Gabriel Rios
AOS 575
September 20, 2022

Problem set 1

1. (a) Determine an $O(\Delta x^2)$ accurate one-sided finite difference scheme.

   Choose points $j+1$ and $j+2$ to use as potential grid points for the approximation of $\frac{\partial\psi}{\partial x_j}$. Writing out the Taylor series expansion for ψ at each of these points (note that a superscript number in parentheses indicates differentiation order):

$$\psi_{j+1} = \psi_j + \Delta x\psi_j^{(1)} + \frac{1}{2}\Delta x^2\psi_j^{(2)} + \frac{1}{6}\Delta x^3\psi_j^{(3)} + ... \quad (1)$$

$$\psi_{j+2} = \psi_j + 2\Delta x\psi_j^{(1)} + 2\Delta x^2\psi_j^{(2)} + \frac{4}{3}\Delta x^3\psi_j^{(3)} + ... \quad (2)$$

   Using a linear combination of $\psi_j, \psi_{j+1}, \&\psi j+2$, try to eliminate the error term that will become the second-order term (currently the $O(\Delta x^3)$) term in these equations):

$$4\psi_{j+1} - \psi_{j+2} - 3\psi_j = (4-2-0)\Delta x\psi_j^{(1)} + (2-2)\Delta x^2\psi_j^{(2)} + \left(\frac{2}{3} - \frac{4}{3}\right)\Delta x^3\psi_j^{(3)} + ... \quad (3)$$

$$4\psi_{j+1} - \psi_{j+2} - 3\psi_j = 2\Delta x\psi_j^{(1)} - \frac{2}{3}\Delta x^3\psi_j^{(3)} + ... \quad (4)$$

$$\frac{4\psi_{j+1} - \psi_{j+2} - 3\psi_j}{2\Delta x} = -\frac{1}{3}\Delta x^2\psi_j^{(3)} + ... \quad (5)$$

   Therefore, $\frac{4\psi_{j+1}-\psi_{j+2}-3\psi_j}{2\Delta x}$ is an $O(\Delta x^2)$ accurate one-sided difference scheme.

   (b) How does the magnitude of the leading order truncation error term compare to that of the second-order central difference approximation?

   The second-order central difference approximation is constructed using the expansions at $j-1$ and $j-1$ as follows:

$$\psi_{j-1} = \psi_j - \Delta x\psi_j^{(1)} + \frac{1}{2}\Delta x^2\psi_j^{(2)} - \frac{1}{6}\Delta x^3\psi_j^{(3)} + ... \quad (6)$$

$$\psi_{j+1} = \psi_j + \Delta x\psi_j^{(1)} + \frac{1}{2}\Delta x^2\psi_j^{(2)} + \frac{1}{6}\Delta x^3\psi_j^{(3)} + ... \quad (7)$$

   Combining these to solve for $\psi_j^{(1)}$:

$$\frac{\psi_{j+1} - \psi_{j-1}}{2\Delta x} = \psi_j^{(1)} + \frac{1}{3}\Delta x^3\psi_j^{(3)} + ... \quad (8)$$

   Comparing the leading order terms from Equations 5 and 8, they are of the same magnitude, indicating they have similar accuracies and error magnitudes.

2. Expanding the equations given to get ψ^{n+1} as a function of ψ^n:

$$\psi^* = \psi^n + \alpha i\omega\Delta t\psi^n \quad (9)$$

$$\psi^{**} = \psi^n + \frac{1}{2}\alpha i\omega\psi^* = \psi^n + \frac{1}{2}\alpha i\omega\Delta t\psi^n - \frac{1}{2}\alpha(\omega\Delta t)^2\psi_n \quad (10)$$

$$\psi^{n+1} = \psi^n + \Delta t i\omega\Delta t\psi^{**} = \psi^n + i\omega\Delta t\psi^n - \frac{1}{2}(\omega\Delta t)^2\psi^n - \frac{1}{2}\alpha i(\omega\Delta t)^3\psi^n \quad (11)$$

(a) Find the value of α that gives the highest order of accuracy. To minimize the error in Equation 11, compare error terms to the Taylor series expansion about ψ^{n+1}, which when expanded to infinite terms, approximates an exact solution:

$$\psi^{n+1} = \psi^n + i\omega\Delta t\psi^n - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{6}i(\omega\Delta t)^3 + ... \tag{12}$$

$$\tag{13}$$

Comparing the term from Equation 11 with the α terrm to the corresponding $\omega\Delta t$ term from Equation 14:

$$-\frac{1}{6}i(\omega\Delta t)^3 = -\frac{1}{2}\alpha i(\omega\Delta t)^3 \tag{14}$$

$$\alpha = \frac{1}{3} \tag{15}$$

With $\alpha = \frac{1}{3}$, the approximation for ψ^{n+1} becomes third-order accurate, as the resultant leading error term will be $O((\omega\Delta t)^4)$ before dividing through by Δt, creating a third-order leading error term.

(b) Find the value of α that gives the largest stable time step, Δt. This will be done by using the maximum stable amplication factor A, which is 1, to solve for α using Equation 11. Doing so gets:

$$\frac{\psi^{n+1}}{\psi^n} = A = 1 + i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{2}\alpha i(\omega\Delta t)^3. \text{ Multiplying by the complex conjugate:} \tag{16}$$

$$||A||^2 = \left[1 + i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{2}\alpha i(\omega\Delta t)^3\right]\left[1 + i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 + \frac{1}{2}\alpha i(\omega\Delta t)^3\right] \tag{17}$$

$$||A||^2 = 1 \geq= 1 + \left(\frac{1}{4} - \alpha\right)(\omega\Delta t)^4 + \frac{1}{4}\alpha(\omega\Delta t)^6 \tag{18}$$

$$0 \geq \frac{1}{4} - \alpha + \frac{1}{4}\alpha^2(\omega\Delta t)^2. \text{ Solving for } \alpha: \tag{19}$$

$$\alpha \leq \frac{1}{2} \tag{20}$$

Therefore, $\alpha = \frac{1}{2}$ grants the largest stable timestep for this scheme.

(c) Using the results from parts (a) and (b) and Equation 16 for A, the expression for the A with the highest order of accuracy is:

$$A_a = 1 + i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{6}i(\omega\Delta t)^3 \tag{21}$$

$$A_b = 1 + i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{4}i(\omega\Delta t)^3 \tag{22}$$

Getting $||A||^2$ by multplying by the complex conjugate and getting the square root grants the plots seen in Figure 1.

(d) Using Equations 21 and 22, as well as the following equation to find relative phase error R, we get:

$$R = \frac{\theta}{\omega\Delta t} = \frac{1}{\omega\Delta t}\arctan\left(\frac{\text{Im(A)}}{\text{Re(A)}}\right) \rightarrow \tag{23}$$

$$R_a = \frac{1}{\omega\Delta t}\arctan\left(\frac{\omega\Delta t - \frac{1}{6}(\omega\Delta t)^3}{1 - \frac{1}{2}(\omega\Delta t)^2}\right) \tag{24}$$

$$R_b = \frac{1}{\omega\Delta t}\arctan\left(\frac{\omega\Delta t - \frac{1}{4}(\omega\Delta t)^3}{1 - \frac{1}{2}(\omega\Delta t)^2}\right) \tag{25}$$

These relative phase errors are plotted in Figure 2.

(e) No computational modes were found for this time-stepping scheme. This was determined by substituting the expression for A into various terms of ψ^n as shown below, and because there was single solution for α, there is no computational mode.

From Equation 11 and using $A = \frac{\psi^{n+1}}{\psi^n}$:

$$\psi^{n+1} = \psi^n + i\omega\Delta t\psi^n - \frac{1}{2}(\omega\Delta t)^2\psi^n - \frac{1}{2}\alpha i(\omega\Delta t)^3\psi^n \tag{26}$$

$$\psi^{n+1} = \psi^n \left[1 + i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{2}\alpha i(\omega\Delta t)^3\right] \tag{27}$$

$$\frac{\psi^{n+1} - \psi^n}{\Delta t} = \psi^{n'}\left[1 + \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{6}(\omega\Delta t)^2\right] \tag{28}$$

$$= i\omega\psi^n\left[1 + \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{6}(\omega\Delta t)^2\right] \tag{29}$$

$$\text{Substituting the expression for A:} \tag{30}$$

$$A^2\psi^{n-1} - A\psi^{n-1} = i\omega A\psi^{n-1}\Delta t\left[1 + \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{6}(\omega\Delta t)^2\right] \tag{31}$$

$$\text{Solving for A:} \tag{32}$$

$$A = i\omega\Delta t - \frac{1}{2}(\omega\Delta t)^2 - \frac{1}{6}i(\omega\Delta t)^3 \tag{33}$$

3. (a) A numerical model using Python was used to integrate the Lorenz equations using an explicit second-order Runge-Kutta scheme using the Heun method. The figures generated were done so with $\Delta t = 0.01$ and 15,000 timesteps with initial values $(x_0, y_0, z_0) = (1, 1, 1)$. The code has been emailed to Bob Hallberg and Sam Ditkovsky. The code is also appended to this document.

(b) The r values chosen were 9, 25, and 49. These values were chosen to get a range of chaotic behavior from the system. When $r = 9$, the system is modeled as attracting towards only one of the roots $\sqrt{r} = \pm 3$, with the system approaching the value very closely. For $r = 25$, the system oscillates around both roots of r (± 5) but stays at a further distance from the roots than the previous r value, indicating more chaotic behavior. For $r = 49$, the system is less distributed and appears more chaotic, although this may be a function of numerical scheme errors.

(c) This time-stepping scheme was chosen as a balance between accuracy, stability, and my ability to program a numerical scheme given my time constraints. Runge-Kutta methods allow for a relatively simple numerical scheme to achieve good accuracy and stability with little storage constraints, making it useful for a long-run model when chaotic behavior is being modeled. The Heun method of the Runge-Kutta second-order method was chosen because it allows for modest timesteps with a low amplification factor, making it a useful method to model a system for extended periods of time, assuming a sufficiently small Δt. The length of the timestep was chosen to be $\Delta t = 0.01$. This is below the maximum allowable timestep of 0.037 for the nominal case of $r = 25$, and was increased over this maximum to allow for the modeling of a system with a larger r value. After this value, the amplification becomes excessive and the system diverges.

In an ideal case, a more stable and efficient scheme such as RK4 would be better-suited to model this system, as it has an amplification factor orders of magnitude smaller than Heun's RK2, with a smaller phase error, despite a larger amount of storage needed due to the increased number of steps. Through this method, a longer model run could be performed for a wider range of r values, which is more applicable to a series of cases in atmospheric science.

Appendix

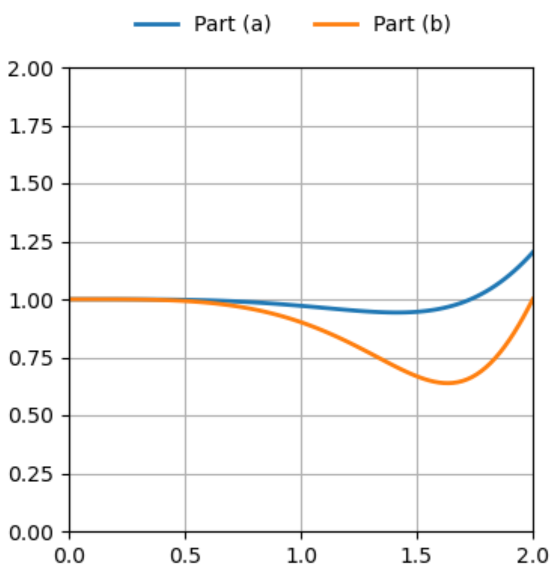


Figure 1: Magnitude of the amplification factor for part (a) $\alpha = \frac{1}{3}$ and part (b) $\alpha = \frac{1}{2}$.

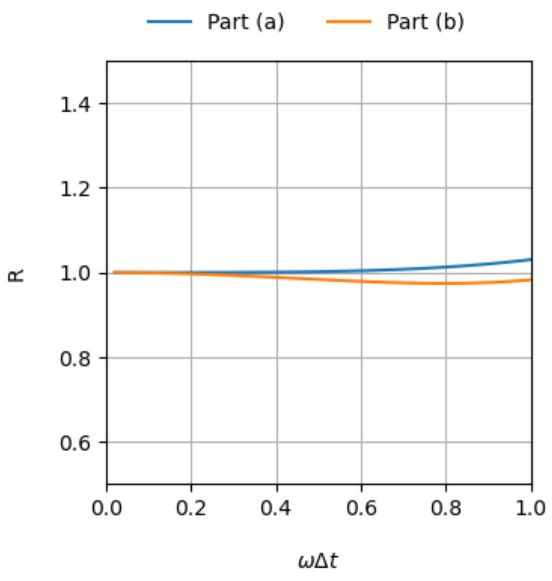
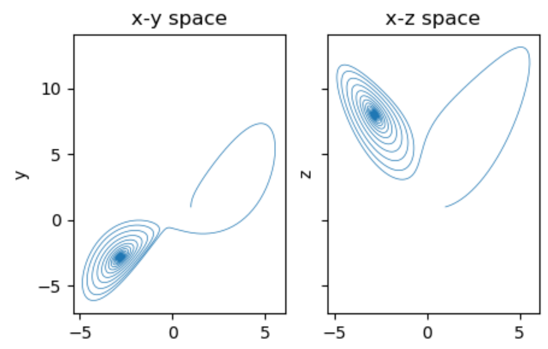
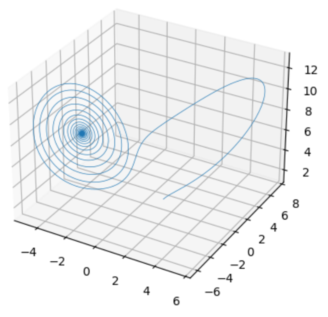


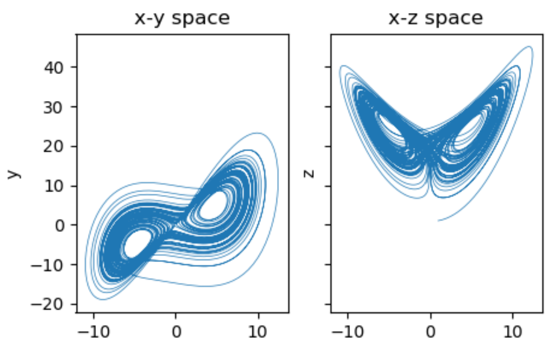
Figure 2: Magnitude of the relative phase error for part (a) $\alpha = \frac{1}{3}$ and part (b) $\alpha = \frac{1}{2}$.



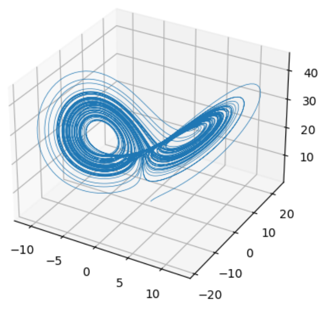
(a)



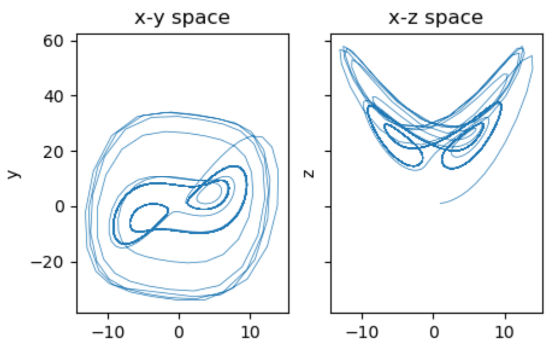
(b)



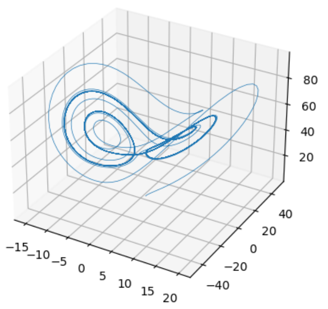
(c)



(d)



(e)



(f)

Figure 3: Plots for (a) $r = 9$, (b) $r = 25$, and (c) $r = 49$ as integrated by the chosen numerical method. Plots in x-y & x-z space are shown in the first two columns, with a plot in 3D space shown in the third.

aos_575-pset_1-code-rios

September 20, 2022

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

0.0.1 Problem 3: Lorenz equation discretization

This is ugly but it works. Future work would include creating a single Runge-Kutta method to account for all variables discretized herein.

[82]:
```python
# Define differential functions for each coordinate direction
def F_x(x, y, z, t, r):
    return -3*(x - y)
def F_y(x, y, z, t, r):
    return -x*z + r*x - y
def F_z(x, y, z, t, r):
    return x*y - z

# Define the RK2 Heun method for each coordinate direction.
# All three methods are equivalent, just for different directions.
def diff_x(x, y, z, t, dt, r):
    # Define Heun-specific coefficients
    a, b, c = 1, 1/2, 1
    # Calculate xi values
    xi_1 = x
    xi_2 = x + dt*a*F_x(x, y, z, t, r)
    # Calculate approximation for timestep (n+1)
    x_ = x + dt*(b*F_x(xi_1, y, z, t + c*dt, r) + b*F_x(xi_2, y, z, t + c*dt,␣
 ↪r))
    return x_
def diff_y(x, y, z, t, dt, r):
    a, b, c = 1, 1/2, 1
    xi_1 = y
    xi_2 = y + dt*a*F_y(x, y, z, t, r)
    y_ = y + dt*(b*F_y(x, xi_1, z, t + c*dt, r) + b*F_y(x, xi_2, z, t + c*dt,␣
 ↪r))
    return y_
def diff_z(x, y, z, t, dt, r):
    a, b, c = 1, 1/2, 1
    xi_1 = z
    xi_2 = z + dt*a*F_z(x, y, z, t, r)
    z_ = z + dt*(b*F_z(x, y, xi_1, t + c*dt, r) + b*F_z(x, y, xi_2, t + c*dt,␣
 ↪r))
    return z_

# Define initial values
r = 25
x_0, y_0, z_0 = 1, 1, 1
# Initialize arrays to store x, y, z values
x = np.array([x_0])
y = np.array([y_0])
z = np.array([z_0])

# Define time and timestep
dt, t_0 = 0.01, 0
t = np.array([t_0])
N = 15000 # number of steps

# Iterate through timesteps
for i in range(0, N):
    # Generate x^{n-1}
    x_ = diff_x(x[i], y[i], z[i], t[i], dt, r)
    x = np.append(x, x_)
    # Generate y^{n-1}
    y_ = diff_y(x[i], y[i], z[i], t[i], dt, r)
    y = np.append(y, y_)
    # Generate z^{n-1}
    z_ = diff_z(x[i], y[i], z[i], t[i], dt, r)
    z = np.append(z, z_)
    # Step forward in time
    t = np.append(t, t[i] + dt)
```