HW 2 Computational Physics

August 30, 2005 Due September 6

0. Go back to the syllabus and read the requirements for homework solutions.

1. Garcia, Chapter 1, Excercise 23. This problem illustrates round-off error: following the instructions with most calculators gives an answer accurate to three or so digits. The problem then instructs you to do better. Here's how: take the difference between your analytic expressions for the two accelerations (so that R cancels) before using the calculator.

2. For $f(x) = e^{2x}$, write a program that computes $f'(x)$ at $x = 3$ using both the forward and centered derivative formulas [Garcia Eq. (2.7) and (2.39) without taking the limit]. Do so for step sizes h ranging from 10^{-20} to 10^{-1}. Make a log-log plot of the absolute error *vs.* step size, similar to Garcia's Fig. 1.3. Explain your results: which formula is more accurate, and why? (This is an example of algorithmic or truncation error.) For larger h the log-log plot looks linear. What is the slope of the linear portion for each algorithm? What's going on at very small h? Here is a snippet of code:

```
from Numeric import *

# Array of step sizes h.
h = 10. ** arange(-20,0)   # Gives an array of powers.

# FS estimate and its absolute error.
x = 3.
fs = (exp(2*(x+h))-exp(2*x))/h
error_fs = abs(fs - 2*exp(2*x))
error_fs = log10(error_fs)   # For log-log plotting.

# CS estimate and its absolute error.
cs = (exp(2*(x+h))-exp(2*(x-h)))/(2*h)
error_cs = abs(cs - 2*exp(2*x))
error_cs = log10(error_cs)

# Now figure out how to plot.
```

   Hand in: (i) a sheet of paper showing brief derivations of the expressions that you need to compute (the FS and CS approximations); (ii) your code (with comments); (iii) plots; and (iv) an analysis of the results.

3. I will send you an email that contains the following data with function values f_j at times t_j:

$$t_0, t_1, t_2, \ldots, t_{n-1} \qquad f_0, f_1, f_2, \ldots, f_{n-1}.$$

   Write a Python program that calculates (approximately) the time derivative at time t_2.

4. For the same data used in the previous problem, write a Python program that calculates (approximately) the integral $\int_{t_0}^{t_{n-1}} f(t)\,dt$. Snippet:

```
from Numeric import *

....

# Set up arrays t and f (initially set to zero for no reason)
t = zeros(n,Float)
f = zeros(n,Float)


# Somehow input the numbers I email into these arrays.
....

# Implement the sum which approximates the integral.
sum = 0.
for j in arange(n):
sum = sum + f[j]


sum = sum*delta
```

5. Write a Python program that contains a function that evaluates $f(x) = x^2 + 1$ for any input x. The function could start with a line like `def f(x):` (be sure to remember the colon) and end with a `return`.

   (a) In your program's main portion, call the function for values of x ranging from 1 to 5 and print the results in two columns (each row containing $x, f(x)$).

   (b) Now define an array `a=arange(1,8)` and print the result `f(a)`. Explain.