It is good practice to use high-quality existing routines rather than writing your own. Here you will compare your `rk4` function with scipy’s ordinary differential equation solver `odeint`, with particular attention to the step size `tau` and the run time.

1. Consider the stiff set of ordinary differential equations:

\[
\frac{dy}{dt} = -ay, \quad \text{where } y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}, \quad a = \begin{pmatrix} -9998 & -19998 \\ 9999 & 19999 \end{pmatrix},
\]

with initial conditions \(y_1(0) = 2, \ y_2(0) = -1\). Show that this is solved exactly by

\[
y(t) = \begin{pmatrix} 2e^{-t} \\ -e^{-t} \end{pmatrix}
\]

by plugging this into the differential equation.

2. Numerically solve for \(y(t)\) for \(t = 0\) to \(t = 6\) using your Runge-Kutta function `rk4` from the last homework. Use a step size \(\tau = 0.0001\). Print the time taken by the call to `rk4` and print the final value of \(y\) at \(t = 6\). Plot the analytic and numerical \(y_1(t)\) and \(y_2(t)\) vs. \(t\) together on one graph. Snippet:

```python
def stiff(yn,tn):
    # RHS of the first-order coupled ODE's.
    ...
    return rhs
def rk4(f,y0,t):
    ...
    # Main code
    y0=array([2.,-1.])
tfinal, tau = 6., 0.0001
t = arange(0,tfinal+tau,tau)
start = time.clock()
y = rk4(stiff,y0,t)
finish = time.clock()
elapsed = finish - start
```

3. Repeat using a step size \(\tau = 0.01\). Start at \(t = 0\) and calculate for ten or twenty time steps, printing the calculated values, until your numerical answer goes bad.

4. Find the eigenvalues of \(a\) and explain why (2) works and (3) does not.

5. Solve for times \(0 \leq t \leq 6\) using `odeint` and a step size \(\tau = 0.01\). To learn how to use `odeint` try

```python
from scipy.integrate import odeint
help(odeint)
```

In your code print the time for the call to `odeint` and plot your solution and the analytic solution together as in (2). Do you see the benefit of using an algorithm such as `odeint` designed to handle stiff problems? Discuss.