelengths is an appealing input method. The type of expansion is called a Fourier series. In theory any surface can be represented by a Fourier series and so the coefficients and wavelengths could be entered. Alternatively, points along the path could be entered and the Fourier coefficients computed in the program.

(b) Properties of bike and rider: Mass, Rolling and Drag coefficients. Power use profile. This input is more tricky since it is not clear how to parameterize the use of power by the rider. For example will the power from the rider decay with time? And if it does decay, how would the decay be represented? In the homework codes, we will assume constant power use by the rider.

(c) Accuracy needed for the integration. This is critical because the accuracy requirement will set the characteristics of the integration especially the step size.

Output could simply be writing results to a screen or file, or we might put an interface to graphics routines that would plot the results. The specific output depends on the user

requirement. Types of output could position and velocity as a function of time and the total energy used by the rider.

(4) Methods to validate program.

As mentioned above there are two ways to solve this problem one which is more limited in its validity conditions (i.e., curvature and speed can not be too great) but the two implementations could be used to validate each other in the domains where they are both valid.

Individual parts of the code can be checked e.g. is the drag force reasonable, does the bike accelerate downhill even with no rider power? Does it reach terminal velocity going hill and does it stop going up-hill when no new power added.

Some conditions such a flat surface and constant speed can be solved analytically to validate the solution. Care needs to be exercised here because not all cases are tested with this validation. For example, curvature effects are not tested with the simple analytical solutions.

Rider power could be cut to zero at some, time and the decay of the speed could be verified.

Knowing how much power riders can supply can be used to test the order of magnitude correctness and the calories used riding a bike.

(5) Issues in programing the solution.

There are issues common to all program of this type such as how maintain and achieve the accuracy requested by the user especially if the requested accuracy cannot be achieved. These issues determine the integration method and the step size used in the integration.

Specific to the bike some issues are:

(a) What happens in the program if hills are too steep and the bike can't get up them?

(b) What happens at the tops of hills if the bike leaves the ground? Do we want the program to handle this case? (The 1-D solution method does not allow this to happen).

(c) What methods should be used to compute energy used? Do we compute as the equations are solved or save the results of the integration and then compute the energy?

Example layout of a program for this problem:

Here we write out in English the major parts of the program (basically what the main program and each major module will do).

Main program:

Start: Tell user what the program does and

Get_input — Get input from user either by reading from keyboard or file, or by decoding command line. Input will include surface description, coefficients for forces, mass, and power use profile.
Initialize the integrator and time, and pre-calculate any quantities that are constant (for example value of gravity)
Loop over the time needed starting at zero and going to the time requested by the user in time steps requested by the user.

Determine the number of steps needed by the integrator. Basically do this by using the last step size, and see if the accuracy is adequate. If it is not then we half the step size and take twice as many steps. If the accuracy if far better than we need, then double step size and see if still OK. By using multiples of two, we can always fit an integer number of steps into time step requested by the user.

Integrate to the next time step

Output the results

End loop over time steps
Output any final values such as total energy used and travel time.
End Main

Get_input routine
This will basically collect the user input that we need. These include:

(a) Information about the path followed by bike. In theory any surface can be represented by Fourier series and so these coefficients and wavelengths could be entered.
(b) Properties of bike and rider: Mass, Rolling and Drag coefficients. Power use profile. This is more tricky since it is not clear how to parameterize the use. In homework codes, we will assume constant power use (except for graduate homework).
(c) Accuracy needed for the integration

Step size calculation:
The issue here is getting the step size that satisfies the accuracy requirements. To get the error estimate to decide on step size we could try two step sizes and check the difference in the results. If the difference is small then the current step size is OK.

Integrate to the next time step: Given we have the step sizes this is now relatively easy. Note the routines to do this calculation, e.g., gravity, drag, rolling forces etc will be shared with the error/step size calculation modules.

Output of results: This could simply be writing results to a screen or file, or we might put an interface to graphics routines that would plot the results. Again depends on user requirement.

12.010 Computational Methods of Scientific Programming
Fall 2011