Salgın Hastalık Modelleme

Hepimizin mağlumu olan ve hepimiz etkileyen bu Covid-19 salgını bir çok sayısal veri ile karşımıza çıkıyor. Bir çok uzman tıp adamını dinliyor ve evlerimizde kalmaya özen gösteriyoruz.

Bende evde kalan biri olarak kendi çalışma alanımda kullandığım yöntemlerin kullanımına dair kısa bir döküman hazırladım. Salgın hastalıkların matematiksel modellemesi bilim dünyasında yaygın ve popüler bir konu.

Sars, Ebola, Influenza gibi salgınların toplumda yayılışının matematiksel modellemesi konusunda bir çok çalışma mevcut. Bu modellemelerde salgınların nasıl kontrol altına alınacağı, algın hızının yayılmaya etkisi gibi birçok başlık irdeleniyor. Bende bu fizksel sistemlerin simülasyonu ve kaos teorisi alanıda çalışan biri olarak bu alanda çalışmak isteyenlere, python programlama dili kullanarak nasıl simülasyonlar geliştiebileceklerine yönelik öncü olabilecke bir döküman hazırlamaya çalıştım.

Bu dökümanda salgın modellerinden olan SIR modelini anlatmaya çalışacağım ve bir python kodu vasıtasıyla bir kaç simülasyon durumuna bakacağız. Hadi başlayalım:

SIR Modeli

SIR modeli, temel olarak edipemik modelleme türüdür. Bir çok versiyonu mevcut birçok akademik makaleye konu olmuş durumdadır. Ebola salgınından, Influenza salgınına kadar birçok salgın probleminin matematiksel modellemesinde kullanılan bir yöntemdir. Temel olarak üç temel değişkene(zamana bağlı) dayanır:

- S(t)=Suspected-Şüpheli popülasyon
- I(t)=Infected-(Enfeksiyonu almış popülasyon)
- R(t)=Recovered-Hastalıktan arınmış popülasyon

Bu üç değişkene ilaveten ik adet serbest parametre de kullanılır:

- v(infection rate)(enfeksiyon yüzdesi)
- σ-recovery rate(iyileşme yüzdesi)

SIR modeli bir ordinary-diffrential equation(ODE) sistemi aslında ve temel denklem sistemi şudur:

$$\frac{dS(t)}{dt} = -\frac{vS(t)I(t)}{N}$$
$$\frac{dI(t)}{dt} = \frac{vS(t)I(t)}{N} - \sigma I(t)$$
$$\frac{dR(t)}{dt} = \sigma I(t)$$

Bu ODE içinde tabiki kısıtlarımız mevcut:

- $S(0) = S_0 \geq 0$
- $I(0) = I_0 \geq 0$
- $R(0) = R_0 \geq 0$

Bu başlangıç koşullarında sınırlamalrı dikkate alarak modelimizin nümerik simülasyonunu yapabiliriz. AM burada dikkat etmemiz gereken bir koşulumuzdaha mevcut buda toplam poülasyonun sabit olması durumu yani:

$$S + I + R = N$$

N popülasyonu temsil ediyor.

BU durumda poülasyon sayısı daima değişken olduğuna göre analize yapmadan önce bizim sistem değişkenlerinin ölçeğini ayarlamamız çok daha akıllıca olacaktır ki bu sayede sayılardan çok oranların değişimine bakabiliriz. Ölçekleme şu şekilde olacak:

$$\frac{S(t)}{N} = s(t)$$
$$\frac{I(t)}{N} = i(t)$$
$$\frac{R(t)}{N} = r(t)$$

bu sayede artık toplam popülasyon önkoşulumuzu şöyle yazabiliriz:

$$s(t) + i(t) + r(t) = 1$$

bu da bizim SIR modelimizin N bağımlılığından kurtarır ve son SIR modelinin son hali şu hali alır:

$$\frac{ds(t)}{dt} = -vs(t)i(t)$$
$$\frac{di(t)}{dt} = vs(t)i(t) - \sigma i(t)$$
$$\frac{dr(t)}{dt} = \sigma i(t)$$

Artık bu matematiksel modelimiz üzerinden epidemik hastalıkların yayılmasını zamana bağlı olarak simülasyonunu yapabiliriz. Burada bizim serbest parametrelerimiz ve başlangıç durumlarımız bizim çeşitli seneryoları kontrol etmemize olanak sağlıyor. Şimdi bu işin programlama ve nümerik kısmına göz atalım.

Python ile ODE Entegrasyonu

Bizim ODE'miz üzerinden artık numerik değerlendirme yapabiliriz. ODE üzerinden yapacağımız imülasyon aslında bir IVP(initial value problem) yani başlangıç-değer problemi. Bu diferaniyel sistemini zaman içinde entegre ederek bizim sistem değişkenlerimizin değişimini gözlemleyeceğiz. Aşağıda SIR() modelimiz python fonksiyonu olarak tanımlanmış durumda. İlk olrak bir seneryo örneği yapalım ve enfeksiyon oranı=0.2 ve iyileşme oranını=0.1 olarak seçiyoruz.

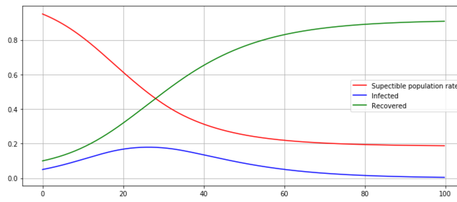
başlangıç koşulları olarak ta (s(0),i(0),r(0))=(0.95,0.05,0) yani toplumun yüzde 95'i şüpheli, yüzde 5 i hasta olam durumunu simüle ediyoruz.Zamanda t=0 dan t=300 e kadar 900 adım için simülasyonu çalıştırıyoruz:

In [27]:
```
from scipy import linspace
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
%matplotlib inline
    #v=enfeksiyon oran(infection rate)
    #sigma=iyilieşme oranı(recovery rate)
v= 0.2
sigma=0.1
def SIR(t, z):
    s, i, r = z
    return [-v*s*i, v*s*i-sigma*i,sigma*i]

a, b = 0, 300
t = linspace(a, b, 900)
plt.figure(figsize=(12,5))

sol1 = solve_ivp(SIR, [a, b], [0.95, 0.05,0.1], t_eval=t)
plt.grid(True)
sus,=plt.plot(sol1.t[:300], sol1.y[0][:300], 'r', label="Supectible population rate") # plot
ting t, suspectible
inf,=plt.plot(sol1.t[:300], sol1.y[1][:300], 'b', label="Infected") # plotting t, b separate
ly
rec,=plt.plot(sol1.t[:300], sol1.y[2][:300], 'g', label="Recovered") # plotting t, c separat
ely
plt.legend(handles=[sus, inf,rec])
```

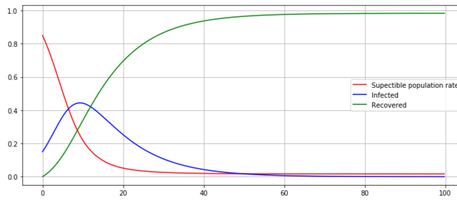
Out[27]:
```
<matplotlib.legend.Legend at 0x6f5a9b47f0>
```



İkinci bir senaryo yapalım ve enfeksiyon oranınını biraz daha arttıralım ve toplumdaki enfeksiyonlu insan yüzdesini de arttıralım

In [34]:
```
v=0.4
plt.figure(figsize=(12,5))
sol2 = solve_ivp(SIR, [a, b], [0.85, 0.15,0.0], t_eval=t)
plt.grid(True)
sus,=plt.plot(sol1.t[:300], sol2.y[0][:300], 'r', label="Supectible population rate") # plot
ting t, suspectible
inf,=plt.plot(sol1.t[:300], sol2.y[1][:300], 'b', label="Infected") # plotting t, b separate
ly
rec,=plt.plot(sol1.t[:300], sol2.y[2][:300], 'g', label="Recovered") # plotting t, c separat
ely
plt.legend(handles=[sus, inf,rec])
```

Out[34]:
```
<matplotlib.legend.Legend at 0x6f5ae14ba8>
```



Bu iki seneryo gördüğünüz üzere farklılıklar mevcut. Lakin bizim sevindirecek durumlarda gözlemleyebiliriz. Bu basit modelde dahil zaman içinde iyileşenlerin miktarı artıyor lakin yayılma hızı arttıkça gördüğünüz üzere hatalanma miktarı da bir anda artabiliyor.

Son bir örnek olarak ta hastalığın yayılma oranını iyice iyileşme oranına göre arttırıp bu çalışmayı şimdilik sonlandıralım.

In [35]:
```
v=0.8
plt.figure(figsize=(12,5))
sol2 = solve_ivp(SIR, [a, b], [0.70, 0.30,0.0], t_eval=t)
plt.grid(True)
sus,=plt.plot(sol1.t[:300], sol2.y[0][:300], 'r', label="Supectible population rate") # plot
ting t, suspectible
inf,=plt.plot(sol1.t[:300], sol2.y[1][:300], 'b', label="Infected") # plotting t, b separate
ly
rec,=plt.plot(sol1.t[:300], sol2.y[2][:300], 'g', label="Recovered") # plotting t, c separat
ely
plt.legend(handles=[sus, inf,rec])
```

Out[35]:
```
<matplotlib.legend.Legend at 0x6f5be36438>
```