University of Colorado
Department of Computer Science

Numerical Computation CSCI 3656

Spring 2016

Problem Set 12 Solutions

1. *[15 pts]* Use your Simpson's 1/3 rule code from PS11 to compute $\int_{1.0}^{1.8}[e^x + 0.5]dx$ with $h = 0.1$ and $h = 0.05$. Now use one layer of Romberg integration on those results to get a better answer. Finally, comment on how close each of the three numerical answers involved in the calculations above — i.e., with $h = 0.05$, with $h = 0.1$, and with Romberg — is to the correct value that you figured out using calculus in PS11.

   Simpson's 1/3 rule gives 3.73136575 and 3.73136748 for $h = 0.05$ and $h = 0.10$ (respectively). The equation for one layer of Romberg integration is:

$$MA + \frac{1}{2^n - 1}(MA - LA)$$

   where MA and LA are the more and less accurate approximations (respectively), and $n = 4$ (from the error term of Simpson's 1/3 rule). Using the formula, the approximation for the integral is 3.73136564. The actual value for the integral is 3.73136564. The absolute errors for $h = 0.10$, $h = 0.05$, and Romberg are: 1.8486e-6, 1.1564e-7, and 1.1002e-10 (respectively).

2. *[10 pts]* Now compute $\int_{1.0}^{1.8}[e^x+0.5]dx$ using three-term Gaussian quadrature. Comment on the accuracy of the answer and the computational complexity of this approach, compared to the methods that you have used to solve this same problem on PS11 and PS12.

   From Table 5.1 on p276, the "magic points" for three-term Gaussian quadrature are at $-\sqrt{3/5}$, 0, and $\sqrt{3/5}$. The weights are 5/9, 8/9, and 5/9. That means that

$$\int_{1.0}^{1.8}[e^x + 0.5]dx = 5/9(e^{-\sqrt{3/5}} + 0.5) + 8/9(e^0 + 0.5) + 5/9(e^{\sqrt{3/5}} + 0.5)$$
$$= 3.3503$$

   This accuracy was obtained with only three evaluations of the function $[e^x + 0.5]$.

$$\int_{1.0}^{1.8}[e^x + 0.5]dx = \int_{-1}^{1}[0.4 * (e^{0.4t+1.4} + 0.5)]dt$$

   Let $f(t) = 0.4 * (e^{0.4t+1.4} + 0.5)$. Then,

$$\int_{1.0}^{1.8}[e^x + 0.5]dx = 5/9 * f(-\sqrt{3/5}) + 8/9 * f(0) + 5/9 * f(\sqrt{3/5})$$
$$= 3.73136521$$

   See section 5.5 (p. 273) of Sauer. This accuracy was obtained with only three (modified) function evaluations. For the problems in PS11 and PS12, we can compare the number of function evaluations and accuracy in the following table:

Method	Function Evaluations	Absolute Error
Trap. (h = 0.1)	9	2.776e-3
Trap. (h = 0.05)	17	6.9401e-4
Simp. (h = 0.1)	9	1.8486e-6
Simp. (h = 0.05)	17	1.1564e-7
Romberg	26	$1.1002e{-}10$
GQ (3-term)	3	4.23913-7

The Newton-Cotes methods require $(b-a)/h+1$ function evaluations, in the interval $[a,b]$ with step size h.

3. *[6 pts]* Consider the following system of ordinary differential equations:

$$\begin{aligned} \dot{x}(t) &= 16(y-x) \\ \dot{y}(t) &= 45x - y - xz \\ \dot{z}(t) &= xy - 4z \end{aligned}$$

Is this system of equations—a famous example from chaos theory called the Lorenz equations—linear? Explain your reasoning: why or why not? Of what *order* is this ODE system?

This set of equations is nonlinear because of the xz and xy terms. (Either of those terms, alone, would be enough to make them nonlinear, by the way.) Products, powers, and transcendental functions of/in the state variables are nonlinear. The order of the system is three because there are three coupled first-order ODEs.

4. *[24 pts]* In your favorite programming language, implement the forward Euler method for solving that system of differential equations numerically.

Your program should take the following inputs:

- an initial condition $[x(t=0), y(t=0), z(t=0)]^T$
- a time step size h
- the number of time steps N over which to solve the ODE system

Here's some Matlab code that does that:

```
function [trajectory] = Lorenz_FE(current_state,h,N)
trajectory=[];
for steps=1:N
   x=current_state(1,1);
   y=current_state(2,1);
   z=current_state(3,1);
   xdot=16*(y-x);
   ydot=45*x-y-x*z;
   zdot=x*y-4*z;
```

```
    slope_vector=[xdot,ydot,zdot]';
    new_state = current_state + h * slope_vector;
    trajectory=[trajectory,new_state];
    current_state=new_state;
end
end
```

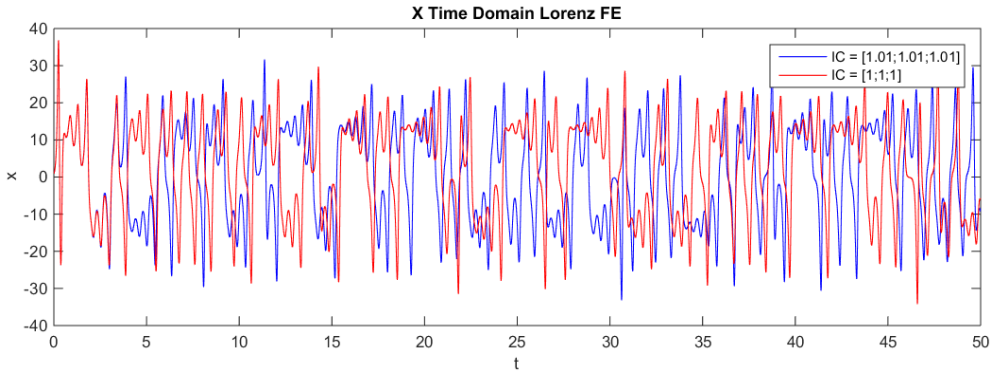
Generate two 50,000-point-long trajectories using $h = 0.01$ (that is, $N = 50,000$), one from the initial condition $[x, y, z]^T = [1, 1, 1]^T$ and the other from the initial condition $[x, y, z]^T = [1.01, 1.01, 1.01]^T$.

Make time-domain plots of the x coordinates (i.e., x versus t) for both trajectories—preferably on the same set of axes so you can compare them easily. The initial conditions are almost identical, but the time course of the x coordinate of the two trajectories should diverge fairly quickly. That's one of the main attributes of chaos.

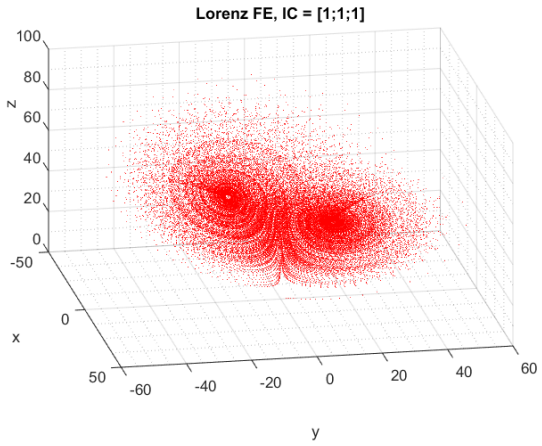
Make state-space plots of both trajectories—preferably in 3D (x vs. y vs. z)—or just y vs. x if your environment doesn't support 3D plotting. These two state-space plots should look pretty much identical to one another. That's the other main attribute of chaos.

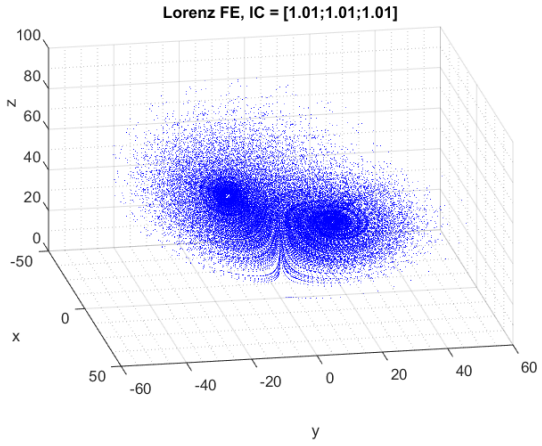
Please turn in your plots and a copy of your solver code.

Here is a time-domain plot of the first 5000 points of the x value:



Here are state-space plots for the different initial conditions:


Lorenz FE, IC = [1;1;1]
z
x
y


Lorenz FE, IC = [1.01;1.01;1.01]
z
x
y