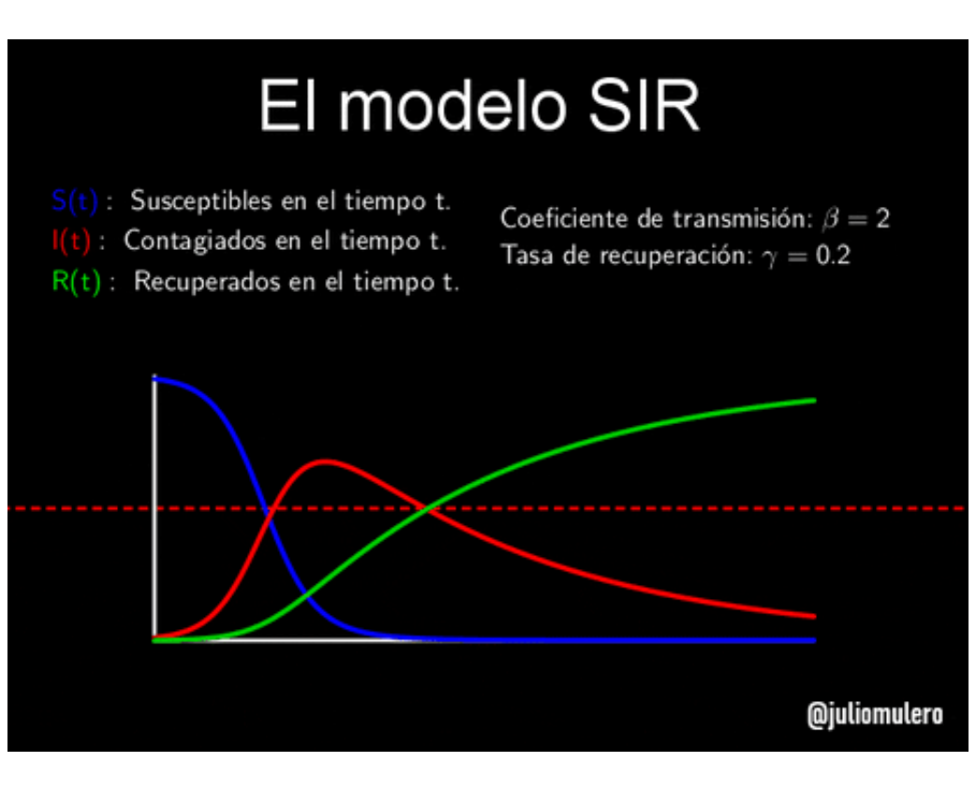
Desarrollo de un modelo SIR para la población de Ecuador

Autor: Bryam David Vega Moreno
Maestro: Diego Quisi
Materia: Simulación
Universidad: Universidad Politécnica Salesiana
Carrera: Ciencias de la computación

Introducción

El modelo SIR es un modelo matemático que permite predecir el comportamiento de una enfermedad infecciosa, a partir de ciertas condiciones iniciales



Este modelo clasifica una población en tres grupos distintos:

- **Suceptible:** Número de personas que son propensas a la enfermedad
- **Infectado:** Número de personas que tienen la enfermedad y que pueden infectar a la gente susceptible
- **Recuperado:** Númeto de persoans que no pueden contraer la enfermedad, porque se han recuperado completamente, o porque son inmunes

Ademas de estos grupos, este modelo depende principalmente de tres parámetros:

- **Tasa de transmisión (β):** Describe que tan rápido se transmite la infección de un individuo a otro
- **Tasa de recuperación (γ):** Describe qué tan rápido un individuo infectado se recupera.
- **Población (N):** Número de población del país.

Para poder resolver este modelo debemos hacer uso de las siguientes fórmulas:

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\beta S \frac{I}{N},$$
$$\frac{\mathrm{d}I}{\mathrm{d}t} = \beta S \frac{I}{N} - \gamma I,$$
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I.$$

Para poder realizar un modelo básico SIR, utilizaremos la libreria SciPy el cual contiene un conjunto de paquetes que permiten resolver sistemas de ecuaciones, integrarlas y graficarlas. A continuación mostramos el código

---

Librerias a importar

Para el análisis de datos

```
In [1]: import pandas as pd
        import numpy as np
        from datetime import timedelta, datetime
```

Librerias para visualización de datos

```
In [2]: import altair as alt
        import matplotlib.pyplot as plt
```

Librerias para resolución de ecuaciones diferenciales

```
In [3]: import scipy.integrate as spi
        from scipy.optimize import minimize
        from scipy.integrate import solve_ivp
```

Desarrollo del modelo básico SIR

Para empezar a desarrollar el modelo debemos tener en cuenta que necesitaremos crear las variables para nuestro numero de población, número de infectados y número de recuperados, así como también nuestra tasa de transmisión y de recuperación, para ello ponemos los siguientes puntos:

- **N:** Número de población
- **I0:** Número inicial de infectados
- **R0:** Número inicial de recuperados
- **S0:** Número inicial de personas recuperadas que no es mas que N-I0-R0
- **beta:** Tasa de transmisión
- **gamma:** Tasa de recuperación teniendo en cuenta que una persona puede recuperarse en 15 días, es decir 1/15

```
In [4]: N     = 16000000
        I0    = 10000
        R0    = 0
        S0    = N - I0 - R0
        beta  = 0.3
        gamma = 1/15
        t     = np.linspace(0, 365, 365)
```

Definido nuestras variables, procedemos a realizar las fórmulas para poder resolver nuestro problema de ecuaciones diferenciales, para ello creamos un método denominado **deriv** que contiene las formulas expuestas en la introducción

```
In [5]: def deriv(y, t, N, beta, gamma):
            S, I, R = y
            dSdt = -beta * S * I / N
            dIdt = beta * S * I / N - gamma * I
            dRdt = gamma * I
            return dSdt, dIdt, dRdt
```

Después de haber realizado las funciones para resolver nuestra ecuación diferencial y tener los valores iniciales, procedemo a resolver el módelo con las condicionales iniciales, con ello vamos a obtener los valores para 365 como estaba inicializado nuestra variable **t**

```
In [6]: y0 = S0, I0, R0
        ret = spi.odeint(deriv, y0, t, args=(N, beta, gamma))
        S, I, R = ret.T
```

Procedemos a crear un pequeño dataframe que nos explica como va variado el SIR a medida que avanza el tiempo, con la finalidad de ver mejor los datos, cabe recalcar que dichos estan normalizados de 0 a 1 ya que estan divididos para el número de población

```
In [7]: df = pd.DataFrame({'time':t.astype('int'),'susceptible':S/N,'infected':I/N,'recovered':R/N})
        df.head(5)
```

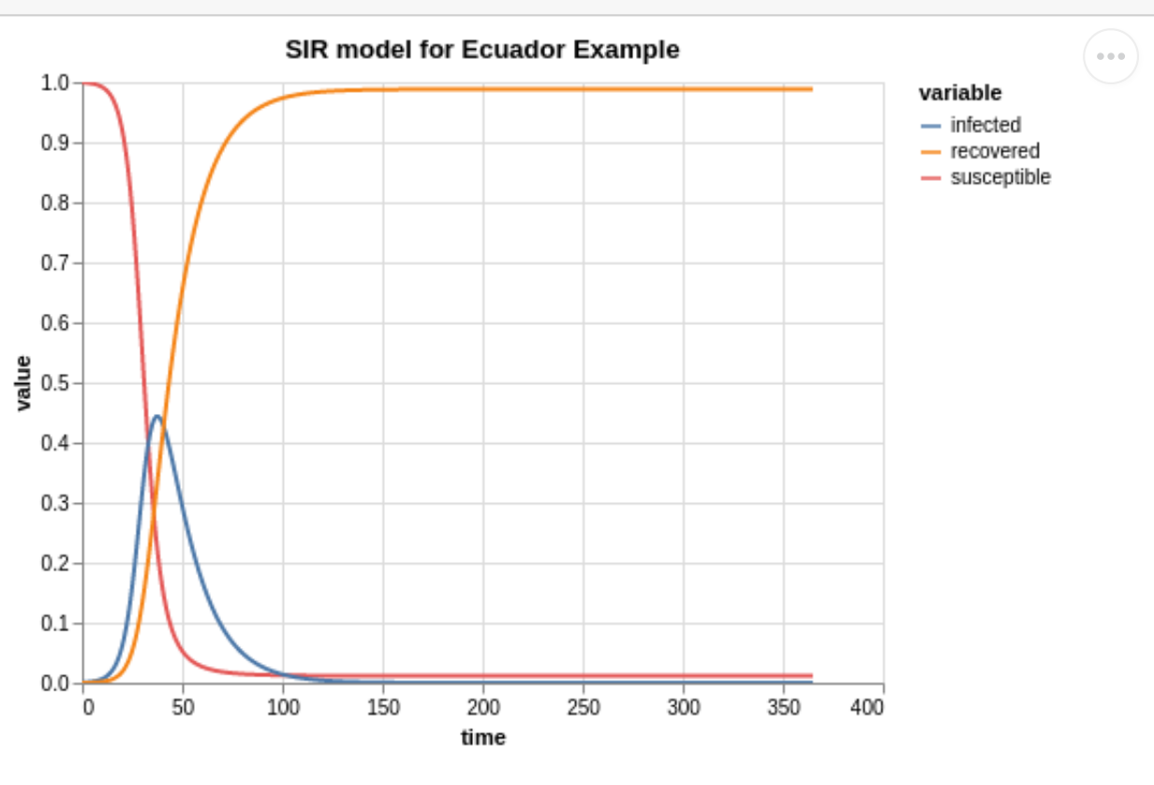
Out[7]:

	time	susceptible	infected	recovered
0	0	0.999375	0.000625	0.000000
1	1	0.999163	0.000790	0.000047
2	2	0.998896	0.000997	0.000107
3	3	0.998558	0.001260	0.000182
4	4	0.998132	0.001591	0.000277

Por último graficamos los datos utilizando la libreria **altair** la cual nos permite hacer la gráfica un poco más interáctive y con un mejor diseño. Con esta gráfica podemos darnos cuenta que a medida que se aumenta el número de infectados, el número de recuperados también aumenta por lo que por consiguiente, el número de personas susceptibles también tiende a bajar a medida que aumentan los recuperados y bajan las infecciones.

```
In [8]: alt.Chart(df.melt('time')).mark_line().encode(
            x='time',
            y=alt.Y('value'),
            color='variable'
        ).properties(
            title='SIR model for Ecuador Example'
        ).interactive()
```

Out[8]:



Ahora, cuando podemos estimar beta y gamma, hay varios conocimientos derivados de el. Si D, es el promedio de días para recuperarse de infecciones , se deriba de gamma.

$$D = \frac{I}{\gamma}$$

Ademas de ello, podemos estimar la naturaleza de la enfermadad en terminos del poder de la infección.

$$R_0 = \frac{\beta}{\gamma}$$

Se llama número de reproducción básico **R0**. Es el número promedio de personas infectadas de otra persona. Si es alto, la probabilidad de pandemia también es mayor. Si bien antes teniamos como variables fijas beta y gamma, ahora lo que procederemos a hacer es estimar β y γ para ajustar el modelo SIR a los casos confirmados reales (el número de personas infecciosas). A continuación vamos a proceder con el código y la explicación

---

Simulación del modelo SIR para COVID-19

Lo primero que procedemos a hacer es crear nuestras variables I0,R0,S0 con la finaliad de tener datos iniciales, estos datos pueden ir variando para ir simulando los diferentes escenarios. Para los valores tomados, tendremos en cuenta que existiran 10 millones suceptibles, mientras que habran 10 infectados y 0 recuperados. De la misma manera tomaremos 365 días a partir del día del primer caso. A continuación mostramos las variables

```
In [9]: I0=10
        R0=0
        S0 = 100000
        t = 365
        y0 = S0,I0,R0
```

Con las variables y condiciones iniciales procedemos a leer los datos que necesitamos para poder realizar nuestro modelo SIR, en este caso utilizamos los dataset de numero de confirmados y numero de recuperados con la finalidad de que se pueda obtener una mejor curva para los recovered puesto que después de un conjunto de experimentos, existia un problema en el número de recuperados. El código de lectura y obtención de datos es el siguiente:

```
In [10]: def filter_country(df,country,start_date):
             country_df = df[df['Country/Region'] == country]
             return country_df.iloc[0].loc[start_date:]

         def load_data(path_confirmed,path_recovered,country,date):
             df_confirmed = filter_country(pd.read_csv(path_confirmed),country,date)
             df_recovered = filter_country(pd.read_csv(path_recovered),country,date)
             return df_confirmed,df_recovered
```

```
In [11]: data_confirmed,data_recovered=load_data('./in/time_series_covid19_confirmed_global.csv','./in/time_series_covid19_re
         covered_global.csv','Ecuador','3/1/20')
```

Ahora procedemos a realizar una función con el fin de optimizar los valores de beta y gamma con el fin de mejorar dichos valores con respecto a la data que tenemos, para ello utilizamos dos funciones para a la final ver sus diferencias, en la primera función **loss_confirmed_recovered** utilizamos la data de personas infectadas y recuperadas a fin de mejorar la curva de recovered, mientras que en la otra función **loss_confirmed** solamente utilizamos la data de infectados. A continuación mostramos la función que nos permite optimizar dicho proceso.

NOTA

A partir de este punto vamos a optener dos soluciones, debido a que tenemos diferentes funciones de optimización y tomamos en cuenta diferentes combinaciones de tipos de datos, con esto en cuenta vamos a referirnos a dichas funciones con la siguiente nomenclatura a patir de aqui:

- **loss_confirmed_recovered:** primera función
- **loss_confirmed:** segunda función

```
In [12]: def loss_confirmed_recovered(point, data, recovered):
             size = len(data)
             beta, gamma = point
             def SIR(t, y):
                 S = y[0]
                 I = y[1]
                 R = y[2]
                 return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
             solution = solve_ivp(SIR, [0, size], [S0,I0,R0], t_eval=np.arange(0, size, 1), vectorized=True)
             l1 = np.sqrt(np.mean((solution.y[1] - data)**2))
             l2 = np.sqrt(np.mean((solution.y[2] - recovered)**2))
             alpha = 0.1
             return alpha * l1 + (1 - alpha) * l2
```

```
In [13]: def loss_confirmed(point, data):
             size = len(data)
             beta, gamma = point
             def SIR(t, y):
                 S = y[0]
                 I = y[1]
                 R = y[2]
                 return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
             solution = solve_ivp(SIR, [0, size], [S0,I0,R0], t_eval=np.arange(0, size, 1), vectorized=True)
             return np.sqrt(np.mean((solution.y[1] - data)**2))
```

En el siguiente código procedemos a realizar la optimización para ambos casos explicados anteriormente

```
In [15]: optimal_cr = minimize(
                     loss_confirmed_recovered,
                     [0.001, 0.001],
                     args=(data_confirmed,data_recovered),
                     method='L-BFGS-B',
                     bounds=[(0.00000001, 0.4), (0.00000001, 0.4)])

         optimal_c = minimize(
                     loss_confirmed,
                     [0.001, 0.001],
                     args=(data_confirmed),
                     method='L-BFGS-B',
                     bounds=[(0.00000001, 0.4), (0.00000001, 0.4)])
```

```
In [16]: beta_cr,gamma_cr = optimal_cr.x
         beta_c,gamma_c = optimal_c.x
```

Despues de optimizar dichos valores, procedemos a motrar los valores que obtuvimos con la primera función

```
In [17]: print("valor gamma: {}".format(gamma_cr))
         print("valor beta: {}".format(beta_cr))

         valor gamma: 0.021909766044310948
         valor beta: 1.2249257546538603e-06
```

Mientras que por otro lado, estos son los valores que obtuvimos con la segunda función

```
In [18]: print("valor gamma: {}".format(gamma_c))
         print("valor beta: {}".format(beta_c))

         valor gamma: 4.574930844836162e-06
         valor beta: 4.574930844836162e-06
```

Con dichos valores procedemos a obtener R_0 con el objetivo de conocer mas que nada su valor. En este caso mostrarmos el R_0 con los valores de beta y gamma de la primera función

```
In [19]: R_0= beta_cr/gamma_cr
         print("Número de reproducción R_0: {}".format(R_0))

         Número de reproducción R_0: 5.5907751464645256e-05
```

Mientras que por otro lado mostramos el valor de R_0 con los valores de beta y gamma de la segunda función

```
In [20]: R_0= beta_c/gamma_c
         print("Número de reproducción R_0: {}".format(R_0))

         Número de reproducción R_0: 1.0
```

Una vez obtenido el valor de R_0 el cual es un indicador para ver el número de reproducción del virus, procedemos a realizar la resolución del modelo SIR, no necesitamos de R_0 en este momento, simplemente es un indicador que nos permite saber como se reproduce el virus en nuestra ciudad. Ahora con ello en mente mostramos el código en donde resolvemos el modelo SIR

```
In [21]: def extend_index(index, new_size):
                 values = index.values
                 current = datetime.strptime(index[-1], '%m/%d/%y')
                 while len(values) < new_size:
                     current = current + timedelta(days=1)
                     values = np.append(values, datetime.strftime(current, '%m/%d/%y'))
                 return values

         def predict(beta, gamma, data):
                 predict_range = t
                 new_index = extend_index(data.index, predict_range)
                 size = len(new_index)
                 def SIR(t, y):
                     S = y[0]
                     I = y[1]
                     R = y[2]
                     return [-beta*S*I, beta*S*I-gamma*I, gamma*I]
                 extended_actual = np.concatenate((data.values, [None] * (size - len(data.values))))
                 return new_index, extended_actual, solve_ivp(SIR, [0, size], [S0,I0,R0], t_eval=np.arange(0, size, 1))
```

Procedemos a realizar las predicciones para nuestros datos, teniendo en cuenta el beta y gamma. A continuación realizamos dichas predicciones para la primera y segunda función

```
In [25]: new_index, extended_actual, prediction_cr = predict(beta_cr, gamma_cr, data_confirmed)
         new_index, extended_actual, prediction_c = predict(beta_c, gamma_c, data_confirmed)
```

Después de realizar las predicciones para el modelo SIR procedemos a armar un dataframe para poder visualizar nuestro modelo terminado. A la final tendremos dos datasets ya que el uno es de la primera función, mientras que el otro es de la segunda función

```
In [26]: df_cr = pd.DataFrame(
                     {'date':[i for i in range(0,len(new_index))],
                     'suceptible': prediction_cr.y[0],
                     'infected': prediction_cr.y[1],
                     'recovered': prediction_cr.y[2]})
         df_c = pd.DataFrame(
                     {'date':[i for i in range(0,len(new_index))],
                     'suceptible': prediction_c.y[0],
                     'infected': prediction_c.y[1],
                     'recovered': prediction_c.y[2]})
```

```
In [34]: df_cr.head(5)
```

Out[34]:

	date	suceptible	infected	recovered
0	0	100000.000000	10.000000	0.000000
1	1	99998.711348	11.058154	0.230498
2	2	99997.286259	12.228338	0.485403
3	3	99995.710512	13.522228	0.767260
4	4	99993.968185	14.952895	1.078920

```
In [35]: df_c.head(5)
```

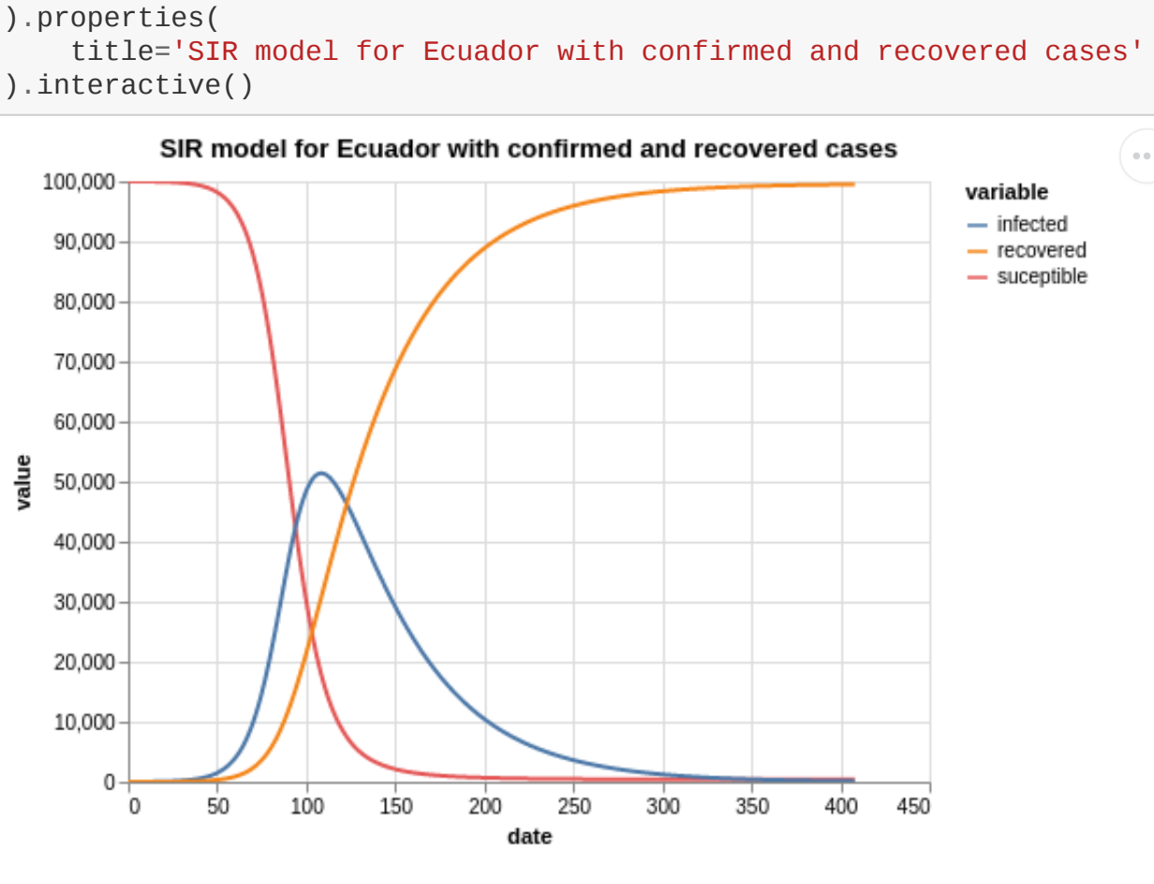
Out[35]:

	date	suceptible	infected	recovered
0	0	100000.000000	10.000000	0.000000
1	1	99994.199144	15.800798	0.000058
2	2	99985.031025	24.968825	0.000150
3	3	99970.567197	39.432509	0.000294
4	4	99947.686515	62.312961	0.000523

Con este conjunto de datos procedemos a graficarlos, para poder ver la diferencia que existe al usar la primera función y la segunda función

```
In [33]: alt.Chart(df_cr.melt('date')).mark_line().encode(
             x='date',
             y=alt.Y('value'),
             color='variable'
         ).properties(
             title='SIR model for Ecuador with confirmed and recovered cases'
         ).interactive()
```

Out[33]:



```
In [32]: alt.Chart(df_c.melt('date')).mark_line().encode(
             x='date',
             y=alt.Y('value'),
             color='variable'
         ).properties(
             title='SIR model for Ecuador only confirmed cases'
         ).interactive()
```

Out[32]:

Como podemos apreciar, con el uso de la primera función existe una mejor visualización de las personas recuperadas, es decir, es muy importante tomar en cuenta dicho valor para que exista una curva correcta con respecto a los casos recuperados. Sin embargo, como podemos ver en la gráfica de la segunda función en donde solamente se tomo en cuenta el número de personas confrmadas, podemos notar claramente como no existe número de personas récuperadas y al no existir recuperadas por ende dichas personas van a seguir infectadas, por lo cual esto es erroneo ya que si existen personas recuperadas. Por lo tanto la mejor manera de simular el modelo SIR, es con la función de optimización **loss_confirmed_recovered** ya que toma en cuenta a las personas infectadas y recuperadas y de esa manera podemos ver que al existir recuperados, el número de infectados bajan lo cual tiene lógica.

Conclusiones

Como podemos ver, el modelo SIR es un modelo que nos permite ver el grado de infección de un virus y ver como este transcurre a lo largo del tiempo, de la misma manera nos permite de cierta manera predecir, el número de contagios, como también el número de contagiados y recuperados dando una cierta tendencia con respecto a los datos pasados. Hemos notado que los valores de beta y gamma los cuales corresponden a las carácterísticas principales del modelo SIR, son valores muy importantes para poder aplanar la curva de infectados, de la misma manera obtuvimos el R_0 el cual nos va a permitir ver el número de reproducción del virus.

Recursos

- **COVID-19 dynamics with SIR model:** https://www.lewuathe.com/covid-19-dynamics-with-sir-model.html