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
```

```python
    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)
```