MATH 5620/6865: FINAL: JORDAN SAETHRE

This submission is for Jordan Saethre. The work was done in collaboration with Paul Mundt.

(1) (a) (10 pts) Use the fast Fourier transform to solve the following pure initial value problem to obtain $u(x, \tau)$:

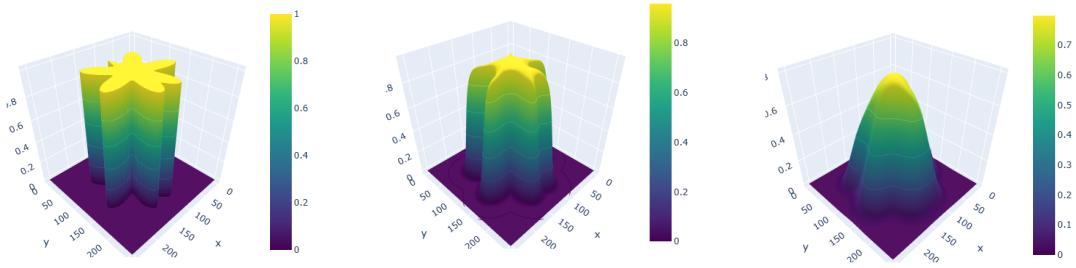
$$\begin{cases} u_t = \Delta u \\ u(x, y, 0) = u_0 \end{cases}$$

in the domain $[-\pi, \pi]^2$ with u_0 being the indicator function of the domain bounded by

$$\begin{cases} x = r\cos(\theta) \\ y = r\sin(\theta) \end{cases}$$

where $r = 0.5\pi + 0.2\pi\sin(6\theta)$ and $\theta \in [0, 2\pi]$.

FFT on 2D Heat Equation Output: $n = 256$ and $\tau = 0.01$ initial condition, after 2 iterations, and after 10 iterations.



FFT on 2D Heat Equation Code:

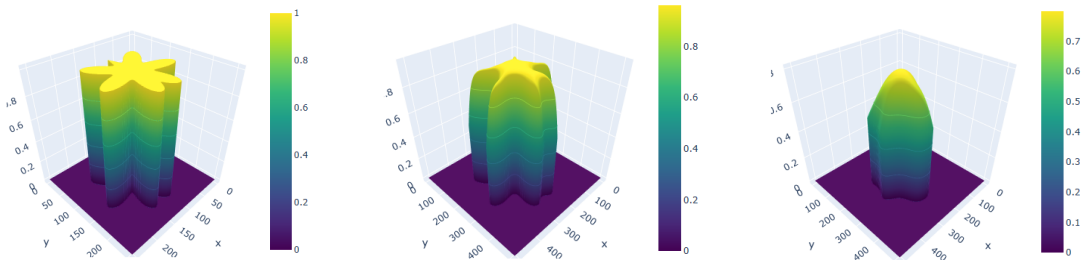
```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

a = -np.pi
b = np.pi
n = 256
tau = 0.01
k = np.linspace(-n/2,n/2+1, n)
x = np.linspace(a,b,n+1)
y = np.linspace(a,b,n+1)
iterates1 = 2
iterates2 = 10

# initial condition
x_theta = lambda theta: (np.pi/2 + (np.pi/5)
    *np.sin(6*theta))*np.cos(theta)
y_theta = lambda theta: (np.pi/2 + (np.pi/5)
```

Date: May 31, 2020.

```
    *np.sin(6*theta))*np.sin(theta)

def initial_mat(x_theta,y_theta,n, x):
    initial=np.zeros(shape=(n,n))
    for i in range(0,len(x)-1):
        for j in range(0,len(x)-1):
            ri=np.sqrt(x[i]**2 +x[j]**2)
            r=np.sqrt(x_theta(np.arccos(x[i]/ri))**2
                + y_theta(np.arccos(x[i]/ri))**2)
            if abs(ri)>abs(r):
                initial[i,j]=0
            else:
                initial[i,j]=1
    return(initial)

# solve with fourier transforms
def fourier_solve(u,k,tau):
    ker = np.fft.fftshift(np.exp(-k**2*tau))
    for i in range(0, len(x)-1):
        u[i] = np.fft.ifft(np.fft.fft(u[i])*ker)
    u = np.transpose(u)
    for j in range(0,len(y)-1):
        u[j] = np.fft.ifft(np.fft.fft(u[j])*ker)
    return u

def fourier_iterate(x_theta,y_theta,n,x,k,iterates):
    u = initial_mat(x_theta,y_theta,n,x)
    for i in range(0, iterates):
        u_next = fourier_solve(u,k,tau)
        u = u_next
    return u

def plot_surface(solution):
    surface_df = pd.DataFrame(solution)
    surface_df.to_csv('surface_data.csv', index = False)
    surface_data = pd.read_csv('surface_data.csv')
    fig = go.Figure(data=[go.Surface(z=surface_data.values,
        colorscale='Viridis')])
    fig.update_traces(contours_z=dict(show=True,
        usecolormap=True, highlightcolor="limegreen",
         project_z=True))
    fig.update_layout(title='Solution', autosize=False,
        width=500, height=500,
        margin=dict(l=65, r=50, b=65, t=90),
        xaxis = dict(visible = False))
    fig.show()

plot_surface(initial_mat(x_theta,y_theta,n,x))
plot_surface(fourier_iterate(x_theta,y_theta,n,x,k,iterates1))
plot_surface(fourier_iterate(x_theta,y_theta,n,x,k,iterates2))
```

(b) (10 pts) Set $\tau = 0.01$ and use N^2 points to discretize the domain with $N = 512$. At each step, after the $u(x, \tau)$ is solved, set a new initial condition as the indicator function of $u(x, \tau) > \frac{1}{2}$, solve the heat equation, and iterate again and again to model a motion of the boundary of the domain.

FFT on 2D Heat Equation Output: $n = 512$ and $\tau = 0.01$ initial condition, after 2 iterations, and after 10 iterations.



FFT on 2D Heat Equation Code:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go

# parameters
a = -np.pi
b = np.pi
n = 512
tau = 0.01
k = np.linspace(-n/2,n/2+1, n)
x = np.linspace(a,b,n+1)
y = np.linspace(a,b,n+1)
iterates1 = 2
iterates2 = 10

# initial condition
x_theta = lambda theta: (np.pi/2 + (np.pi/5)
    *np.sin(6*theta))*np.cos(theta)
y_theta = lambda theta: (np.pi/2 + (np.pi/5)
    *np.sin(6*theta))*np.sin(theta)

def initial_mat(x_theta,y_theta,n, x):
    initial=np.zeros(shape=(n,n))
    for i in range(0,len(x)-1):
        for j in range(0,len(x)-1):
            ri=np.sqrt(x[i]**2 +x[j]**2)
            r=np.sqrt(x_theta(np.arccos(x[i]/ri))**2
                + y_theta(np.arccos(x[i]/ri))**2)
            if abs(ri)>abs(r):
                initial[i,j]=0
            else:
                initial[i,j]=1
    return(initial)
```

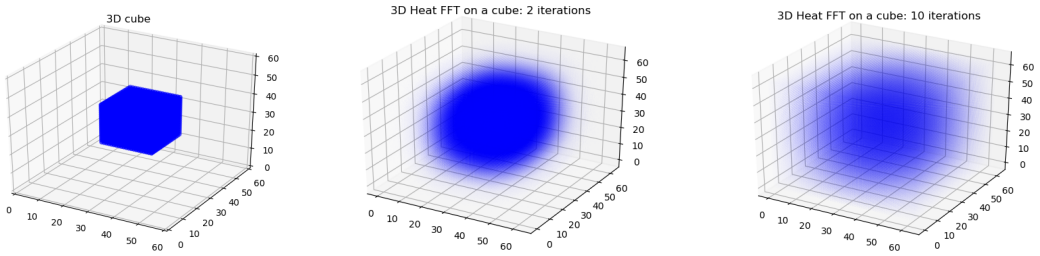
```
34 |
35 | # solve with fourier transforms
36 | def fourier_solve(u,k,tau):
37 |     ker = np.fft.fftshift(np.exp(-k**2*tau))
38 |     for i in range(0, len(x)-1):
39 |         u[i] = np.fft.ifft(np.fft.fft(u[i])*ker)
40 |     u = np.transpose(u)
41 |     for j in range(0,len(y)-1):
42 |         u[j] = np.fft.ifft(np.fft.fft(u[j])*ker)
43 |     return u
44 |
45 | def fourier_iterate(x_theta,y_theta,n,x,k,iterates):
46 |     u = initial_mat(x_theta,y_theta,n,x)
47 |     for i in range(0, iterates):
48 |         u_next = fourier_solve(u,k,tau)
49 |         u = u_next
50 |     for j in range(0, len(u[0])):
51 |             for k in range(0,len(u[0])):
52 |                 if (u_next[j,k] < 1/2):
53 |                     u_next[j,k] = 0.0
54 |     return u
55 |
56 | def plot_surface(solution):
57 |     surface_df = pd.DataFrame(solution)
58 |     surface_df.to_csv('surface_data.csv', index = False)
59 |     surface_data = pd.read_csv('surface_data.csv')
60 |     fig = go.Figure(data=[go.Surface(z=surface_data.values,
61 |         colorscale='Viridis')])
62 |     fig.update_traces(contours_z=dict(show=True,
63 |         usecolormap=True, highlightcolor="limegreen",
64 |         project_z=True))
65 |     fig.update_layout(title='Solution', autosize=False,
66 |         width=500,height=500,
67 |         margin=dict(l=65, r=50, b=65, t=90),
68 |         xaxis = dict(visible = False))
69 |     fig.show()
70 |
71 | plot_surface(initial_mat(x_theta,y_theta,n,x))
72 | plot_surface(fourier_iterate(x_theta,y_theta,n,x,k,iterates1))
73 | plot_surface(fourier_iterate(x_theta,y_theta,n,x,k,iterates2))
```

(c) (10 pts) What can you say about the motion? Why?

The motion of the boundary in part (b) evolves from a flower toward a circle. This is because the boundary is essentially a 1D heat equation at $z = 1/2$. Just like in the example we did in class for 1D heat the curve tended towards a straight line, since we are dealing with a loop it will tend towards the least curvature which is a circle.

(d) (20 pts) Do above for 3-dimensional case with an initial condition as the indicator function of the unit box (i.e., $[-1,1]^3$). The domain of your computation is $[-\pi,\pi]^3$. You can set $\tau = 0.05$ and $N = 64$.

FFT on 3D Heat Equation Output: $n = 64$ and $\tau = 0.05$ initial condition, after 2 iterations, and after 10 iterations.



FFT on 3D Heat Equation Code:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# parameters
a = -np.pi
b = np.pi
n = 64
tau = 0.05
k = np.linspace(-n/2,n/2+1, n)
x = np.linspace(a,b,n+1)
y = np.linspace(a,b,n+1)
z = np.linspace(a,b,n+1)
iterates1 = 2
iterates2 = 10

def initial_mat(x,y,z,n):
    initial=np.zeros(shape=(n,n,n))
    for i in range(0,len(x)-1):
        for j in range(0,len(x)-1):
            for k in range(0,len(x)-1):
                if abs(x[i]) < 1 and abs(y[j])
                < 1 and abs(z[k]) < 1:
                    initial[i,j,k]=1
                else:
                    initial[i,j,k]=0
    return(initial)

# solve with fourier transforms
def fourier_solve(u,k,tau):
    ker = np.fft.fftshift(np.exp(-k**2*tau))
    for k in range(0,len(x)-1):
        for i in range(0, len(x)-1):
            u[k,i] = np.fft.ifft(np.fft.fft(u[k,i])*ker)
```

```
35          u[k] = np.transpose(u[k])
36          for j in range(0,len(y)-1):
37              u[k,j] = np.fft.ifft(np.fft.fft(u[k,j])*ker)
38          u[k] = np.transpose(u[k])
39      u = np.transpose(u)
40      for k in range(0,len(x)-1):
41          for i in range(0, len(x)-1):
42              u[k,i] = np.fft.ifft(np.fft.fft(u[k,i])*ker)
43          u[k] = np.transpose(u[k])
44          for j in range(0,len(y)-1):
45              u[k,j] = np.fft.ifft(np.fft.fft(u[k,j])*ker)
46          u[k] = np.transpose(u[k])
47      u=np.transpose(u)
48      for k in range(0,len(x)-1):
49          for i in range(0, len(x)-1):
50              u[k,i] = np.fft.ifft(np.fft.fft(u[k,i])*ker)
51          u[k] = np.transpose(u[k])
52          for j in range(0,len(y)-1):
53              u[k,j] = np.fft.ifft(np.fft.fft(u[k,j])*ker)
54          u[k] = np.transpose(u[k])
55      return u
56
57  def fourier_iterate(x,y,z,k,iterates,tau):
58      u = initial_mat(x,y,z,n)
59      for i in range(0, iterates):
60          u_next = fourier_solve(u,k,tau)
61          u = u_next
62      return u
63
64  # plots
65  u_initial = initial_mat(x,y,z,n)
66  u = fourier_iterate(x,y,z,k,iterates1,tau)
67
68  X,Y,Z=u.nonzero()
69  X1,Y1,Z1=u_initial.nonzero()
70
71  fig=plt.figure(1)
72  ax = fig.add_subplot(111,projection='3d')
73  ax.scatter(X1,Y1,Z1,c='blue')
74  plt.xlim(0,60)
75  plt.ylim(0,60)
76  ax.set_zlim(0,60)
77  plt.title('3D cube')
78  plt.show()
79
80  fig=plt.figure(1)
81  ax = plt.axes(projection='3d')
82  ax.scatter(X,Y,Z,c='blue',s=u)
83  plt.title('3D Heat FFT on a cube: 2 iterations')
84  plt.show()
```

```
85 |
86 |u = fourier_iterate(x,y,z,k,iterates2,tau)
87 |
88 |fig=plt.figure(1)
89 |ax = plt.axes(projection='3d')
90 |ax.scatter(X,Y,Z,c='blue',s=u)
91 |plt.title('3D Heat FFT on a cube: 10 iterations')
92 |plt.show()
```

(2) In 1927, W. O. Kermack and A. G. McKendrick created a model in which they considered a fixed population with only three compartments: susceptible, $S(t)$: infected, $I(t)$: and removed, $R(t)$: The compartments used for this model consist of three classes:

$S(t)$ is used to represent the number of individuals not yet infected with the disease at time t, or those susceptible to the disease.

$I(t)$ denotes the number of individuals who have been infected with the disease and are capable of spreading the disease to those in the susceptible category.

$R(t)$ is the compartment used for those individuals who have been infected and then removed from the disease, either due to immunization or due to death. Those in this category are not able to be infected again or to transmit the infection to others.

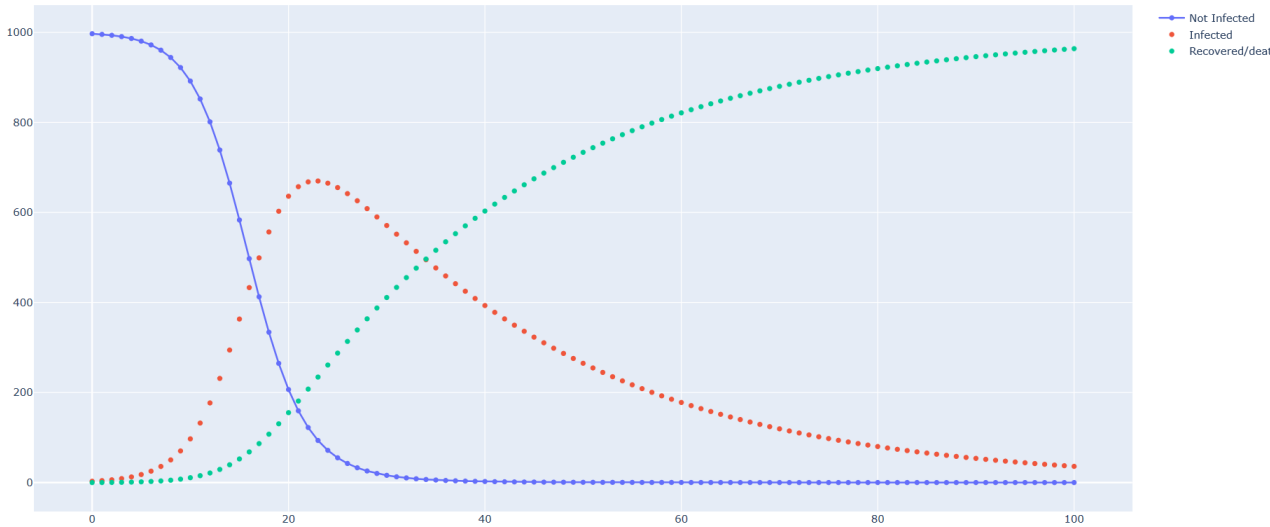
Using a fixed population, $N = S(t) + I(t) + R(t)$, Kermack and McKendrick derived the following equations:

$$\frac{dS}{dt} = -\frac{\beta SI}{N}$$
$$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I$$
$$\frac{dR}{dt} = \gamma I$$

where β and γ are two parameters.

(a) (20 pts) Use a high order method (e.g.,RK) you like to solve above ordinary differential equations with $S(0) = 997$, $I(0) = 3$, $R(0) = 0$, $\beta = 0.0004$, and $\gamma = 0.04$.

Solutions to SIR Model:



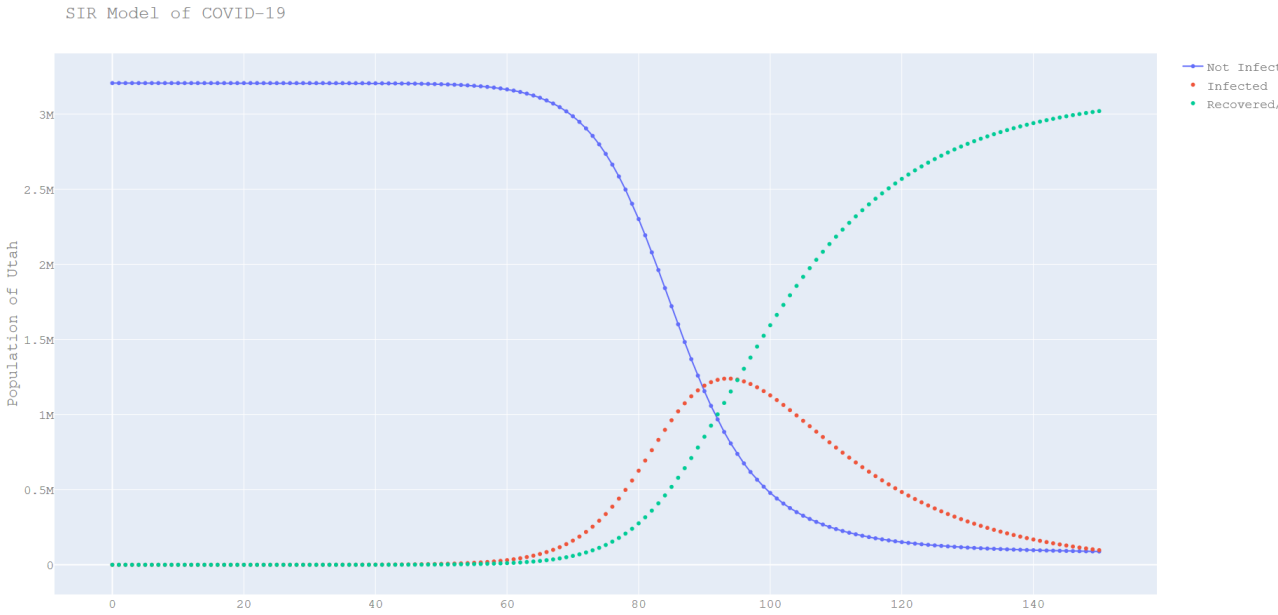
4th Order Runge Kutta to solve SIR Model Code:

```
import numpy as np
import scipy as sci
import plotly.graph_objects as go

beta=0.0004
gamma=0.04
S0=997
I0=3
R0=0
N=S0+I0+R0
S=lambda t,S,I,R: -beta*S*I
I=lambda t,S,I,R: beta*S*I-gamma*I
R=lambda t,S,I,R: gamma*I


def Runge_Kutta_4(fS,fI,fR,S,I,R,t,h,M):
    Sref=[S]
    Iref=[I]
    Rref=[R]
    tlist=[t]
    for i in range(1,M+1):
        F1S=h*fS(t,S,I,R)
        F1I=h*fI(t,S,I,R)
        F1R=h*fR(t,S,I,R)
        F2S=h*fS(t+0.5*h,S+0.5*F1S,I+0.5*F1I,R+0.5*F1R)
        F2I=h*fI(t+0.5*h,S+0.5*F1S,I+0.5*F1I,R+0.5*F1R)
        F2R=h*fR(t+0.5*h,S+0.5*F1S,I+0.5*F1I,R+0.5*F1R)
        F3S=h*fS(t+0.5*h,S+0.5*F2S,I+0.5*F2I,R+0.5*F2R)
        F3I=h*fI(t+0.5*h,S+0.5*F2S,I+0.5*F2I,R+0.5*F2R)
        F3R=h*fR(t+0.5*h,S+0.5*F2S,I+0.5*F2I,R+0.5*F2R)
        F4S=h*fS(t+h,S+F3S,I+F3I,R+F3R)
        F4I=h*fI(t+h,S+F3S,I+F3I,R+F3R)
        F4R=h*fR(t+h,S+F3S,I+F3I,R+F3R)
        S=S+(1/6)*(F1S+2*F2S+2*F3S+F4S)
        I=I+(1/6)*(F1I+2*F2I+2*F3I+F4I)
        R=R+(1/6)*(F1R+2*F2R+2*F3R+F3R)
        #R=N-S-I
        t+=h
        tlist.append(t)
        Sref.append(S)
        Iref.append(I)
        Rref.append(R)
    return(Sref,Iref,Rref,tlist)


Y=Runge_Kutta_4(S,I,R,S0,I0,R0,0,1,100)
#print(Y[2])
```

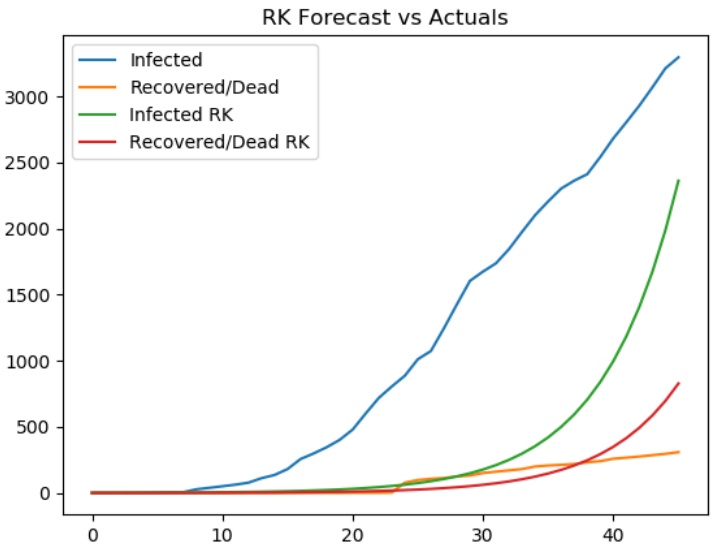
```
48 fig = go.Figure()
49 fig.add_trace(go.Scatter(x=Y[3], y=Y[0],
50     mode='lines+markers',name='Not Infected'))
51 fig.add_trace(go.Scatter(x=Y[3], y=Y[1],
52     mode='markers', name='Infected'))
53 fig.add_trace(go.Scatter(x=Y[3], y=Y[2],
54     mode='markers', name='Recovered/death'))
55 fig.show()
```

(b) (10 pts) Can you use the current data on the coronavirus from US, Utah, or Salt Lake City to predict the curve in the next two weeks, roughly?

COVID-19 Two week projection:



COVID-19 Forecast Vs. Actuals:



4th Order Runge Kutta COVID-19 Two week projection Code:

```
from requests import get
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import scipy as sci
import plotly.graph_objects as go

url='https://covidtracking.com/data/state/utah#historical'

response=get(url)
soup=BeautifulSoup(response.text,'html.parser')
skill=soup.find('tbody',
    class_=
    "state-history-table state-history-module--history--2YbCy")
pop=skill.find_all('tr')

Infected=[]
Recovered=[]
Dead=[]
for a in pop:
    info=a.find_all('td')
    Infected.append(info[3].get_text())
    Recovered.append(info[6].get_text())
    Dead.append(info[7].get_text())

COVID=pd.DataFrame(zip(Infected,Recovered,Dead),
    columns=['Infected','Recovered','Dead'])
COVID['Infected']=COVID['Infected'].str.replace(',','')
COVID['Recovered']=COVID['Recovered'].str.replace('N/A','0')
COVID['Dead']=COVID['Dead'].str.replace('N/A','0')
COVID['Infected']=pd.to_numeric(COVID['Infected'])
COVID['Recovered']=pd.to_numeric(COVID['Recovered'])
COVID['Dead']=pd.to_numeric(COVID['Dead'])

N=3206000
COVID['S']=N-COVID['Infected']
COVID['I']=COVID['Infected']
COVID['R']=COVID['Recovered']+COVID['Dead']

Sh=list(COVID['S'])
Ih=list(COVID['I'])
Rh=COVID['R']
SS=Sh[::-1]
II=Ih[::-1]
RR=Rh[::-1]
beta=[((SS[i+1]-SS[i])/(SS[i]*II[i])) for i in range(len(Sh)-1)]

```

```
T=np.linspace(0,len(Sh)-1,len(Sh))
import numpy as np
from scipy.optimize import curve_fit
from scipy.integrate import odeint

def fitfuncg(t, gamma):
    'Function that returns Ca computed from an ODE for a k'
    def myode(I,t):
        return gamma*I


    I0=Ih[0]
    Casol = odeint(myode, I0, T)
    return(Casol[:,0])

k_fitg, kcovg = curve_fit(fitfuncg, T, Ih, p0=1.3)

import matplotlib.pyplot as plt
II=Ih[::-1]
RR=Rh[::-1]
plt.figure(1)
plt.plot(T,II, label='Infected')
plt.plot(T,RR, label='RD')
plt.legend()
plt.show()

beta=0.000000073
gamma=abs(k_fitg[0])
S0=3205999
I0=1
R0=0
N=S0+I0+R0
S=lambda t,S,I,R: -beta*S*I
I=lambda t,S,I,R: beta*S*I-gamma*I
R=lambda t,S,I,R: gamma*I


def Runge_Kutta_4(fS,fI,fR,S,I,R,t,h,M):
    Sref=[S]
    Iref=[I]
    Rref=[R]
    tlist=[t]
    for i in range(1,M+1):
        F1S=h*fS(t,S,I,R)
        F1I=h*fI(t,S,I,R)
        F1R=h*fR(t,S,I,R)
        F2S=h*fS(t+0.5*h,S+0.5*F1S,I+0.5*F1I,R+0.5*F1R)
        F2I=h*fI(t+0.5*h,S+0.5*F1S,I+0.5*F1I,R+0.5*F1R)
        F2R=h*fR(t+0.5*h,S+0.5*F1S,I+0.5*F1I,R+0.5*F1R)
        F3S=h*fS(t+0.5*h,S+0.5*F2S,I+0.5*F2I,R+0.5*F2R)
```

```
        F3I=h*fI(t+0.5*h,S+0.5*F2S,I+0.5*F2I,R+0.5*F2R)
        F3R=h*fR(t+0.5*h,S+0.5*F2S,I+0.5*F2I,R+0.5*F2R)
        F4S=h*fS(t+h,S+F3S,I+F3I,R+F3R)
        F4I=h*fI(t+h,S+F3S,I+F3I,R+F3R)
        F4R=h*fR(t+h,S+F3S,I+F3I,R+F3R)
        S=(S+(1/6)*(F1S+2*F2S+2*F3S+F4S))
        I=(I+(1/6)*(F1I+2*F2I+2*F3I+F4I))
        R=(R+(1/6)*(F1R+2*F2R+2*F3R+F3R))
        #R=N-S-I
        t+=h
        tlist.append(t)
        Sref.append(S)
        Iref.append(I)
        Rref.append(R)
    return(Sref,Iref,Rref,tlist)


Y=Runge_Kutta_4(S,I,R,S0,I0,R0,0,1,150)

fig = go.Figure()
fig.add_trace(go.Scatter(x=Y[3], y=Y[0],mode='lines+markers',
    name='Not Infected'))
fig.add_trace(go.Scatter(x=Y[3], y=Y[1],mode='markers',
    name='Infected'))
fig.add_trace(go.Scatter(x=Y[3], y=Y[2],mode='markers',
    name='Recovered/death'))
fig.update_layout(title="SIR Model of COVID-19",
    xaxis_title="Time (days)",yaxis_title="Population of Utah",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="#7f7f7f"
    )
)
fig.show()

Wiz=Y[3]
Ye=Y[1]
Yeet=Y[2]
plt.figure(2)
plt.plot(T,II, label='Infected')
plt.plot(T,RR, label='Recovered/Dead')
plt.plot(Wiz[0:len(Sh)],Ye[0:len(Sh)], label='Infected RK')
plt.plot(Wiz[0:len(Sh)],Yeet[0:len(Sh)],
    label='Recovered/Dead RK')
plt.title('RK Forecast vs Actuals')
plt.legend()
plt.show()
```

(3) (20 pts) State what did you learn from numerical analysis and how do you think on this direction as a branch of Mathematics.

Numerical analysis has taught me the fundamentals of modeling and solving many problems in many different fields especially those related to engineering. Some of the highlights of these topics include:

- Derivative and Integral approximation schemes including Taylor Series expansion, Runge-Kutta, Newton-Cotes formulas, and Lagrangian interpolation.
- Numerically solve initial value problems with Multi-step methods such as the popular Predictor-Corrector using Adams-Bashforth and Adams-Moulton formulas.
- Numerically solve two point boundary problems with Shooting Method and Finite Difference Method.
- Methods for solving partial differential equations of the various types with Dirichlet, Neumann, and Robin boundary conditions:
  - **Parabolic** $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$. Example: heat diffusion equation
  - **Elliptic** $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$. Example: Laplace equation
  - **Hyperbolic** $\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} = 0$ Example: wave equation

  Some of the methods we learned for solving some of these PDE's include Explicit, Implicit, and Crank Nicolson for 1D heat and Alternating Direction Implict method and Fast Fourier Transform for 3D heat.

What I enjoyed most about this class is that the methods we have learned are applicable to a huge range of interesting problems. For example with my knowledge of methods to solve the heat equation I am now equipped with the necessary tools to model and determine the parameters of a heat sink on an electrical board. Another example would be using Runge-Kutta to forecast the spread of disease. Yet another example in an even more seemingly unrelated field would be modeling the price of a stock options with the Black-Scholes equation which is a PDE that is a function of the underlying stock.