Computer Project 7
Fourth Order Runge–Kutta for First–Order Systems

DUE: February 24, 2023

Introduction: We can easily extend the fourth order Runge–Kutta numerical method to systems of first–order differential equation of the form

$$
\begin{aligned}
x_1' &= f_1(t, x_1, x_2, \cdots, x_n), \qquad x_1(0) = x_1^0 \\
x_2' &= f_2(t, x_1, x_2, \cdots, x_n), \qquad x_2(0) = x_2^0 \\
&\vdots \\
x_n' &= f_n(t, x_1, x_2, \cdots, x_n), \qquad x_n(0) = x_n^0
\end{aligned}
$$

with minimal changes to the scheme. The similarity to the original method becomes clear if we rewrite the system in vector form.

$$
\vec{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \quad \vec{F}(t, \vec{x}) = \begin{bmatrix} f_1(t, x_1, x_2, \cdots, x_n) \\ f_2(t, x_1, x_2, \cdots, x_n) \\ \vdots \\ f_n(t, x_1, x_2, \cdots, x_n) \end{bmatrix}, \quad \vec{x}_0 = \begin{bmatrix} x_1^0 \\ x_2^0 \\ \vdots \\ x_n^0 \end{bmatrix}
$$

$$
\begin{aligned}
\frac{d\vec{x}}{dt} &= \vec{F}(t, \vec{x}) \\
\vec{x}(0) &= \vec{x}_0
\end{aligned}
$$

For a given final time, t_f, and number of steps, N, we run through the method exactly as before (with $\Delta t = t_f/N$). The only difference is that the operations become vector operations!

$$
\begin{aligned}
\vec{a}_i &= \vec{F}(t_i,\ \vec{x}_i) \\
\vec{b}_i &= \vec{F}\left(t_i + \frac{\Delta t}{2},\ \vec{x}_i + \frac{\Delta t}{2} \cdot \vec{a}_i\right) \\
\vec{c}_i &= \vec{F}\left(t_i + \frac{\Delta t}{2},\ \vec{x}_i + \frac{\Delta t}{2} \cdot \vec{b}_i\right) \\
\vec{d}_i &= \vec{F}(t_i + \Delta t,\ \vec{x}_i + \Delta t \cdot \vec{c}_i) \\
t_{i+t} &= t_i + \Delta t \\
\vec{x}_{i+1} &= \vec{x}_i + \frac{\Delta t}{6}\left(\vec{a}_i + 2\vec{b}_i + 2\vec{c}_i + \vec{d}_i\right)
\end{aligned}
$$

For this project, your goal is to define a Python function

```
vecRK4(vecFunc, init, startT, finalT, steps)
```

which is the vector implementation of Fourth Order Runge–Kutta. The various function arguments are as follows.

- `vecFunc`: a function that represents the right–hand side of the system of differential equations
- `init`: the vector of initial data
- `startT`: the starting time of the numerical simulation (typically 0)
- `finalT`: the ending time of the numerical simulation
- `steps`: an integer specifying the total number of steps in going from `startT` to `finalT`

`vecRK4` should return two arrays, `T` and `Ret`. `T` should be a one-dimensional array of the times (starting from `startT`, ending on `finalT`, and containing `steps`+1 total elements). `Ret` should be a two-dimensional numpy array containing the simulation data. Each *column* of `Ret` should contain the vector data associated to the corresponding time in `T`.[1]

As for the argument `vecFunc`, it should be a function of the form

```
vecFunc(t, vec)
```

which accepts a time, t, and a vector, `vec = [vec[0], vec[1],` $\cdots$ `vec[n-1]]`. It should return a numpy array of the same shape as the input `vec` but with entries updated by whatever is required by the right–hand side of the system of differential equations.

[1] The reason to arrange the data this way is to make it easier to plot. You may want to begin by storing each individual time increment as a *row* in the data structure. Then use `Ret = np.transpose(Ret)` to transpose the data into the required shape.

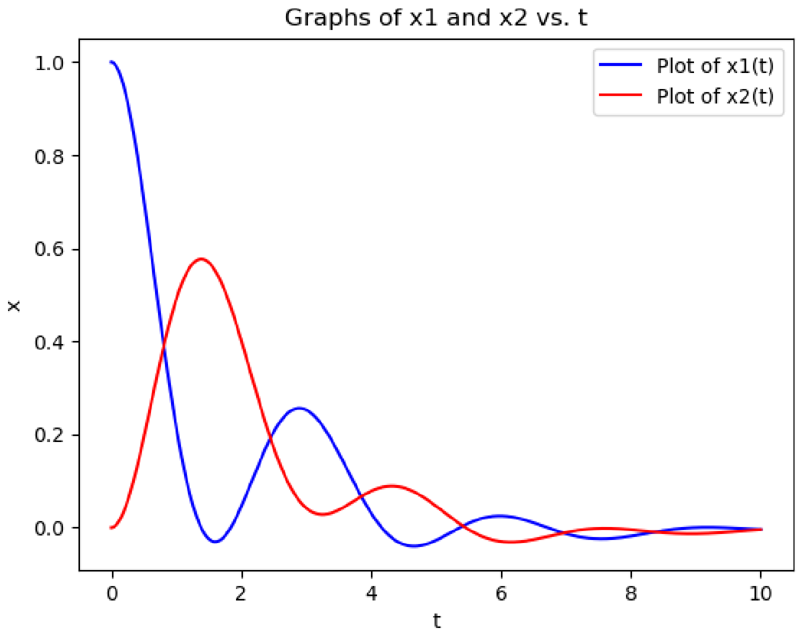
As an example, consider the system of first–order differential equations given below (which describes a pair of coupled spring–mass systems with damping).

$$
\begin{aligned}
x_1' &= x_3, & x_1(0) &= 1\\
x_2' &= x_4, & x_2(0) &= 0\\
x_3' &= -3x_1 + 2x_2 - x_3, & x_3(0) &= 0\\
x_4' &= 2x_1 - 2x_2 - x_4, & x_4(0) &= 0
\end{aligned}
$$

To code this system in Python, we can do something like the following.

```
import math
import numpy as np
import matplotlib.pyplot as plt

def vecRK4(vecFunc, init, startT, finalT, stps):
    # FILL IN YOUR CODE HERE
    return T, Ret

def F(t, vec):
    ret = np.zeros_like(vec)
    ret[0] = vec[2]
    ret[1] = vec[3]
    ret[2] = -3*vec[0] + 2*vec[1] - vec[2]
    ret[3] = 2*vec[0] - 2*vec[1] - vec[3]
    return ret

# Run RK-4 for the system
T, Ret = vecRK4(F, [1,0,0,0], 0, 10, 1000)

# Plot x_1 and x_2
plt.plot(T,Ret[0], color='blue', label = "Plot of x1(t)")
plt.plot(T,Ret[1], color='red', label = "Plot of x2(t)")
plt.xlabel('t')
plt.ylabel('x')
plt.title('Graphs of x1 and x2 vs. t')
plt.legend()
plt.show()
```

If your code is correct, you should see an image like the following.



Instructions: Submit a file with the sample code above but with your implementation of `vecRK4` filled in where prompted.