12.010 Homework #1 **Due Thursday September 25, 2008**

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

$$e\times16^4+d\times16^3+c\times16^2+b\times16^1+a\times16^0= e\times65536+d\times4096+c\times256+b\times16+a.$$

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			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 1 Gbyte file? How long on a Gigabit-per-second 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. For file sizes, k-bytes are also 1024-bytes. A 56K modem could transfer data at 56*1024 bits-per-sec, and a 1 Gbyte file contains 1024*1024*1024*8 bits, the transfer will take 149796.6 seconds, assuming that the full transfer rate is achieved, or the rate could be 56*1000bps, in which case the time will be 153391.7 seconds. In practice these modems rarely achieve rates higher than 45 kbps. A standard, which seems to be adhered to, is b means bit and B means byte.

(b) For a 1Gbps line the transfer rate is 1024*1024*1024 bits-per-second, and again assuming full transfer rate, it would take 8 seconds or if 1000 bps is assumed 8.59 seconds.

Question (3): (10-points) In a computer with 1 Gbyte 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 number (assume all of the memory can be used for storage). The numbers here should be exact, not approximations.

Answers:

(a) In 1 Gbyte, 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 $N = \sqrt{1024 * 1024 * 1024 / 8} = 11585(.237)$ 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 $N = \dfrac{-1 \pm \sqrt{1+8M}}{2}$ 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 (exactly fits), 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? (For 8-byte floating point IEEE 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 $2^{1023}=9\times10^{307}$ 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 $2^{52}=5\times10^{15}$, and hence we have approximately 15 significant digits.

Question (5): (50-points) Design an algorithm to predict the evolution of galactic systems. In these types of simulations, the galaxy is represented by a set of particulars that interact with each other. The simulation then determines how the systems evolve with time based on assumptions about the interactions. This problem will lead to later questions where you will write a program to simulate the evolution of the galaxy. 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) Approximate equations of motions that might be used in the galaxy evolution.
(2) Converting forces and accelerations: Given the forces acting on the components of the galaxy, how do you calculate their motions. You should write out how you will integrate the equations of motions (i.e., converting acceleration to velocity and velocity to position).
(3) Input and outputs for the program. What types of information will the programs will need input and output? In the input here should be considered carefully given that the problem could include large numbers of particles.
(4) How might the problem be scaled up to large numbers of particles.

Your answer should address each of the items above. Your answer should be equations and written description. The more completely you think about how to solve this problem, the easier the coding of the solution in the next homework will be.

Answer: This solution starts with a summary of my notes on the issues associated with this problem.

(1) This is a basic force balance analysis, which is affected by the forces from many bodies acting on each other. The basic motion of each particle must satisfy **F=*m*a** where **F** is the (vector) sum of the forces acting on the particle from all the other particles, *m* is the mass of the particle and **a** in the (vector) acceleration of the particle. These forces and accelerations must be computed for each particle based on the positions and velocities of the other particle. Consider the forces:

(a) Gravity: This is the main force affecting the particles. For each one, it is the vector some of all the forces from the other particles. As particles get close together it can be very large.
(b) Particle near-range interactions. There could be a large range of forces of this type. Examples would include the interactions of particles of finite size as the pass close to each other (e.g., atmospheric drag type effects) or through non-gravitational forces such as magnetic attractions or Lorenz forces between magnetic fields and particles carrying charge. These types of forces are often

treated through "parametric models" i.e., the full force equations are not solved but rather replaced by an approximate (and easier to compute) model that is valid under certain conditions. For body interactions, especially diffuse type bodies, a parameterization here might include the overlapping areas of the bodies and their relative velocities similar to the way drag forces are parameterized. There could also be mass loss and new particle generation from these interactions. These effects would have an impact on the "book-keeping" of the particles if new ones could come into existence during the run of the program.

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

(2) How to solve the problem? The basic choice is always between analytic solutions and numerical solutions. If there are just two bodies then there is an analytic solution and our codes could be tested against that solution. Even with multiple bodies with 1 large body and many smaller ones that are small enough not to affect each other, the two-body analytic solution can used to verify code. However for the more complex complete problem a numerical solution will be needed.

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. These integrations need to be done for each body.

Force calculations

Gravity: **F** will be a vector pointing between the attracting bodies. At each particle, the total force will the sum of the forces from the other particales. The formulas for gravity are

$$\mathbf{F}_{12} = \frac{GM_1M_2}{r_{12}^2}\mathbf{r_{12}}$$

where GM_1M_2 is the gravitational constant times the masses of the two bodies ($GM=3.98\text{x}10^{14}$ m^3/sec^{-2}), r_{12} is vector between the bodies and in this case the force is due body 2 and applied to body 1. There is the opposite signed force at site 2. When the acceleration at particle 1 is computed, M_1 will drop from the equation.

The motions of the particles can be computed by vector summing all the forces and computing the accelerations. The accelerations are integrated to compute velocities and positions.

Numerical integration: We will return to this in the next homework. For the moment you should consults books such as Numerical Recipes and Abramowitch and Stegum (page 883) that contain methods for numerical integration. Methods such as Simpson's or Bode's rule is probably good for our application and Markoff's formula can be used to get error estimates in the integration. The integration could be done with Simpson's rule:

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{f_1}{3} + \frac{4f_2}{3} + \frac{f_3}{3}\right] + O\left(-\frac{h^5}{90}f^{(4)}\right)$$

where h is the step size $(x_3-x_1)/2$, and $f^{(4)}=d^4f/dx^4$. Although simple, the error in this scheme still goes as the step size to the 5th power, so halving the step size should improve the error in the integration by a factor of 32. To compute the derivatives, a simple expression for the second derivative (applying twice yields the fourth derivative):

$$\frac{d^2f}{dx^2} \approx \frac{f_3 - 2f_2 + f_1}{h^2}$$

where h is the separation of the points used to compute the derivative. (We could also use analytic expressions for the derivatives but again as the forces become more complex this approach will be more difficult). By changing the step size during integration we can change the accuracy of the integration. Of course, at some point if the step size is too small there will be accumulate excessive rounding. If this is a problem a more complication integration formula may be needed which have an error that depends on the integration to a higher power.

M. Abramowitch, I.A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, Wiley, New York, 1970.
W. Press, S.I A. Teukolsky, W. T. Vetterling and B. P. Flannery *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press. Cambridge, 2007.

<u>**(3)** English version of program including input and output</u>
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 the length of time the simulation will cover and the initial positions, velocities and mass of each particle. If interaction forces are to be included, then the physical size of the particles may also be needed. For small numbers of particles, explicit lists could be used but for large numbers of particles, maybe a central location, particle density and area to be covered could be specified and the program would compute the positions and velocities consistent with these values. Some random distribution could also be considered.
Initialize the integrator and time, and pre-calculate any quantities that are constant (for example GM for each body. Although in the interaction case discussed above the mass of a body might change during the run.)
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
End Main
```

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

(a) Duration on the simultion
(b) Positions, velocities, mass and possibly size of the particles for each particle.
(c) Accuracy needed for the integration. Generally these can be specified as absolute values (i.e. position errors of no more then X meters) or in relative terms (i.e. the fractional error in the position should not exceed more than some fraction of the separation of the particles (e..g., error no more than 1 part-per-million of the average separation or minimum separation.)

Step size calculation:
The step size calculation needed to make sure that the numerical integration will be accurate enough to meet the user requirements. It is far better
To get the error estimate to decide on step size we could use the numerical formulas above (in terms of derivatives of the function), or 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 force calculations will be shared with the error routine.

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. Also with many particles the output of all results could be cumbersome and so some thought should be given to this aspect. We could also consider computing total energy to see if it is conserved which is would the simple gravity case.

This problem is going to be continued in other homework and so more of the details will be given in the home works. At that time we will consider the detailed variable names to use, and the exact nature of each module in the program.

(4) Large numbers of particles could pose a serious problem for this program. Initial versions can be tested for speed characteristics as the number of particles in increased. In suspect for large numbers, the speed will decrease as the square of the number of particles.
Maintaining integration accuracy with large numbers of particles could also be problematic. The easiest integration would use the same step size for all particles but this could be inefficient when only a small number of particles are close together and need a

small step (due to rapid acceleration changes). So developing an integration approach that would use different step sizes for different particles could be useful.
It may also be necessary to simplify the equations of motion to speed up calculation. One approach here when clusters are separated would be to compute the forces at distant clusters using the center of mass and total mass of the clusters rather than the individual particles.
Finally, thinking about how to split the calculations into separate quasi-independent steps that would allow machines running in parallel to do the calculations could speed up the runs considerably. Chris Hill will discuss these topics later in semester.