Homework 4 Solution
PHZ 5156, Computational Physics
September 15, 2005

Part 1

With the proposed solution y(t) the matrix product –ay on the right-hand side of the ODE produces –y. The derivative dy/dt on the left-hand side also produces -y. The two sides are equal, so the proposed y(t) is indeed a solution.

Completed code:

```
#Homework 4
#Computational Physics PHZ 5156 September 2005
from scipy import *
from scipy.integrate import odeint
import time
import Gnuplot,Gnuplot.funcutils
from LinearAlgebra import *


def rk4(f,y0,t):
        #Implements 4th order Runge Kutta, using an external f(y).
        #Usage: y=rk4(f,y0,t)
        #Input f is the name of the "RHS" function.
        #Input y0 is an array that contains the initial conditions.
        #Notice that y0 can have arbitrary length.
        #Input t is an array of times.
        #Output array y has shape len(t) x len(y0) where
        #y[n,:] contains y(tn) for all computed times.

        y = zeros( (len(t),len(y0)), Float)             # Allocate space
        y[0] = y0
        tau = t[1]-t[0]
        for n in arange(0,len(t)-1):
                tn,yn = t[n],y[n]
                h1 = tau*f(yn,tn)
                h2 = tau*f(yn+h1/2.,tn+tau/2.)
                h3 = tau*f(yn+h2/2.,tn+tau/2.)
                h4 = tau*f(yn+h3,tn+tau)
                y[n+1] = yn + h1/6.+h2/3.+h3/3.+h4/6.
        return y

def ode(y,t):
   # RHS of stiff coupled ODE
   return -dot(a,y)
```

```
def analytic(t):
   #Analytic solution
   y1 = 2.*exp(-t)
   y2 = -exp(-t)
   return y1,y2

def part2():

   # Part (2): rk4 with tiny step
   tau = 0.0001
   t = arange( 0., 6+tau, tau)
   y0 = array( (2.,-1.) )
   start = time.clock()
   y = rk4(ode,y0,t)
   finish = time.clock()
   elapsedb = finish - start
   print "\nPart (2)"
   print "rk4 with tau =",tau,"takes time = ",elapsedb
   print "Final values"
   print "y1 analytic:",y1a[-1]
   print "y1 numeric: ",y[-1,0]
   print "y2 analytic:",y2a[-1]
   print "y2 numeric: ",y[-1,1]

   g=Gnuplot.Gnuplot(debug=0)
   g.title('HW 4 Part (2): Stiff ODE solved by RK4 with tau=0.0001')
   g('set data style lines')
   g1 = Gnuplot.Data(ta,y1a,title='y1  ana')
   g2 = Gnuplot.Data(ta,y2a,title='y2  ana')
   g3 = Gnuplot.Data(t,y[:,0],title='y1  RK4')
   g4 = Gnuplot.Data(t,y[:,1],title='y2  RK4')
   g.plot(g1,g2,g3,g4)
   return

def part3():

   #Part (3): RK4 with tau = 0.01
   tau = 0.01
   t = arange( 0.,10*tau, tau )
   y0 = array( (2.,-1.) )
   y = rk4(ode,y0,t)
   y1,y2 = analytic(t)
   print "\nPart (3)"
   print "Values at first 10 times"
   print "y1 analytic:",y1[0:len(t)]
   print "y1 numeric: ",y[:,0]
   print "y2 analytic:",y2[0:len(t)]
   print "y2 numeric: ",y[:,1]
```

This file contains the code for the entire homework assignment. Separate parts are solved in separate functions; these are invoked from the main program.

```
    return

def part4():
    #Part (4): Eigenvalues
    evalues = eigenvalues(a)
    print "\nPart (4)"
    print "Eigenvalues=\n",evalues

def part5():
    #Part (5): odeint with tau = 0.01
    tau = 0.01
    t = arange( 0., 6+tau, tau)
    start = time.clock()
    y0 = array( (2.,-1.) )
    y = odeint(ode,y0,t)
    finish = time.clock()
    elapsede = finish-start
    print "\nPart (5)"
    print "odeint with tau=0.01 takes time = ",elapsede

    g = Gnuplot.Gnuplot(debug=0)
    g.title('HW 4 Part (5): Stiff ODE solved by odeint with tau=0.01')
    g('set data style lines')
    g1 = Gnuplot.Data(ta,y1a,title='y1  ana')
    g2 = Gnuplot.Data(ta,y2a,title='y2  ana')
    g3 = Gnuplot.Data(t,y[:,0],title='y1  odeint')
    g4 = Gnuplot.Data(t,y[:,1],title='y2  odeint')
    g.plot(g1,g2,g3,g4)

    print "Final values at times:",ta[-1],t[-1]
    print "y1 analytic:",y1a[-1]
    print "y1 numeric: ",y[-1,0]
    print "y2 analytic:",y2a[-1]
    print "y2 numeric: ",y[-1,1]


#Main program
a = array( ((-9998,-19998),(9999,19999)),  Float)

print "\n HW4 \n"
#Analytic solution
tau = 0.05
ta = arange( 0.0, 6+tau, tau )
y1a,y2a = analytic(ta)

part2();  part3()
part4();  part5()
```

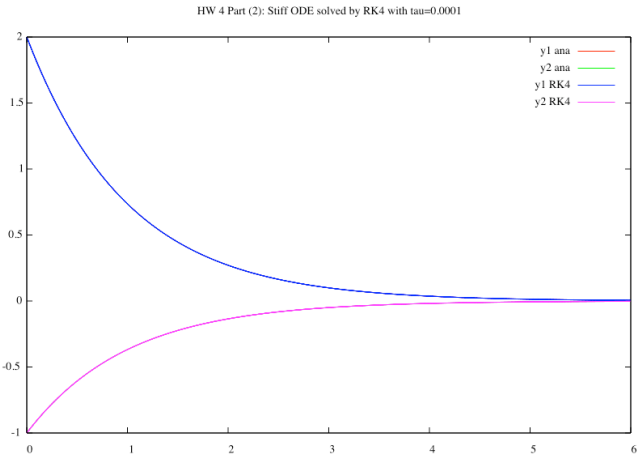
Here the four separate functions are invoked. In fact I commented out all but one at a time while I was working on this.

Part 2

Executing the RK4 code with tau=0.0001 produces the output

```
Part (2)
rk4 with tau = 0.0001 takes time =  15.84
Final values
y1 analytic: 0.00495750435333
y1 numeric:  0.00495750435333
y2 analytic: -0.00247875217667
y2 numeric:  -0.00247875217667
```

This took a very long time to run (nearly 16 seconds) because the step size was so very small. To go from 0 to 6 with a step size of 0.0001 takes 60,000 iterations of the RK4 algorithm. On the other hand the numerical results agree *exactly* with the analytical at the last time step – an amazing accuracy. The plotted results show the same agreement:



Part 3

Running RK4 with tau=0.01 produces the following output.

```
Values at first 10 times
y1 analytic: [ 2.         1.98009967  1.96039735  1.94089107  1.92157888  1.90245885
      1.88352907  1.86478764  1.84623269  1.82786237]
y1 numeric:  [ 2.00000000e+00   1.98009967e+00   1.96015930e+00  -9.51417246e+02
      -3.81810495e+09  -1.52911324e+16  -6.12394713e+22  -2.45258020e+29
      -9.82234088e+35  -3.93375028e+42]
y2 analytic: [-1.         -0.99004983 -0.98019867 -0.97044553 -0.96078944 -0.95122942
     -0.94176453 -0.93239382 -0.92311635 -0.91393119]
y2 numeric:  [ -1.00000000e+00  -9.90049834e-01  -9.79960625e-01   9.52387691e+02
       3.81810495e+09   1.52911324e+16   6.12394713e+22   2.45258020e+29
       9.82234088e+35   3.93375028e+42]
```

The divergence shows that RK4 is **unstable** for this ODE with tau=0.01.

Part 4

The eigenvalues of matrix *a* are:

```
Eigenvalues=
[  1.00000000e+00   1.00000000e+04]
```

This explains what goes wrong in part 3. The RK4 algorithm, like the Euler algorithm, is conditionally stable for this problem. That is, it is only stable for a step size tau less than some threshold value. For Euler the stability condition is tau<2/(maximum eigenvalue), that is, tau<2/10000 (using the result above). I don't know the stability condition for RK4, but can guesstimate that tau needs to be less than some number of order 1/(maximum eigenvalue), that is, tau<O(0.0001). At least it is easy to see the reason for the instability in part 3: tau there is bigger than the RK4 stability threshold.

Part 5

Running odeint with tau=0.01 produces the output

```
odeint with tau=0.01 takes time =  0.00999999999999
Final values at times: 6.0 6.0
y1 analytic: 0.00495750435333
y1 numeric:  0.00495754122813
y2 analytic: -0.00247875217667
y2 numeric:  -0.00247877059452
```

Notice that this is stable even for the step size tau=0.01 for which RK4 was unstable. The answers are accurate to about five decimal places. This is less accurate than RK4 with the extremely small step size tau=0.0001, but is quite satisfactory accuracy. And, most important, odeint gave this satisfactory result with a larger step size that enabled it to run much faster – an elapsed time of around 0.01 s. The plotted results also show this good agreement: