PHZ 5156, Computational Physics
Homework 2 Solution
September 13, 2005

Problem 1 (Garcia 1.23)

(a) $v = 2\pi r/T$ with $r = R\cos 40$ and $T = 24*3600$ s gives $v = 3.553076903 \times 10^2$ m/s.

(b) $a = v^2/r = 2.583870440 \times 10^{-2}$ m/s^2.

(c) a' is the same with R replaced by R+2m=6378002 m. This gives

$v' = 2.553078017 \times 10^2$ m/s

$a' = 2.583871249 \times 10^{-2}$ m/s^2

$a' - a = 8.090000000 \times 10^{-9}$ m/s^2

This is accurate at most to three places (8.09).

This is repaired by canceling some terms using algebra:

$a = v^2/r = (2\pi/T)^2 R \cos 40$

a' = same with R replaced by R+2m

hence

$a' - a = (2\pi/T)^2 (2m)\cos 40 = 8.102447279 \times 10^{-9}$ m/s^2

This is exact to the number of places shown. Can see that the first approach was only accurate to two decimal places.

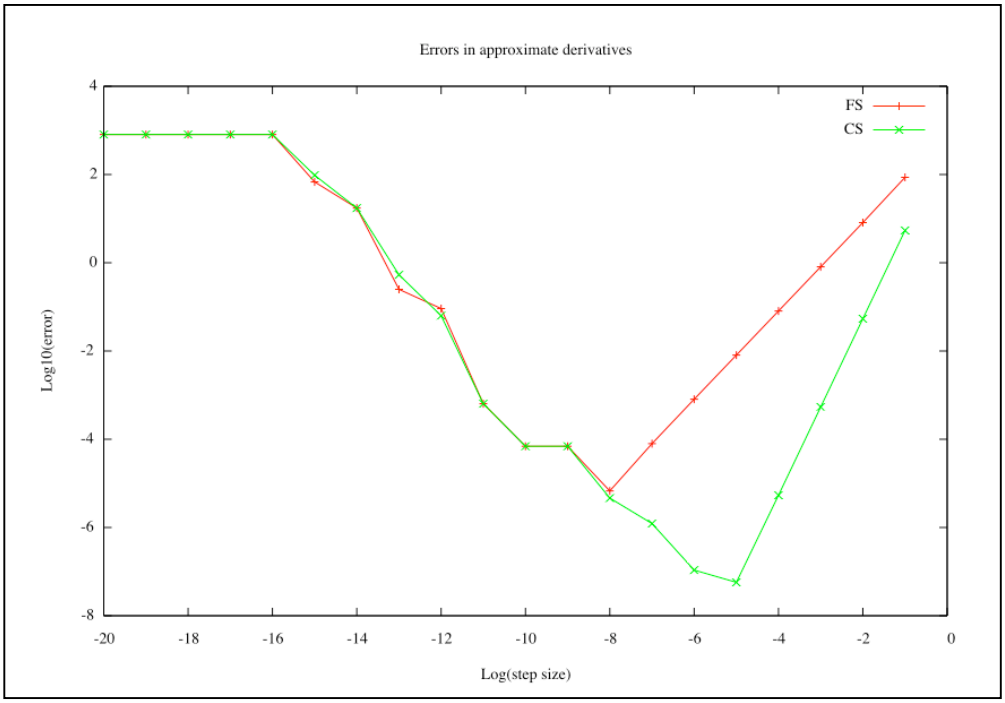
Problem 2

Python code

```
# HW2, problem 3.
# Approximate the derivative of f(x)=exp(2x) at x=3
# using the FS and CS approximations. Plot the results.
# MDJ 9/13/05

from scipy import *
import Gnuplot,Gnuplot.funcutils
g = Gnuplot.Gnuplot(debug=0)
g('set terminal aqua')

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

# Calculate for x=3
x = 3.
exact = 2.*exp(2.*x)
print "exact derivative = ",exact

# FS estimate and its absolute error.
fs=(exp(2.*(x+h))-exp(2.*x))/h
print "fs approximations=\n",fs
error_fs=abs(fs-exact)

# CS estimate and its absolute error.
cs=(exp(2.*(x+h))-exp(2.*(x-h)))/(2.*h)
print "cs approximations=\n",cs
error_cs=abs(cs-exact)

# Take logs to make log-log plot.
error_fs=log10(error_fs)
error_cs=log10(error_cs)
h=log10(h)

# Do the plotting with Gnuplot
d1 = Gnuplot.Data(h,error_fs,title='FS')
d2 = Gnuplot.Data(h,error_cs,title='CS')
g.title("Errors in approximate derivatives")
g('set data style linespoints')
g.xlabel("Log(step size)")
g.ylabel("Log10(error)")
g.plot(d1,d2)
```

Python output:

```
exact derivative =  806.857586985
fs approximations=
[  0.          0.          0.          0.          0.
     738.96444519  824.22957348  806.60811363  806.94917415  806.85822468
     806.85765624  806.85765624  806.85759372  806.85766648  806.85839396
     806.86565562  806.93827813  807.66498275  814.98022339  893.20247601]
cs approximations=
[  0.          0.          0.          0.          0.
     710.54273576  824.22957348  806.32389654  806.92075244  806.85822468
     806.85765624  806.85765624  806.85758235  806.85758576  806.85758709
     806.85758704  806.85759237  806.85812489  806.91137857  812.24740592]
```



Observations:

(a) CS is more accurate than FS

(b) There are two regions – for larger values of h the errors get smaller as h decreases; but eventually for too small h the error begins to grow.

(c) The slopes for larger h are close to 1 for FS and 2 for CS. That is because the FS error goes as h^1 and the CS error goes as h^2. The error in this larger-h region is algorithmic error (also known as step-size or truncation error).

(d) The error in the smaller-h region is round-off error. Notice in the output that for very small h the FS and CS approximations go to zero (which is nonsense). That is because for very small h when the computer calculates f(x+h) it returns a value equal to what it computes for f(x).

Problems 3 and 4

Python code:

```
# Homework 2 problems 3 and 4
# MDJ  9/13/05

from scipy import *

t=array((0,.1,.2,.3,.4,.5,.6,.7,.8,.9,1.0,1.1,1.2,1.3,1.4,1.5))
f=array((1.,0.99959219,0.9983691,0.99633173,0.99348174,
                0.98982144,0.98535384,0.98008256,0.97401192,
                0.96714685,0.95949297,0.95105652,0.94184436,
                0.93186403,0.92112365,0.909632))
n=len(t)
tau = t[1] - t[0]

print "Problem 3 - using CS approximation"
m = 2
deriv = (f[m+1]-f[m-1]) / (2.*tau)
print "deriv = ",deriv
print

print "Problem 4 - as stated"
sum = 0.
for j in arange(0,n):
        sum = sum + f[j]
sum = sum * tau
print "integral = ",sum
print

print "Problem 4 - using trapezoidal approximation"
sum = 0.5*(f[0]+f[n-1])
for j in arange(1,n-1):
        sum = sum + f[j]
sum = sum * tau
print "integral = ",sum
```

Output:

```
Problem 3 - using CS approximation
deriv =  -0.0163023

Problem 4 - as stated
integral =  1.54992049

Problem 4 - using trapezoidal approximation
integral =  1.45443889
```

The function was f(t) = cos(2t/7).

The correct answers are f'(0.2)=-0.016318 and (for the integral from t=0 to 1.5) integral=1.454; the trapezoidal approximation does better.

Problem 5:

Python code

```
# Homework 2 Problem 5
# MDJ 9/13/05

from scipy import *

def f(x):
        return  x**2 + 1

print "Problem 5  part (a)"
for n in arange(6):
        print  n,f(n)

print
print "Problem 5  part (b)"
a = arange(1,8)
print  f(a)
```

Output

```
Problem 5 part (a)
0 1
1 2
2 5
3 10
4 17
5 26

Problem 5 part (b)
[ 2  5 10 17 26 37 50]
```

Part (b) illustrates an interesting Python feature: it was not necessary to choose the type of the function's input argument. When the input was an array, the function calculated x**2+1 for an array – that is, element by element.