12.010 Computational Methods of Scientific Programming
Fall 2008

12.010 Homework #2 Solution **Due Tuesday, October 14, 2008**

Question (1): *(25-points) (a) Write, compile and run a fortran program that generates two tables of the Gamma function. The Gamma function satisfies the following equation*

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t}\, dt$$

where z is the argument of the gamma function and can be complex. (See (See http://mathworld.wolfram.com/GammaFunction.html for information on the Gamma function). Gamma functions are rarely computed by directly integrating the equation above. Generally they are evaluated by series expansions.
For one table, table will be generated for z between 1 and 10 in increment of 1 when z is an integer. This table should give values of Γ(z), Γ(z+1/3), Γ(z+1/2) and Γ(z+2/3).
For the second table, Γ(z) should be generated for z between -1 and +1 in increments of 0.1. Results should be tabulated with at least 6-significant digits.

The submission should include
(a) A discussion of the algorithms used in the program with rationales for the choices
(b) The tables generated by the program and
(c) The fortran source code.

Solution:
The Gamma function can be computed in a variety of ways and the reason for the two types of tables (one using integer arguments with specific rational offsets and the other using non-integer values is that the methods of computing Gamma functions with these two types of arguments are very different.
There are a number of sources of information on computing Gamma functions and the one I like to use (for this and may other numerical functions and applications) is:
M. Abramowitz, I.A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, Wiley, New York, 1970. This book is available online at http://www.convertit.com/Go/Convertit/Reference/AMS55.ASP and the specific information on Gamma functions is at:
http://www.convertit.com/Go/Convertit/Reference/AMS55.ASP?Res=150&Page=255
Specifically for integer arguments we use:
$\Gamma(n+1) = n!$

$$\Gamma(n+1/3) = \frac{1.4.7.10....(3n-2)}{3^n}\Gamma(1/3) \quad \Gamma(1/3) = 2.6789385347077479 \quad (6.1.11)$$

$$\Gamma(n+1/2) = \frac{1.3.5.7....(2n-1)}{2^n}\Gamma(1/2) \quad \Gamma(1/2) = 1.7724538509055161 \quad (6.1.12)$$

$$\Gamma(n+2/3) = \frac{2.5.8.11...(3n-1)}{3^n}\Gamma(2/3) \quad \Gamma(2/3) = 1.3541179394264005 \quad (6.1.13)$$

These equations may be easily coded and maintaining accuracy in the calculation is easy. The only caution is that n! grow rapidly and 13! will overflow integer*4 maximize size and so factorials are normally computed as real*8 values.

The calculation of Gamma for arbitrary arguments is more difficult because of the slow convergence of the standard formulas for this calculation. In the home work solution we implement three different methods for this calculation: Euler's Formula, Euler's infinite product and Series expansion originally published in 1933.

Euler's formula 6.1.2

$$\Gamma(z) = \lim_{n\to\infty} \frac{n!n^z}{z(z+1)(z+2)....(z+n)} \qquad (z \neq 0,-1,-2,..)$$

Euler's Infinite product 6.1.3

$$1/\Gamma(z) = ze^{\gamma z}\prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)e^{-z/n} \qquad (|z|<\infty)$$

Series expansion 6.1.34 (page 256)

$$1/\Gamma(z) = \sum_{k=1}^{\infty} c_k z^k$$

The c_k coefficients are in the text (and ocr entries at the bottom of the web page)

In the code the equation 6.1.2 was modified so that it could be computed for increasing values of n. Using this method, the change in the estimates of Gamma(z) could be monitored to see if the answer had converged with the desired accuracy. Both (6.1.2) and (6.1.3) are very slowing converging and by experimentation it was found that when the increments to the zero 10^{-6} times smaller than the tolerance set (1.e-6), the series had converged to the tolerance. The tolerance is a settable parameter (eps) in the code and this can be changed to experiment with the accuracy of the code (eg., eps = 1.d-7 makes all the entries the same in the output.

In the output we also used the extended ASCII table to output the ± symbol. This might not generate the same symbol on all systems.

The solution is implemented in HW02_01.f and the results are:

```
% gfortran HW02_01.f -o HW02_01
% HW02_01

Table of gamma functions of positive n integers
-----------------------------------------------
   n      gamma(n)     gamma(n+1/3)     gamma(n+1/2)     gamma(n+2/3)
   1            1.         0.892980         0.886227         0.902745
   2            1.         1.190639         1.329340         1.504575
   3            2.         2.778158         3.323351         4.012201
   4            6.         9.260528        11.631728        14.711405
   5           24.        40.128956        52.342778        68.653222
   6          120.       214.021098       287.885278       389.034926
   7          720.      1355.466952      1871.254306      2593.566175
   8         5040.      9940.090984     14034.407293     19884.007341
   9        40320.     82834.091537    119292.461995    172328.063626
  10       362880.    773118.187683   1133278.388949   1665837.948385

Table of gamma functions for non-integer arguments
--------------------------------------------------
Three methods are used here:
Eulers formula              GammaEul
```

```
Eulers infinite product     GammaInf
Asymptotic series expansion GammaSer
Tolerance on calculation is    0.100E-05
      z       GammaEul(z)         GammaInf(z)         GammaSer(z)
   -1.00         ±Infinity           ±Infinity           ±Infinity
   -0.90        -10.570565          -10.570542          -10.570564
   -0.80         -5.738555           -5.738547           -5.738555
   -0.70         -4.273671           -4.273666           -4.273670
   -0.60         -3.696933           -3.696930           -3.696933
   -0.50         -3.544908           -3.544905           -3.544908
   -0.40         -3.722981           -3.722979           -3.722981
   -0.30         -4.326852           -4.326849           -4.326851
   -0.20         -5.821149           -5.821147           -5.821149
   -0.10        -10.686288          -10.686285          -10.686287
    0.00         ±Infinity           ±Infinity           ±Infinity
    0.10          9.513507            9.513506            9.513508
    0.20          4.590843            4.590842            4.590844
    0.30          2.991568            2.991568            2.991569
    0.40          2.218159            2.218159            2.218160
    0.50          1.772453            1.772453            1.772454
    0.60          1.489191            1.489191            1.489192
    0.70          1.298054            1.298055            1.298055
    0.80          1.164229            1.164229            1.164230
    0.90          1.068628            1.068628            1.068629
    1.00          0.999999            0.999999            1.000000
```

Question (2): *(25-points).*

Write a program that reads a file containing text, determines

(1) The average and root-mean-square (RMS) scatter about the mean of the number of characters per word and

(2) The average and root-mean-square (RMS) scatter about the mean of the number of words per sentence. A sentence can end with a period or question mark.

The text below is contained in the file Q2_text.txt.

The basic analysis of spacecraft tracking data requires relating the position and velocity of the spacecraft to the position and velocity of the tracking system. The coordinate system used in spacecraft navigation is shown in Figure 1. The basic measurements are of r and its time derivative and sequences of these measurements, combined with knowledge of the tracking station location and equations of motions of the spacecraft, allow the position of the spacecraft denoted here by distance r, right ascension, a, and declination, d, and its velocity to be determined as a function of time. In addition to knowing the coordinates of the spacecraft in an inertial coordinate system, the coordinates of solar system bodies are also needed in this frame. Tracking data collected on spacecraft near planets can also be used to improve the ephemerides the planets through the gravitational perturbations of the spacecraft motions. Large combined analysis of tracking data and direct measurements of planets (radar and optical positions) are used to generate planetary ephemeredes.

Hints:

Remember if reads are coded as read(,'(a)') then the file Q2_text.txt can be re-directed*

into the program using:
Q2F < Q2text.txt where Q2F is the name of the program (you can call the program any name you like).

Solution

This problem is one of careful booking keeping and thinking about the how to detect end of words and ends of sentences. It is also a case where the example text did not contain all possible scenarios and so a good code will check for sentence structures that are not in the example text. Specifically the punctuation elements that are missing are : ; and ? Only the last of these will have an impact on determining the end of a sentence. The other element missing was a hyphenated word that straddles a line back (the common place the hyphenate). The homework solution does take into account these missing elements. The ambiguous part of the sample text is what to do with the numeric 1 value in the text. The question asks for character counts, which could be interpreted as only letters or letters and numeric values. The homework solution only counts letters and not number but either solution is acceptable. It is common when implementing an algorithm to have ambiguous statements about what is needed and one complexity of implementing different possible options needs to be considered.

The homework solution reads the text from a file and the name of file is passed in the runstring. The Fortran library function getarg is used to do this. There can be differences between implementations of this function in that in some cases argument 0 is the program name (as it is in C) and in other cases argument 1 is the program name. The C-style implementation is used here.

The solution is implemented in HW02_02.f and the output is:

When only letters are counted:

```
% HW02_02 Q2_text.txt
12.010 HW02_02: In file Q2_text.txt there are:
Mean chacters per word   5.38 with RMS  3.15 in  166 words;
Mean words per sentence 27.67 with RMS 15.87 in    6 sentences
```

When the numeric 1 is counted as a character the result is:

```
% HW02_02 test.txt
12.010 HW02_02: In file test.txt there are:
Mean chacters per word   5.35 with RMS  3.16 in  167 words;
Mean words per sentence 27.83 with RMS 15.66 in    6 sentences
```

Question (3): (50-points) Write a Fortran program that will compute the motions of a set of particles undergoing mutual gravitational attraction. The program should generate the trajectories of each of the particles with an error tolerance that is proportional to the separation of the particles. The program should be able to handle large numbers of particles and thought should be given as to how to input the initial positions and velocities of particles when there are a large number of particles.

As a test of your program: Evaluate the trajectories of the 6 particles below with an error tolerance of 1.e-6. (This case is similar to the collision of two Sun-Earth-Moon systems) The integration should be run for 515-days and the positions and velocities at the end of 515 days should be included in the output.

The values below give the mass (kg), X and Y position (km) and X and Y velocities (km/s) of the 6 particles to be evaluated.

2.0e+30 kg XY 0.0e+00 0.0e+00 (km) VXY 0.000e+00 0.000e+00 (km/sec)
8.8e+28 kg XY 1.5e+08 0.0e+00 (km) VXY 0.000e+00 2.857e+01 (km/sec)
7.3e+22 kg XY 1.5e+08 1.0e+07 (km) VXY -2.424e+01 2.857e+01 (km/sec)
2.0e+30 kg XY -1.0e+09 0.0e+00 (km) VXY 1.000e+01 0.000e+00 (km/sec)
8.8e+28 kg XY -8.5e+08 0.0e+00 (km) VXY 1.000e+01 2.857e+01 (km/sec)
7.3e+22 kg XY -8.5e+08 1.0e+07 (km) VXY -1.424e+01 2.857e+01 (km/sec)

Your answer to this question should include:

(a) The algorithms used and the design of your program
(b) The Fortran program source code (I will compile and run your programs). If you program does not run or takes more than few minutes to run, let me know so that I will treat it with caution.
(c) The results from the test case above with positions and velocities at the end of 515 days. You could also explore the effects of changing the accuracy tolerance on the results.

Solution

This problem is a N-body orbital problem where the error tolerance on the integration is specified. Error tolerances of this nature are a fractional error (sometimes called relative error) and quantify the ratio of the error to a spatial scale in the problem. These types of definitions are often vague as to the spatial scale to be used. If we look at the initial coordinates, they are of order 10^9 km and 10^{-6} of this scale is 1000 km. In the homework implementation we use as the spatial scale the closest pair of bodies and ensure that their relative positions are known to the 10-6 tolerance. After day 490 of the integration, the bodies to come very close together and as a result meeting the 10-6 accuracy requirement becomes very difficult. We also specify a minimum step size (about 1-second) and when the bodies are close, step size smaller than this are needed to maintain the accuracy. The way we evaluate accuracy is to integrate each time step twice: Once with the nominal step size and the other in two steps with half the step size. The difference is used to test whether the step size should be halved or doubled. If the tolerance is not meet, the step halved and the procedure repeated until we meet the tolerance or the minimum size is reached. If the tolerance is exceeded by at least a factor of 40 the step size is doubled. The reason for the factor of forty is that a 4^{th} order Runge-Kutta integration is used and so halving the step size should improve the accuracy by a factor of 32. If we did not test for a ratio larger than 32, then the algorithm was bounce back and forth between the two step sizes. The code does include a counter of the number of times the step is changed at each integration step and if this exceeds a tolerance than the iteration is stopped. This limit is reached in the current problem when the bodies are very close to each other.

The integrator used in this solution is a 4^{th} 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.

Solution to $y'' = f(x,y,y')$ for a step in x (time) of h

$$y_{n+1} = y_n + h[y'_n + \frac{1}{6}(k_1 + k_2 + k_3)] + O(h^5)$$

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

$$k_1 = hf(x_n, y_n, y'_n)$$

$$k_2 = hf(x_n + h/2, y_n + hy'_n/2 + hk_1/8, y'_n + k_1/2)$$

$$k_3 = hf(x_n + h/2, y_n + hy'_n/2 + hk_1/8, y'_n + k_2/2)$$

$$k_4 = hf(x_n + h, y_n + hy'_n + hk_3/2, y'_n + k_3)$$

To implement this integration we need to compute the acceleration (function *f*) at four locations given by the above calculations. Notice also here that this integration is valid when the acceleration depends on velocity as well as position. While the standard gravitational problem depends only on position, this implementation of the integrator allows to easily added drag type forces the equations if we wanted to. These types of forces could be useful in allowing capture of bodies in the problem.

The basic modules in the program are:
read_runstring – allows the name of the file with body definitions, the duration of the integration and the accuracy tolerance on the integrator to given in the runstring of the program.
read_nbody – reads the body definitions in the form of mass, position x and y, and velocity x and y. The implementation in this routine only interprets lines that start with a space. All other lines are interpreted as comments.
report_ICS – routine to write out the position and velocities of the bodies are specified times. A character string is also passed to annotate the type output.
int_step – this subroutine advances the integration by one time step using the integrator discussed above.
eval_step – this subroutine evaluates if the current time is adequate for the precision needed and determines if the step size should be increased or decreased.
get_accel – this subroutine compures the accelerations of all the bodies given the positions passed in its arguments list. Time and velocity are also passed but these are needed in the gravitational only model. Drag type forces could be easily added to this routine.
Animate – this is simple routine that uses VT100 escapes sequences to plot the motions of the bodies on a 85 by 45 grid. See for example:
http://pegasus.cs.csubak.edu/Tables_Charts/VT100_Escape_Codes.html
report_error – subroutine that reports IOSTAT errors during the run.

The main program calls most of the routines above and loops over time, using the dynamically set time step, until the end time is reached. Because the time step can change, the last time step may send the integrator across the desired time step and so there is a check and re-calculation of the time step at the end of the integration.

Communication in the program is through a combination of an included common block and variables passed into and out of routines (remember on Fortran, pointers as normally passed to functions and subroutines).

The code here needs to be compiled with fortran90 or gfortran because we use the array multiplication and addition features of f90. (All these operations would need to be done with do loops in standard f77). G77 will not compile the current code. F90 is available on Athena when the add sunsoft command is used (see web page on accessing Fortran on Athena).

```
% ssh -X linerva.mit.edu
athena% add sunsoft
athena% f90 HW02_03.f -o HW02_03
Run the same way as below.
```

The code is implemented in HW02_03.f with include file NBody.h The data file with the test case is NBody.dat

The output of the program as requested in the homework is:

```
% HW02_03 Nbody.dat
+ 12.010 HW 02 Q 03: Initial Conditions At time        0.00000 days
Body     Mass (kg)   PosX (km)     PosY (km)    VelX  (km/s)  VelY   (km/s)
   1        0.200000E+31       0.0000000E+00      0.0000000E+00        0.0000000E+00
0.0000000E+00
   2        0.880000E+29       0.1500000E+09      0.0000000E+00        0.0000000E+00
0.2857000E+02
   3        0.730000E+23       0.1500000E+09      0.1000000E+08       -0.2424000E+02
0.2857000E+02
   4        0.200000E+31      -0.1000000E+10      0.0000000E+00        0.1000000E+02
0.0000000E+00
   5        0.880000E+29      -0.8500000E+09      0.0000000E+00        0.1000000E+02
0.2857000E+02
   6        0.730000E+23      -0.8500000E+09      0.1000000E+08       -0.1424000E+02
0.2857000E+02
…
animation space removed.
…
+ 12.010 HW 02 Q 03: Final conditions At time      515.00000 days
Body     Mass (kg)   PosX (km)     PosY (km)    VelX  (km/s)  VelY   (km/s)
   1        2.000000E+30      -1.9270598E+08      4.6801888E+07        3.1375402E+01     -
2.0620740E+00
   2        8.800000E+28      -3.5578701E+08      5.7460426E+07       -1.3151751E+02     -
8.7795577E+01
   3        7.300000E+22      -3.4892707E+08      5.1814765E+07        7.8372633E+01
1.0992686E+02
   4        2.000000E+30      -3.4720247E+08      6.5313058E+07       -1.8176899E+01
8.0907099E+00
   5        8.800000E+28      -2.4315337E+08     -6.3023788E+07        6.8824102E+01
7.9210531E+00
   6        7.300000E+22      -2.3446172E+08     -6.4931289E+07        7.8189206E+01
3.3496439E+01
Smallest step size needed  0.000015 days
Closest approach distance was    1.59785E+04 km Bodies  2 and  4
```

Notice here that step size gets very small during the close encounters of the bodies. If we end the integration earlier before the close encounter, the step size remains much more reasonable.

```
+ 12.010 HW 02 Q 03: Final conditions At time      490.00000 days
Body     Mass (kg)   PosX (km)     PosY (km)    VelX  (km/s)  VelY   (km/s)
   1        2.000000E+30      -2.5155546E+08      5.5598413E+07       -4.5449261E+01
1.9141995E+00
   2        8.800000E+28      -3.6875847E+08     -8.9011141E+06       -1.8474636E+01
```

```
1.7902116E+01
   3        7.300000E+22      -3.6088590E+08     -2.1983262E+06       -3.4036630E+01
3.6907956E+01
   4        2.000000E+30      -3.0404683E+08      5.3414822E+07        5.4071950E+01
7.0407737E-01
   5        8.800000E+28      -3.8601263E+08     -4.9591322E+07        5.9777127E+01
-2.0268385E+01
   6        7.300000E+22      -3.8033650E+08     -4.0240197E+07        4.1114941E+01
-6.8645586E+00
Smallest step size needed  0.500000 days
Closest approach distance was    9.34087E+06 km Bodies  5 and  6
```

In this case, the smallest step size was 0.5 days.