Adapting the SIR Rumor Spreading Model: Limitations, Modifications, and Implications

The primary limitation of the reference model that we will be taking into account is that it does not account for how relevant a rumor is. The relevance of the rumor plays a critical role in how information is spread. The more relevant a rumor is the more likely it is for ignorants to become spreaders and spreaders to stay spreaders while it decreases the likelihood of spreaders becoming stiflers since they are less inclined to stop the discussion about the rumor. This decreases the likelihood of spreaders becoming stiflers such that the rumor will be spread to a wider network if the relevance was higher. The model does not incorporate this variable, which is an essential part to how long a rumor can stay in circulation.

In our adaptation of the aforementioned model, we focus on the inclusion of relevancy as a rate $r \in (0, 1)$. This rate directly impacts the acceleration or deceleration of the information being spread as it affects probability in which an ignorant person will become a spreader. If the relevancy rate increases, the ignorant population is more likely to become spreaders. In the model for the Spreader Density (S(t)) relevance rate affects the rate at which spreaders are created from ignorant individuals. A higher *r* in this context leads to a higher proportion of ignorant people turning into spreaders. In this model *r* also affects the decay rate of spreaders. A higher *r* means that the population of spreaders will be higher in correspondence to the *r* value. For this to hold true, it also means that the rate of ignorants becoming spreaders and spreaders becoming stiflers will decrease. In the model for Stifler Density (R(t)), *r* affects the probability of ignorant and spreader individuals becoming stiflers. This directly impacts the growth rate of the stifler population density as a higher value of *r* decreases the rate at which ignorant and spreader individuals become stiflers as the rumor's life increases.

In our improved model we account for this relevance rate in each of the mean field equations:

$$\frac{dI(t)}{dt} = \; - \; krI(t)S(t)$$

$$\frac{dS(t)}{dt} = \; k\lambda rI(t)S(t) \; - \; kS(t)(\frac{\gamma}{r}S(t) \; + \frac{\eta}{r}R(t)) \; - \; \delta rS(t)$$

$$\frac{dR(t)}{dt} = \; k(1 \; - \; \lambda)rI(t)S(t) \; + \; kS(t)(\gamma rS(t) \; + \; \eta rR(t)) \; + \frac{\delta}{r}S(t)$$

The first equation we derived using our new population density equations was the f(z) equation that represents the growth and death of a rumor according to the stifler population. To do this, we determined our A, B, and C values.

$$\frac{dR(t)}{dI(t)} = \frac{k(1-\lambda)rI(t)S(t)+kS(t)(\gamma rS(t)+\eta rR(t))+\frac{\delta}{r}S(t)}{-krI(t)S(t)}$$

$$= \frac{k(1-\lambda)rI(t)S(t)}{-krI(t)S(t)} + \frac{kS(t)(\gamma rS(t)+\eta rR(t))}{-krI(t)S(t)} + \frac{\frac{\delta}{r}S(t)}{-krI(t)S(t)}$$

$$= - \; (1 \; - \; \lambda) \; + \; \frac{\gamma(1-I(t)-R(t)))}{-I(t)} \; + \frac{\eta R(t)}{-I(t)} + \; \frac{\delta}{-kr^2 I(t)}$$

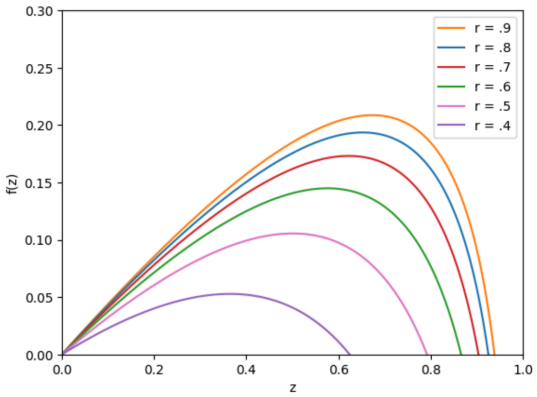
$$= \lambda \; - \; 1 \; + \; \frac{\gamma}{-I(t)} + \; \gamma \; + \; \frac{(-\gamma + \; \eta)R(t)}{-I(t)} + \frac{\delta}{-kr^2 I(t)}$$

$$= (- \; 1 \; + \; \lambda \; + \; \gamma) \; - \; (\gamma \; + \frac{\delta}{kr^2})(\frac{1}{I(t)}) \; + \; (\eta \; - \; \gamma)(\frac{R(t)}{I(t)})$$

$$A \; = \lambda \; + \; \gamma \; - \; 1 \qquad B = \; \frac{\delta}{kr^2} \; + \; \gamma \qquad C = \; \eta \; - \; \gamma$$

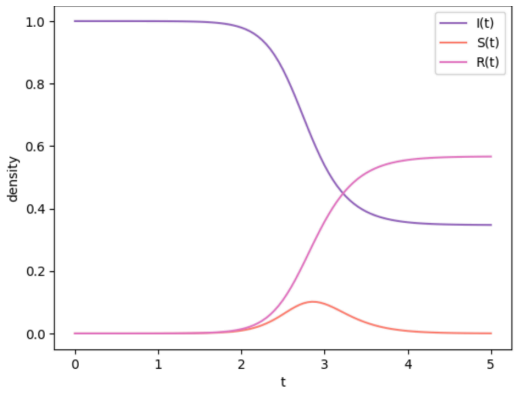
Then we were able to create our f(z) graph in relation to our changing variable of r to see how this affects the peak of the rumor as well as the amount of growth/ traction the rumor was able to get within a controlled population. The reason we only have to calculate the A, B, and C variables is because our new variable, *r*, is not affiliated with time, and because of this it will not change anything within the final equations that were given within the original model. We kept

constant values for all the other variables and modeled the f(z) graph so that we can see the effect that *r* would have.

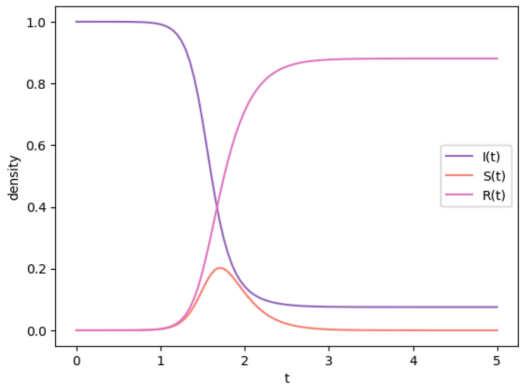


Through this graph, we could determine that the *r* values closely reflected our expectations. The higher the *r* value was, the higher and longer the rumor went within the graph, meaning that its growth and its peak were able to be reached further within the model. Since the x- axis represents the population growth, this would mean that as the population grew, the rumor grew in proportion to the value of *r*, reaching both a higher peak and a further equilibrium point. A high peak shows that the rumor is able to grow larger than the smaller *r* value peaks, representing that it held more value and was more important within the population. The equilibrium point also represents a similar idea, where the smaller *r* values have smaller equilibrium points and larger *r* values have larger equilibrium points. This is because when the rumor is more relevant it will take longer for the rumor to completely die out, meaning that the spreader density will take longer to return back to zero.

After finding the f(z) graph we were able to create two different graphs that show the varying population densities in response to different *r* values. For the graphs, the original model detailed that all the different population densities totaled together should equal 1, and the system reaches equilibrium once the spreaders die out and the rumor is no longer being spread.

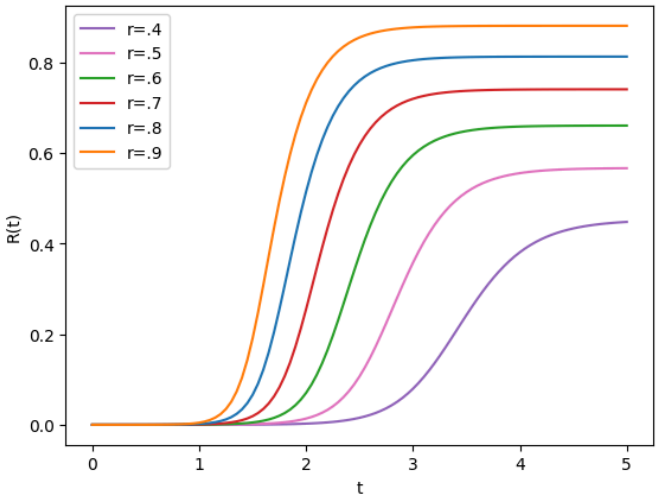


R = 0.4

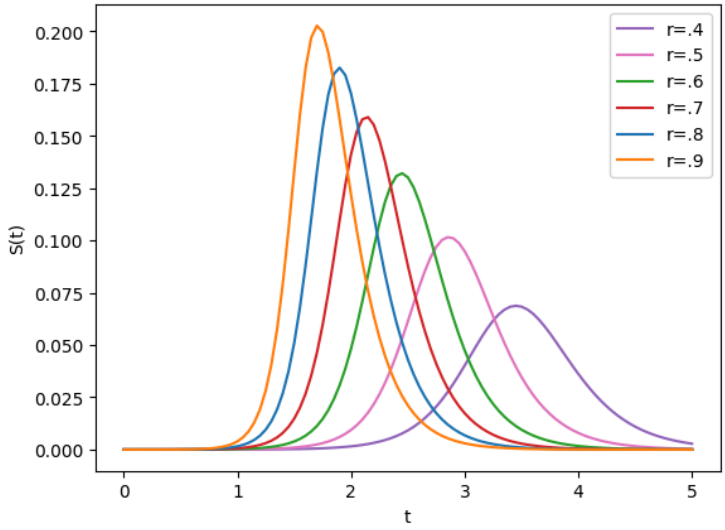


R = 0.9

According to our model, a smaller *r* value results in the equilibrium of the system being reached later and at a smaller spreader density, while a bigger *r* results in a larger spreader density with an earlier point of equilibrium. Proportionally, the bigger *r* will also result in a higher value of stiflers. This is because the population of spreaders is proportional to the population of stiflers due to the fact that all spreaders will eventually become stiflers once equilibrium is reached. So, when the peak of the spread is reached at a smaller density of spreaders, this will be associated with a smaller density in stiflers and a higher density in ignorants, while the inverse also holds true. These graphs are able to represent that our added variable *r* (relevancy) is able to fluctuate the population densities within a model to accurately represent the population changes over time.



Within this model, we compared multiple different values of *r* in correspondence to the population density of spreaders. It can be determined that the spreaders will increase with an increase in *r* as well as reach a peak earlier in the system. This is because the more relevant a rumor is the faster it will be able to spread and thus the faster it will be able to reach an equilibrium state. Within the graph, we can see that as the *r* value decreases, the amount of times it takes to reach a peak and return to equilibrium is longer, due to the rumor taking longer to spread within the population.



Finally, it can be seen that a changing value of *r* clearly shows different corresponding population densities within the stiflers as well. The most relevant information that can be determined by this model is the point in which the model is able to reach equilibrium. As mentioned previously, our data has proven that a high *r* value will lead to a higher density of stiflers since the rumor is likely to have led to a greater number of spreaders. This holds true within this model as well. We were able to determine that the spreader density grew at a faster rate as well as tapered off earlier within the system when *r* was a larger value. This is because the growth of spreaders and the growth of stiflers is proportional to each other. So when the spreader

population grows at a faster rate (as analyzed in the previous graph) the stifler population will also be growing at the same rate, with the stifler population eventually overtaking the spreader population in the long run and allowing the system to reach a steady state.

Conclusion

Our model was able to include one of the many factors that were not included within the reference model in order to better replicate how a rumor travels within a population. However this does not mean that our model is reflective of a realistic population density model. While our model addresses one limitation of the reference, both the reference and our adapted model assume that the population is a homogeneous network. The models do not consider the individuality of people within the population and as a result it also fails to consider the behavior dynamics within networks. It assumes that the individuals within the model will behave in a calculated way when in reality, individuals within populations react in varying ways when encountering a rumor.

Signed Statement

- Analyzing reference model: Khadeejah, Lizeth, Narin
    - Understanding derivation of reference models
    - Meaning of model
    - Equilibrium of reference models
- Deriving adapted model: Khadeejah, Lizeth, Martha, Narin
    - Limitations of reference model
    - Introduction of new variable
    - Impact of variable on each density population
- Analyzing adapted model: Narin, Martha
    - Shifted Equilibrium point
    - New values of A, B, C
    - Different responses to r
- Technical contributions: Martha, Narin
    - Graphical analysis x3
        - Render different graphs with varied r values

Signed by: **Khadeejah Kabir, Lizeth Vera, Martha Salazar, Narin Maisha**

References

Galam, S., Kosfeld, M., Zhao, L. J., Nekovee, M., Isham, V., Peterson, W. A., & Kimmel, A. J. (2012, November 1). *Sir rumor spreading model in the new media age*. Physica A: Statistical Mechanics and its Applications. https://www.sciencedirect.com/science/article/pii/S037843711200934X?via%3Dihub#aep-abstract-id19

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

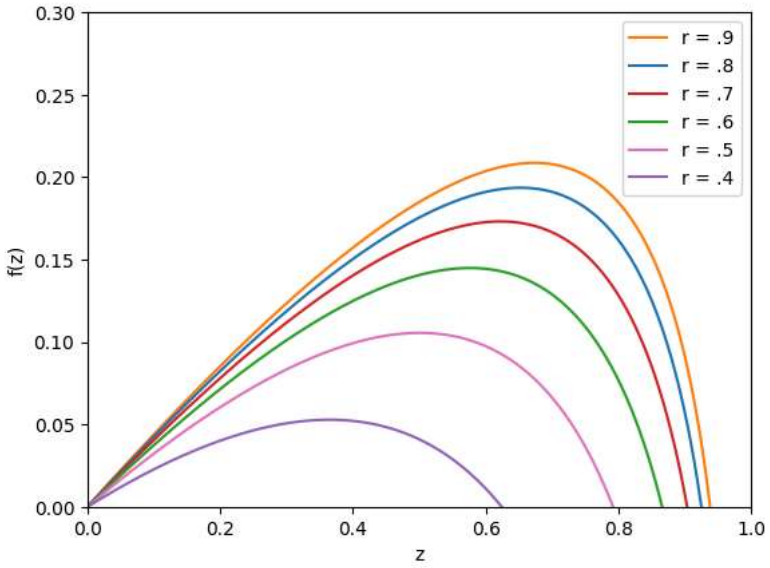
Graphing f(z) for Updated Model

```
def model_func(r, y):
  lambdaa = .5
  gamma = .1
  eta = .2
  delta = .8
  k = 20

  A = lambdaa +(gamma) - 1
  B = (delta/(k*(r**2))) + (gamma)
  C = eta - gamma

  if y == 1:
    value = ((A/(C+1)) +1)*y - (A/(C+1)) + (B/C)
  if y != 1:
    value = ((A/(C+1)) +1)*y + ((A/(C+1)) - B/C)*((1-y)**(-C))- (A/(C+1)) + (B/C)
  return value


t = np.arange(0, 1, 0.001)
N = len(t)
r_t_values = np.zeros(N)
r_y_values = np.zeros(N)
r_t_values[0] = 0
r_y_values[0] = 0

for i in range(0, N):
  a = model_func(.8, t[i])
  r_y_values[i] = a
  r_t_values[i] = i


t = np.arange(0, 1, 0.001)
N = len(t)
r2_t_values = np.zeros(N)
r2_y_values = np.zeros(N)

r2_t_values[0] = 0
r2_y_values[0] = 0

for i in range(0, N):
  a = model_func(.7, t[i])
  r2_y_values[i] = a
  r2_t_values[i] = i


t = np.arange(0, 1, .001)
N = len(t)
r3_t_values = np.zeros(N)
r3_y_values = np.zeros(N)

r3_t_values[0] = 0
r3_y_values[0] = 0

for i in range(0, N):
  a = model_func(.6, t[i])
  r3_y_values[i] = a
  r3_t_values[i] = i
```

```
t = np.arange(0, 1, 0.001)
N = len(t)
r4_t_values = np.zeros(N)
r4_y_values = np.zeros(N)

r4_t_values[0] = 0
r4_y_values[0] = 0

for i in range(0, N):
  a = model_func(.4, t[i])
  r4_y_values[i] = a
  r4_t_values[i] = i


t = np.arange(0, 1, 0.001)
N = len(t)
r5_t_values = np.zeros(N)
r5_y_values = np.zeros(N)

r5_t_values[0] = 0
r5_y_values[0] = 0

for i in range(0, N):
  a = model_func(.5, t[i])
  r5_y_values[i] = a
  r5_t_values[i] = i


t = np.arange(0, 1, 0.001)
N = len(t)
r6_t_values = np.zeros(N)
r6_y_values = np.zeros(N)

r6_t_values[0] = 0
r6_y_values[0] = 0

for i in range(0, N):
  a = model_func(.3, t[i])
  r6_y_values[i] = a
  r6_t_values[i] = i


t = np.arange(0, 1, 0.001)
N = len(t)
r7_t_values = np.zeros(N)
r7_y_values = np.zeros(N)

r7_t_values[0] = 0
r7_y_values[0] = 0

for i in range(0, N):
  a = model_func(.9, t[i])
  r7_y_values[i] = a
  r7_t_values[i] = i


t = np.arange(0, 1, 0.001)
N = len(t)
r8_t_values = np.zeros(N)
r8_y_values = np.zeros(N)

r8_t_values[0] = 0
r8_y_values[0] = 0

for i in range(0, N):
  a = model_func(1, t[i])
  r8_y_values[i] = a
  r8_t_values[i] = i
```

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(t, r7_y_values, color='tab:orange', label='r = .9')
ax.plot(t, r_y_values, color='tab:blue', label='r = .8')
ax.plot(t, r2_y_values, color='tab:red', label='r = .7')
ax.plot(t, r3_y_values, color='tab:green', label='r = .6')
ax.plot(t, r5_y_values, color='tab:pink', label='r = .5')
ax.plot(t, r4_y_values, color='tab:purple', label='r = .4')

ax.set_xlabel("z")
ax.set_ylabel("f(z)")

ax.set_xlim(0,1)
ax.set_ylim(0,.3)
ax.legend()
```

```
<matplotlib.legend.Legend at 0x78ea5fd7c4f0>
```



Runge Kutta Method Original Model

Using λ = .5, γ = .1, η = .2, δ = .8, k = 20

[ ] *6 cells hidden*

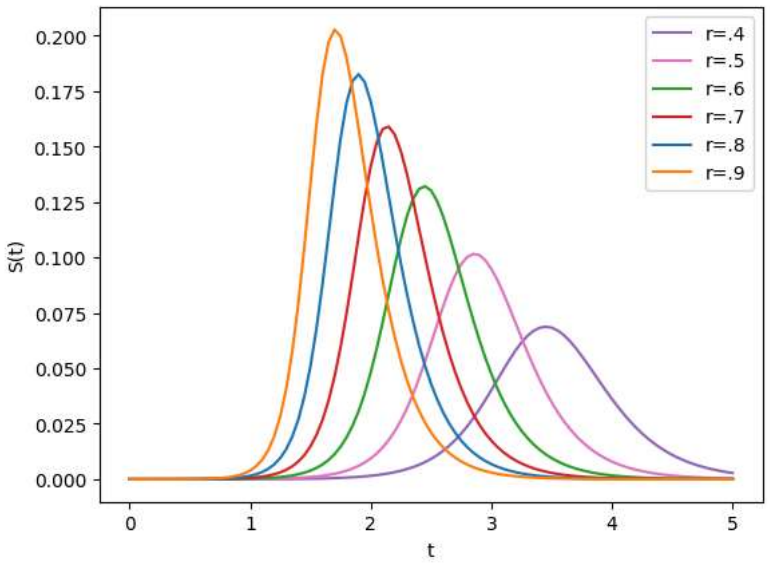
Runge Kutta With Updated Equation

Using λ = .5, γ = .1, η = .2, δ = .2, r = .2

```
def ignorant(r, t, I, S):
  k = 20
  value = -k * r * I * S
  return value

def spreader(r, t, I, S, R):
  lambdaa = .5
  gamma = .1
  eta = .2
  delta = .8
  k = 20
  value = k*I*S*lambdaa*r - (k*S*(1/r))*((gamma)*S + (eta)*R) - (delta*r)*S
  return value

def stifler(r, t, I, S, R):
  lambdaa = .5
  gamma = .1
  eta = .2
  delta = .8
  k = 20
  value = ((1-lambdaa)*k*I*S)*r + (k*S*(r))*((gamma)*S + (eta)*R) + (delta/r)*S
  return value


#Runge-Kutta Method
def runge_kutta(r):
  a = 0
  b = 5
  alpha = 0  #intial condition
  h = .05
  N = int((b - a)/h)

  t_values = np.zeros(N+1)
  S_y_values = np.zeros(N+1)
  I_y_values = np.zeros(N+1)
  R_y_values = np.zeros(N+1)

  #Define the intial values for S(t), I(t), R(t)
  p = 10**6#Given as variable N in the paper
  S_0 = 1/p
  I_0 = (p-1)/p
  R_0 = 0
# Store the initial values
  t_values[0] = a
  S_y_values[0] = S_0
  I_y_values[0] = I_0
  R_y_values[0] = R_0

  for i in range(N):

    #Stage 1
    K1I = ignorant(r, t_values[i], I_y_values[i], S_y_values[i])
    K1S = spreader(r, t_values[i], I_y_values[i], S_y_values[i], R_y_values[i])
    K1R = stifler(r, t_values[i], I_y_values[i], S_y_values[i], R_y_values[i])

    #Stage 2
    K2I = ignorant(r, t_values[i] + h/2, I_y_values[i] + h*.5*K1I, S_y_values[i] + h*.5*K1S)
    K2S = spreader(r, t_values[i] + h/2, I_y_values[i] + h*.5*K1I, S_y_values[i] + h*.5*K1S, R_y_values[i] + h*.5*K1R)
    K2R = stifler(r, t_values[i] + h/2, I_y_values[i] + h*.5*K1I, S_y_values[i] + h*.5*K1S, R_y_values[i] + h*.5*K1R)

    #Stage 3
    K3I = ignorant(r, t_values[i] + h/2, I_y_values[i] + h*.5*K2I, S_y_values[i] + h*.5*K2S)
    K3S = spreader(r, t_values[i] + h/2, I_y_values[i] + h*.5*K2I, S_y_values[i] + h*.5*K2S, R_y_values[i] + h*.5*K2R)
    K3R = stifler(r, t_values[i] + h/2, I_y_values[i] + h*.5*K2I, S_y_values[i] + h*.5*K2S, R_y_values[i] + h*.5*K2R)

    #Stage 4
    K4I = ignorant(r, t_values[i] + h, I_y_values[i] + h*K3I, S_y_values[i] + h*K3S)
    K4S = spreader(r, t_values[i] + h, I_y_values[i] + h*K3I, S_y_values[i] + h*K3S, R_y_values[i] + h*K3R)
    K4R = stifler(r, t_values[i] + h, I_y_values[i] + h*K3I, S_y_values[i] + h*K3S, R_y_values[i] + h*K3R)

    S_y_values[i+1] = S_y_values[i] + (1/6)*(K1S + 2*K2S + 2*K3S + K4S)*h
    I_y_values[i+1] = I_y_values[i] + (1/6)*(K1I + 2*K2I + 2*K3I + K4I)*h
    R_y_values[i+1] = R_y_values[i] + (1/6)*(K1R + 2*K2R + 2*K3R + K4R)*h

    t_values[i+1] = t_values[i] + h
  return t_values, S_y_values, I_y_values, R_y_values
```

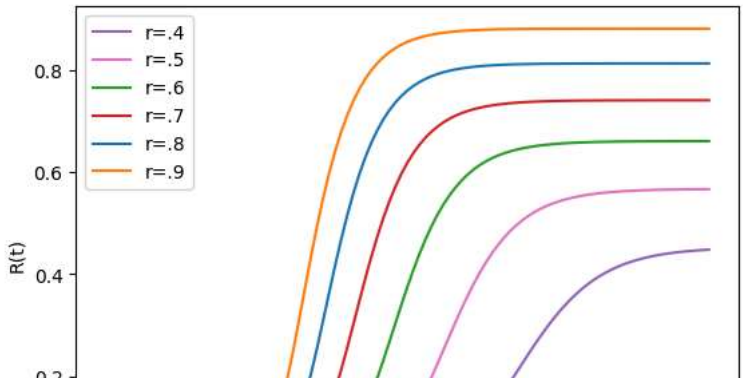
```
a = 0
b = 10
alpha = 0  #intial condition
h = .05
N = int((b - a)/h)

t_values = np.zeros(N+1)

#r = .4

r1_t, r1_S, r1_I, r1_R = runge_kutta(.4)
r2_t, r2_S, r2_I, r2_R = runge_kutta(.5)
r3_t, r3_S, r3_I, r3_R = runge_kutta(.6)
r4_t, r4_S, r4_I, r4_R = runge_kutta(.7)
r5_t, r5_S, r5_I, r5_R = runge_kutta(.8)
r6_t, r6_S, r6_I, r6_R = runge_kutta(.9)


fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(r1_t, r1_S, color='tab:purple', label='r=.4')
ax.plot(r2_t, r2_S, color='tab:pink', label='r=.5')
ax.plot(r3_t, r3_S, color='tab:green', label='r=.6')
ax.plot(r4_t, r4_S, color='tab:red', label='r=.7')
ax.plot(r5_t, r5_S, color='tab:blue', label='r=.8')
ax.plot(r6_t, r6_S, color='tab:orange', label='r=.9')

ax.set_ylabel("S(t)")
ax.set_xlabel("t")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x78ea5f9c5e40>
```



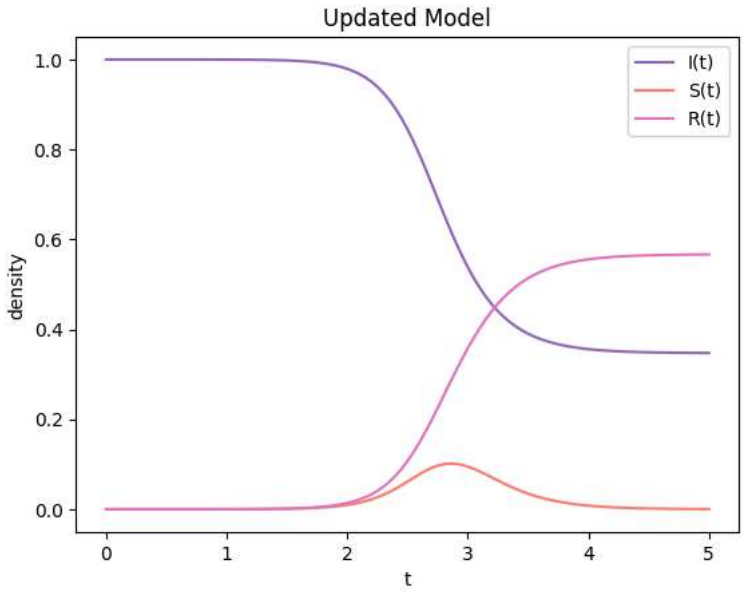

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(r1_t, r1_R, color='tab:purple', label='r=.4')
ax.plot(r2_t, r2_R, color='tab:pink', label='r=.5')
ax.plot(r3_t, r3_R, color='tab:green', label='r=.6')
ax.plot(r4_t, r4_R, color='tab:red', label='r=.7')
ax.plot(r5_t, r5_R, color='tab:blue', label='r=.8')
ax.plot(r6_t, r6_R, color='tab:orange', label='r=.9')

ax.set_ylabel("R(t)")
ax.set_xlabel("t")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x78ea5f86d870>
```




```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(r1_t, r2_I, color='tab:purple', label='I(t)')
ax.plot(r1_t, r2_S, color='salmon', label='S(t)')
ax.plot(r1_t, r2_R, color='tab:pink', label='R(t)')

ax.set_title("Updated Model")
ax.set_xlabel("t")
ax.set_ylabel("density")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x78ea5f7f9570>
```




```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(r6_t, r6_I, color='tab:purple', label='I(t)')
ax.plot(r6_t, r6_S, color='salmon', label='S(t)')
ax.plot(r6_t, r6_R, color='tab:pink', label='R(t)')

ax.set_title("Updated Model")
ax.set_xlabel("t")
ax.set_ylabel("density")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x78ea5f778e20>
```