Modelo SIR datos Covid-19 Mexico

December 10, 2020

1 Modelo SIR COVID-19 México

1.0.1 Corina Cerezo

[1]:
```
import pandas as pd
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import PDEparams as pde
from datetime import datetime, timedelta, date
```

2 Introducción

En este trabajo se aplica el método Monte Carlo para estimar los parámetros del modelo dinámico SIR con la base de datos de Covid-19 México (hasta la fecha 08 de dic de 2020). Posteriormente, una vez conocidos los parámetros, se grafica el modelo en comparación con los datos.

2.1 Modelo SIR

El modelo SIR es un modelo matemático que nos permite predecir el comportamiento de una enfermedad infecciosa, a partir de ciertas condiciones iniciales. Este modelo clasifica a una población en tres grupos distintos:

- **S**: Susceptibles. Número de personas que son propensas a la enfermedad (que aún no se han enfermado).
- **I**: Infectados. Número de personas que tienen la enfermedad y que pueden infectar a la gente susceptible.
- **R**: Recuperado. Número de personas que no pueden contraer la enfermedad, porque se han recuperado completamente, o porque son inmunes o porque fallecieron.

Además este modelo usa principalmente dos parámetros:

- β: Tasa de transmisión. Describe qué tan rápido se transmite la infección de un individuo a otro (probabilidad de que una persona se enferme).
- γ: Tasa de recuperación. Describe qué tan rápido un individuo se recupera.

Ahora bien, la razón de cambio con respecto al tiempo t de la cantidad de personas de los tres grupos ya mencionados está dada por:

$$\frac{\partial S}{\partial t} = -\beta \cdot S \cdot \frac{I}{N}$$
$$\frac{\partial I}{\partial t} = \beta \cdot S \cdot \frac{I}{N} - \gamma \cdot I$$
$$\frac{\partial R}{\partial t} = \gamma \cdot I$$

Estas tres forman un sistema de ecuaciones diferenciales que permiten modelar enfermedades infecciosas.

2.2 Datos

Esta base de datos corresponde al informe de pacientes con Covid-19 hasta el día 08 de diciembre de 2020.

[2]:
```
datos = pd.read_csv('201208COVID19MEXICO.csv', encoding = "ISO-8859-1")
```

[3]:
```
datos.head()
```

[3]:

	FECHA_ACTUALIZACION	ID_REGISTRO	ORIGEN	SECTOR	ENTIDAD_UM	SEXO	\
0	2020-12-08	1468a5	1	4	5	1	
1	2020-12-08	043f64	2	4	9	2	
2	2020-12-08	0e07d8	1	4	15	2	
3	2020-12-08	13757c	1	12	15	1	
4	2020-12-08	002371	1	4	3	2	

	ENTIDAD_NAC	ENTIDAD_RES	MUNICIPIO_RES	TIPO_PACIENTE	...	OTRO_CASO	\
0	15	5	18	1	...	1	
1	9	9	10	2	...	99	
2	15	15	104	2	...	99	
3	15	15	106	1	...	1	
4	3	3	8	1	...	99	

	TOMA_MUESTRA_LAB	RESULTADO_LAB	TOMA_MUESTRA_ANTIGENO	RESULTADO_ANTIGENO	\
0	1	1	2	97	
1	1	1	2	97	
2	1	1	2	97	
3	1	1	2	97	
4	1	1	2	97	

	CLASIFICACION_FINAL	MIGRANTE	PAIS_NACIONALIDAD	PAIS_ORIGEN	UCI
0	3	99	MÃ©xico	97	97
1	3	99	MÃ©xico	97	1
2	3	99	MÃ©xico	97	2
3	3	99	MÃ©xico	97	97
4	3	99	MÃ©xico	97	97

[5 rows x 40 columns]

Se toman solo las variables de interes que son ID_REGISTRO, FECHA_SINTOMAS, FECHA_DEF y RESULTADO

[4]:
```
datos = datos[['ID_REGISTRO', 'FECHA_SINTOMAS', 'FECHA_DEF',␣
 ↪'CLASIFICACION_FINAL']]
datos = datos.rename(columns={"CLASIFICACION_FINAL": "RESULTADO"},␣
 ↪errors="raise")
datos.head(15)
```

[4]:

	ID_REGISTRO	FECHA_SINTOMAS	FECHA_DEF	RESULTADO
0	1468a5	2020-03-27	9999-99-99	3
1	043f64	2020-03-26	2020-03-30	3
2	0e07d8	2020-03-28	2020-04-02	3
3	13757c	2020-03-27	9999-99-99	3
4	002371	2020-03-27	9999-99-99	3
5	11fb00	2020-03-25	2020-04-05	3
6	092521	2020-03-28	9999-99-99	3
7	0955a5	2020-03-28	9999-99-99	3
8	1a1f12	2020-03-27	2020-03-31	3
9	12d93f	2020-03-20	9999-99-99	3
10	197355	2020-03-02	9999-99-99	3
11	0b3ad1	2020-03-15	9999-99-99	3
12	03f951	2020-03-20	9999-99-99	3
13	185d7d	2020-03-11	9999-99-99	3
14	0e7ba2	2020-03-26	9999-99-99	3

[5]:
```
# Notemos a ver si existe gente rara
aux = datos[(datos['RESULTADO'] <= 3) & (datos['FECHA_DEF'] != "9999-99-99")]
aux['FECHA_SINTOMAS'] = pd.to_datetime(aux['FECHA_SINTOMAS'], format="%Y/%m/%d")
aux['FECHA_DEF'] = pd.to_datetime(aux['FECHA_DEF'], format="%Y/%m/%d")
raros = aux[aux['FECHA_SINTOMAS'] > aux['FECHA_DEF']]
raros.head(20)
```

[5]:

	ID_REGISTRO	FECHA_SINTOMAS	FECHA_DEF	RESULTADO
1920860	2e7963	2020-08-27	2020-07-16	2

[6]:
```
#esta persona se contagió de covid después de muerta jajajajaja
#como eso esta raro la quitamos
datos = datos.drop([1920860], axis = 0)
```

[7]:
```
#Se cambia el tipo de dato de FECHA_SINTOMAS
datos['FECHA_SINTOMAS'] = pd.to_datetime(datos['FECHA_SINTOMAS'], format="%Y/%m/
 ↪%d")
datos.head(5)
```

[7]:

	ID_REGISTRO	FECHA_SINTOMAS	FECHA_DEF	RESULTADO
0	1468a5	2020-03-27	9999-99-99	3

```
1       043f64      2020-03-26  2020-03-30          3
2       0e07d8      2020-03-28  2020-04-02          3
3       13757c      2020-03-27  9999-99-99          3
4       002371      2020-03-27  9999-99-99          3
```

Se filtran los pacientes positivos a COVID-19 (con RESULTADO igual a 1, 2 o 3), podemos ver que para esa fecha son 1193254 contagiados.

[8]:
```
positivos = datos[datos['RESULTADO'] <= 3]
print(positivos.info())
#positivos.head(20)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1193254 entries, 0 to 3053711
Data columns (total 4 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID_REGISTRO     1193254 non-null  object
 1   FECHA_SINTOMAS  1193254 non-null  datetime64[ns]
 2   FECHA_DEF       1193254 non-null  object
 3   RESULTADO       1193254 non-null  int64
dtypes: datetime64[ns](1), int64(1), object(2)
memory usage: 45.5+ MB
None
```

Para conocer los casos nuevos por día se suman los positivos por FECHA_SINTOMAS.

[9]:
```
aux = positivos["FECHA_SINTOMAS"].value_counts()
infectados = pd.DataFrame(aux)
infectados.columns = ['infectados']
#infectados.tail(20)
print(infectados.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 296 entries, 2020-11-20 to 2020-02-23
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   infectados  296 non-null    int64
dtypes: int64(1)
memory usage: 4.6 KB
None
```

[10]:
```
#Solo para verificar que los infectados suman los casos positivos
infectados['infectados'].sum()
```

[10]:
```
1193254
```

Para contar los sobrevivientes por día se filtran todos los casos positivos (RESULTADO "1") sin

fecha de defunción (FECHA_DEF valor "99-99-9999"), pues serían los contagiados que no han muerto y se agrupan por la fecha que empezaron con los síntomas (podemos ver que también se tiene 110 días).

[11]:
```
aux = datos[(datos['RESULTADO'] <= 3) & (datos['FECHA_DEF'] == '9999-99-99')]
aux = aux["FECHA_SINTOMAS"].value_counts()
sobrevivientes = pd.DataFrame(aux)
sobrevivientes.columns = ['sobre']
sobrevivientes.head(20)
print(sobrevivientes.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 295 entries, 2020-11-20 to 2020-02-23
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   sobre   295 non-null    int64
dtypes: int64(1)
memory usage: 4.6 KB
None
```

[12]:
```
#Son 86689 personas contagiadas hasta la fecha de la base de datos que siguen␣
 ↪con vida
sobrevivientes['sobre'].sum()
```

[12]: 1082381

Para contar los muertos por día se filtran todos los casos positivos (RESULTADO 1, 2 o 3) con fecha de defunción (FECHA_DEF distinto a "99-99-9999") registrados en la base de datos y se agrupan por la fecha de defunción.

[13]:
```
aux = positivos[(datos['RESULTADO'] <= 3) & (datos['FECHA_DEF'] != '9999-99-99')]
aux['FECHA_DEF'] = pd.to_datetime(aux['FECHA_DEF'], format = "%Y/%m/%d")
aux = aux["FECHA_DEF"].value_counts()
muertes = pd.DataFrame(aux)
muertes.columns = ['muertes']
#muertes.head(20)
print(muertes.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 267 entries, 2020-07-06 to 2020-12-08
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   muertes  267 non-null    int64
dtypes: int64(1)
memory usage: 4.2 KB
None
```

Podemos ver que solo hay 267 días, es decir, que ha habido días en los que la gente no muere.

[14]:
```
#Acumulado de muertes hasta la fecha de la base de datos
muertes['muertes'].sum()
```

[14]:
```
110873
```

Vamos a construir un DataFrame. Para esto juntaremos en una misma base de datos, por día, los casos nuevos de infección, los casos sobrevivientes (recorridos 14 días que implica que la gente ya sanó) y los casos de muertes.

[15]:
```
#creamos una base de datos que tenga todos los días desde 23/feb/2020 hasta 20/
 ↪nov/2020
date_index = pd.date_range('2020/02/23', periods=272, freq='D')
aux0 = pd.DataFrame(index=date_index)
#Se unen los infectados
aux1 = pd.merge(left=aux0,right=infectados, how='outer', left_index=True,␣
 ↪right_index=True)
aux1 = aux1.fillna(0)
#Se unen los muertos
aux2 = pd.merge(left=aux1,right=muertes, how='outer', left_index=True,␣
 ↪right_index=True)
aux2 = aux2.fillna(0)
#Para los sobrevivientes quitamos las fechas de index y se hacen una columna más
aux3 = sobrevivientes.reset_index()
aux3.drop('index', axis = 1, inplace=False)
#Se suman 14 días a la fecha en que la persona se enfermo,
#dado que no murió entonces 14 días despúes se sanó
aux3['recorrido'] = aux3['index']+ timedelta(days=14)
#Se fija la fecha recorrida como index para poder unirla a la otra base de datos
aux3 = aux3.set_index(['recorrido'])
#Unimos la gente que sanó a la base de datos
ims = pd.merge(left=aux2,right=aux3, how='left', left_index=True,␣
 ↪right_index=True)
ims = ims.fillna(0)
ims = ims.drop(['index'], axis='columns')
ims.tail(20)
#print(ims.info())
#len(ims)
```

[15]:

	infectados	muertes	sobre
2020-11-19	6234	368.0	5454.0
2020-11-20	9882	435.0	5498.0
2020-11-21	6095	433.0	5059.0
2020-11-22	6262	473.0	4934.0
2020-11-23	7895	431.0	6674.0
2020-11-24	6631	431.0	7008.0

```
2020-11-25        7037    416.0  5495.0
2020-11-26        5919    453.0  5029.0
2020-11-27        6240    479.0  5001.0
2020-11-28        6185    422.0  4961.0
2020-11-29        5105    433.0  6249.0
2020-11-30        6766    429.0  6095.0
2020-12-01        5963    449.0  6382.0
2020-12-02        3590    391.0  6321.0
2020-12-03        2790    389.0  5952.0
2020-12-04        2020    291.0  9454.0
2020-12-05        1255    319.0  5865.0
2020-12-06         422    242.0  6021.0
2020-12-07         837     64.0  7628.0
2020-12-08           2      1.0  6437.0
```

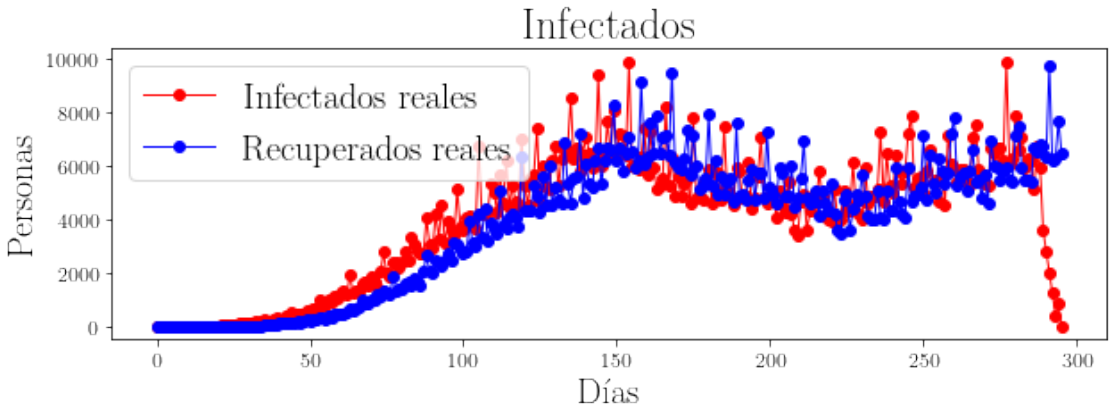
!Ahora sí! Se crea la base de datos a trabajar el modelo SIR

[16]:
```
#definimos la base de datos a trabajar
N=125000000
SIR_datos = {'t' : np.linspace(0, len(ims)-1, len(ims)),
#             'S' : N - ims.infectados - (ims.muertes + ims.sobre),
             'S' : N - ims.infectados.cumsum(),
             'I' : ims.infectados,
             'R' : ims.muertes + ims.sobre}
SIR_datos = pd.DataFrame(SIR_datos)
#print(SIR_datos.info())
SIR_datos.iloc[0,1] = N - SIR_datos.iloc[0,2] - SIR_datos.iloc[0,3]
SIR_datos.tail()
```

[16]:
```
                t           S     I       R
2020-12-04  291.0  123809262.0  2020  9745.0
2020-12-05  292.0  123808007.0  1255  6184.0
2020-12-06  293.0  123807585.0   422  6263.0
2020-12-07  294.0  123806748.0   837  7692.0
2020-12-08  295.0  123806746.0     2  6438.0
```

Y las gráficas de infectados y recuperados son:

[17]:
```
# Gráfica de los infectados
t = np.linspace(0, len(ims)-1, len(ims))
fig = plt.figure(figsize=(10,3))
plt.plot(t, SIR_datos.I, '-ok', color='r', label='Infectados reales', lw = 1)
plt.plot(t, SIR_datos.R, '-ok', color = 'b', label='Recuperados reales', lw = 1)
plt.xlabel('Días')
plt.ylabel('Personas')
plt.title('Infectados')
plt.legend()
plt.show()
```



2.3 Monte carlo para los parámetros β y γ

[18]:
```
def deriv(z, t, b, g):
    S, I, R = z
    N = 125000000
    dSdt = -b * S * I / N
    dIdt = b * S * I / N - g * I
    dRdt = g * I
    return [dSdt, dIdt, dRdt]
```

[19]:
```
# Valores iniciales
# Población total
N = 125000000
t = np.linspace(0, len(ims)-1, len(ims))
# Infectados al tiempo cero
def I0():
    return 1
# Recuperados al tiempo cero
def R0():
    return 0
# Susceptibles al tiempo 0
def S0():
    return N - 1
```

Para encontrar los mejores parámetros se corrieron muchos modelos "jugando" con la cota de los parámetros. Es importante mencionar que dado que γ representa una tasa de recuperación, y puesto que por el contexto de la enfermedad no sucede que el día en el que te enfermas, mueres, entonces γ solo puede tomar valores entre 0 y 1. Aunque es posible que otros valores mejoren el ajuste, por diseño del modelo ni tendría sentido.

[20]:
```
%%time
evaluaciones = []
```

```
for b in range(1,5):
    for s in range(2,30):
        g = 1/s
        my_model = pde.PDEmodel(SIR_datos, deriv, [S0, I0, R0], bounds=[(0,b),␣
 ↪(g,1)],
                        param_names=[r'$beta$', r'$gamma$'], nvars=3, ndims=0,␣
 ↪nreplicates=1, obsidx=None, outfunc=None)
        my_model.fit()
        evaluaciones.append([my_model.best_params.iloc[0,0],my_model.best_params.
 ↪iloc[0,1],my_model.best_error, b, g])
#print(evaluaciones)
```

```
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  0.978348  0.942217
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  0.121435  0.076923
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  0.111528  0.066667
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  0.103966  0.058824
   $beta$   $gamma$
0     1.0  0.963913
```

```
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  0.101201  0.055953
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  0.101203  0.055955
    $beta$   $gamma$
0  0.10118  0.055931
     $beta$  $gamma$
0  0.101188  0.05594
     $beta$   $gamma$
0  0.101177  0.055929
     $beta$   $gamma$
0  0.101368  0.056127
     $beta$   $gamma$
0  0.101187  0.055938
     $beta$   $gamma$
0  0.101177  0.055928
   $beta$   $gamma$
0     1.0  0.963913
     $beta$   $gamma$
0  1.032034  0.996056
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
    $beta$   $gamma$
0  1.00537  0.969028
     $beta$   $gamma$
0  0.220756  0.178636
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.154598  0.111111
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.134976  0.090909
     $beta$   $gamma$
0  0.127637  0.083333
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  1.030061  0.994078
```

```
     $beta$   $gamma$
0  0.111528  0.066667
    $beta$  $gamma$
0  0.10751   0.0625
     $beta$   $gamma$
0  0.103966  0.058824
    $beta$   $gamma$
0  1.03571  0.999745
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.101185  0.055937
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  1.031813  0.995834
     $beta$  $gamma$
0  0.101189  0.05594
     $beta$   $gamma$
0  1.034034  0.998064
    $beta$   $gamma$
0  0.10118  0.055931
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  1.021804  0.985791
     $beta$   $gamma$
0  0.101187  0.055938
     $beta$   $gamma$
0  0.101175  0.055926
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
    $beta$   $gamma$
0  1.03561  0.999645
     $beta$   $gamma$
0  1.019561  0.983542
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  1.031792  0.995814
```

```
     $beta$   $gamma$
0  0.134976  0.090909
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  1.033256  0.997283
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.111528  0.066667
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.101176  0.055927
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  0.101179  0.05593
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.101177  0.055928
     $beta$   $gamma$
0  0.101184  0.055935
     $beta$   $gamma$
0  0.101176  0.055928
     $beta$  $gamma$
0  0.101179  0.05593
     $beta$   $gamma$
0  1.034477  0.998507
     $beta$   $gamma$
0  0.373133  0.333333
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
```

```
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  0.168127    0.125
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.134976  0.090909
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  1.031985  0.996007
     $beta$  $gamma$
0  1.035964      1.0
     $beta$  $gamma$
0  1.035964      1.0
    $beta$  $gamma$
0  0.10751   0.0625
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.101175  0.055926
     $beta$   $gamma$
0  1.035906  0.999941
     $beta$   $gamma$
0  0.101175  0.055927
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.101902  0.056681
     $beta$   $gamma$
0  0.101175  0.055926
     $beta$   $gamma$
0  0.101174  0.055925
     $beta$   $gamma$
0  0.101577  0.056344
     $beta$   $gamma$
0  0.101579  0.056345
    $beta$   $gamma$
0  0.10118  0.055932
     $beta$  $gamma$
0  1.035964      1.0
     $beta$   $gamma$
0  0.101176  0.055927
Wall time: 3min 39s
```

[21]:
```
#Esta función sirve para ordenar
def Sort(sub_li):
    sub_li.sort(key = lambda x: x[2])
    return sub_li
```

[22]:
```
#Ordenamos las evaluaciones
eval_ord = Sort(evaluaciones)
#print(eval_ord)
```

Entonces si la variable β está entre 0 y 1, y la variable γ está entre 0.03448 y 1, entonces los parámetros que alcanzan el menor error son:

[23]:
```
%%time
my_model = pde.PDEmodel(SIR_datos, deriv, [S0, I0, R0], bounds=[(0,1), (1/29,1)],
                        param_names=[r'$beta$', r'$gamma$'], nvars=3, ndims=0,␣
 ↪nreplicates=1, obsidx=None, outfunc=None)
my_model.fit()
```

```
   $beta$   $gamma$
0     1.0  0.963913
Wall time: 1.62 s
```

[24]:
```
print(my_model.best_error)
```

```
83044615951.9893
```

[25]:
```
my_model.best_params
```

[25]:
```
   $beta$   $gamma$
0     1.0  0.963913
```

2.3.1 Likelihood Profile

[26]:
```
%%time
my_model.likelihood_profiles()
```

```
HBox(children=(FloatProgress(value=0.0, description='parameters', max=2.0, style=ProgressStyle(d

HBox(children=(FloatProgress(value=0.0, description='values within parameters', style=ProgressSt



HBox(children=(FloatProgress(value=0.0, description='values within parameters', style=ProgressSt



Wall time: 2min 30s
```

[27]:
```
my_model.result_profiles
```

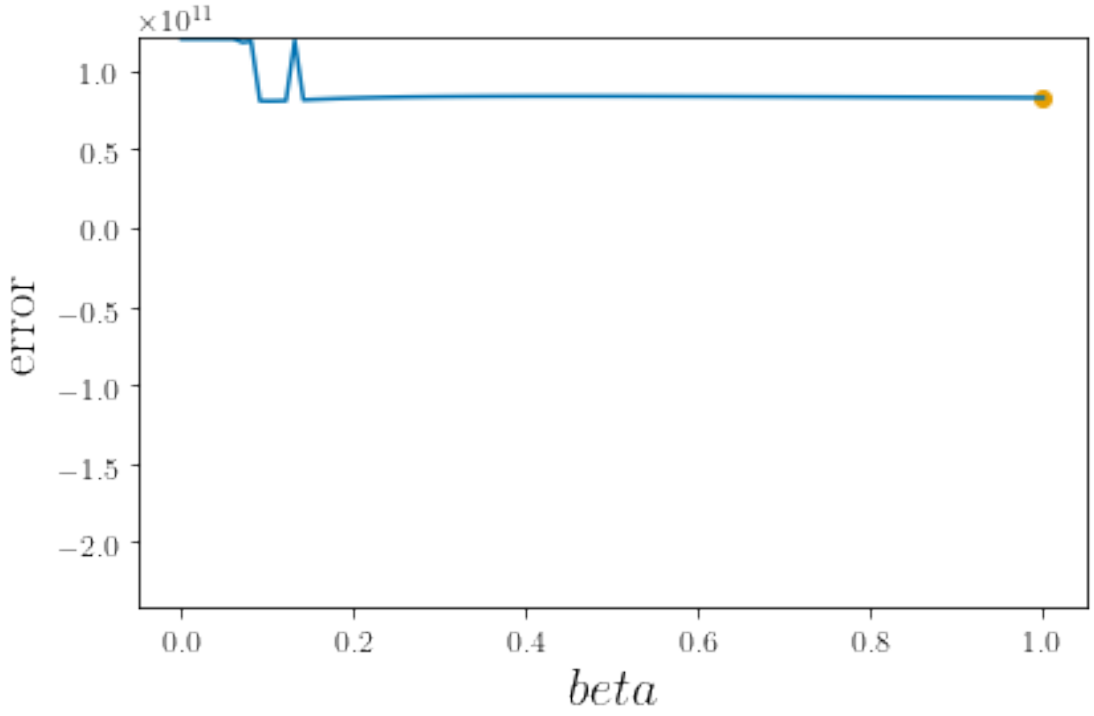
[27]:

	parameter	value	error
0	$beta$	0.000000	1.206454e+11
1	$beta$	0.010101	1.206453e+11
2	$beta$	0.020202	1.206452e+11
3	$beta$	0.030303	1.206451e+11
4	$beta$	0.040404	1.206441e+11
..	...	...	...
195	$gamma$	0.960989	1.202612e+11
196	$gamma$	0.970742	1.205715e+11
197	$gamma$	0.980495	1.193609e+11
198	$gamma$	0.990247	1.205799e+11
199	$gamma$	1.000000	1.206070e+11

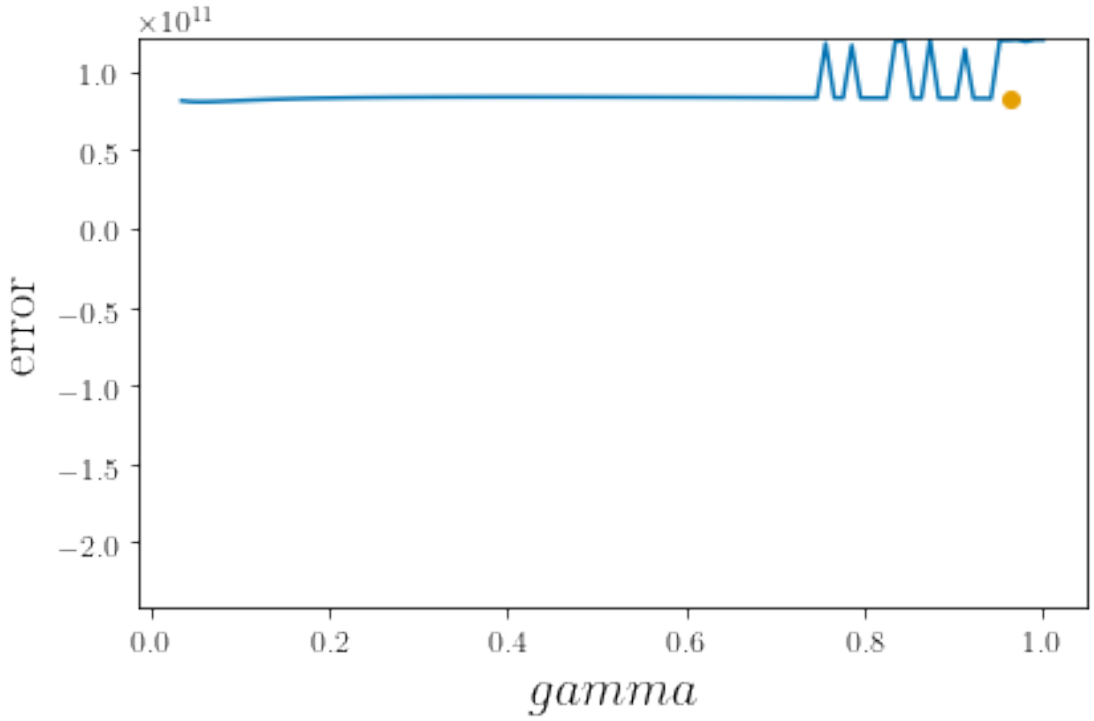
[200 rows x 3 columns]

[28]:
```
my_model.plot_profiles()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```



```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```

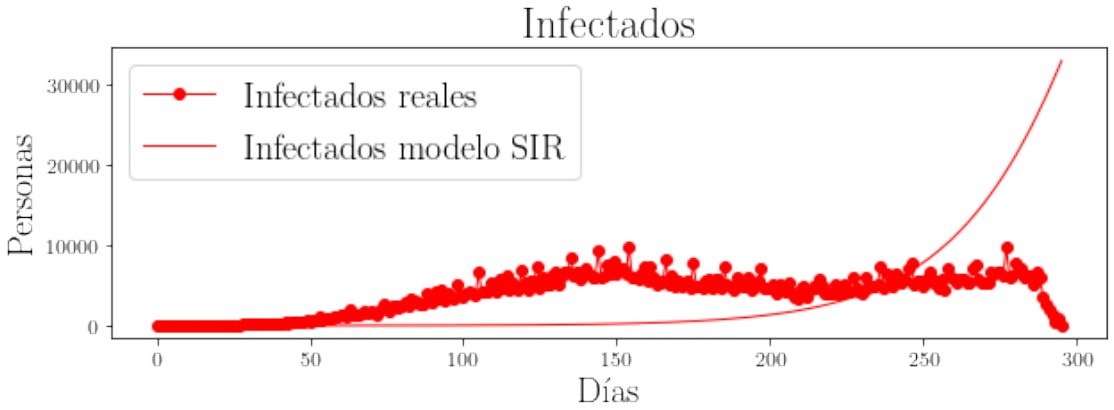


2.4 Gráficas del modelo SIR

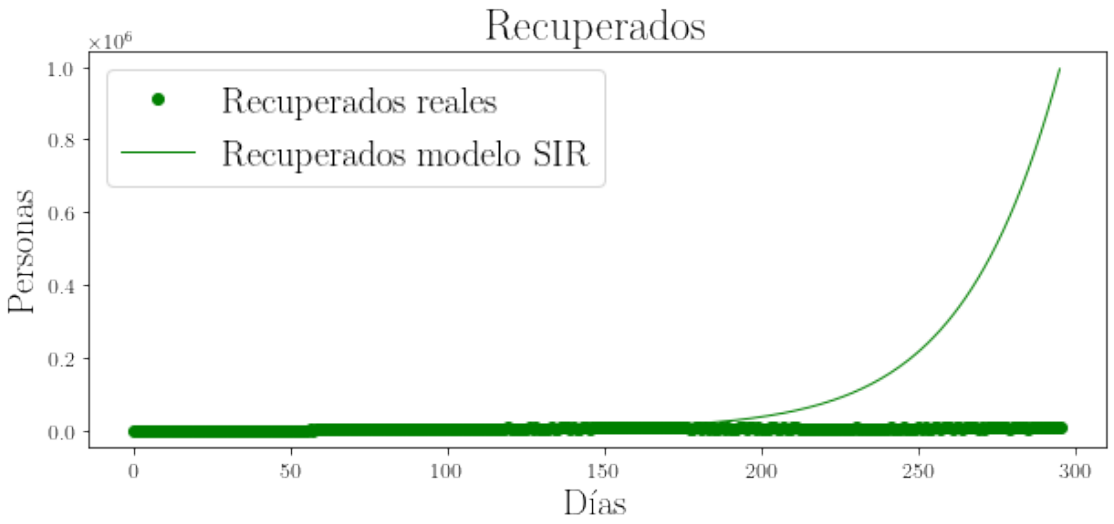
[29]:
```
#Extraemos los valores del mejor modelo
b = my_model.best_params.iloc[0,0]
g = my_model.best_params.iloc[0,1]
#Recordemos los valores iniciales
N = 125000000
I0 = 1
R0 = 0
S0 = N - I0 - R0
z0 = S0, I0, R0
t = np.linspace(0, len(ims)-1, len(ims))
```

[30]:
```
# Integrate the SIR equations over the time grid, t.
ret = odeint(deriv, z0, t, args=(b, g))
S, I, R = ret.T
```

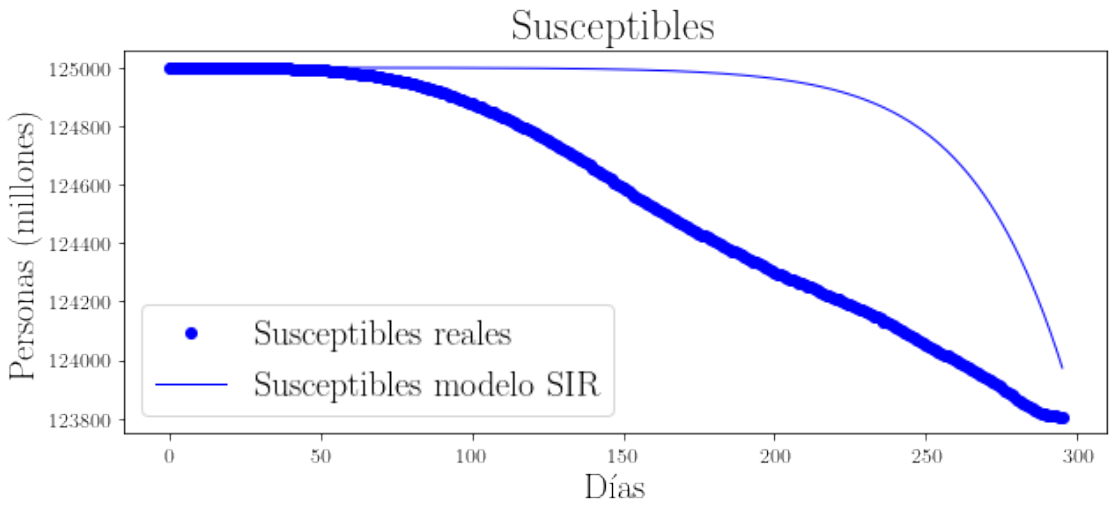
[31]:
```
# Gráfica de los infectados
fig = plt.figure(figsize=(10,3))
plt.plot(t, SIR_datos.I, '-ok', color='r', label='Infectados reales', lw = 1)
plt.plot(t, I, color = 'r', label='Infectados modelo SIR', lw = 1)
plt.xlabel('Días')
plt.ylabel('Personas')
plt.title('Infectados')
plt.legend()
plt.show()
```



[32]:
```
# Gráfica de los Recuperados
fig = plt.figure(figsize=(10,4))
plt.plot(t, SIR_datos.R, 'o', color='g', label='Recuperados reales', lw = 1)
plt.plot(t, R, color = 'g', label='Recuperados modelo SIR', lw = 1)
plt.xlabel('Días')
plt.ylabel('Personas')
plt.title('Recuperados')
plt.legend()
plt.show()
```



[33]:
```
# Gráfica de los Susceptibles
fig = plt.figure(figsize=(10,4))
plt.plot(t, SIR_datos.S/1000, 'o', color='b', label='Susceptibles reales', lw =␣
↪1)
plt.plot(t, S/1000, color = 'b', label='Susceptibles modelo SIR', lw = 1)
plt.xlabel('Días')
plt.ylabel('Personas (millones)')
plt.title('Susceptibles')
plt.legend()
plt.show()
```



2.5 Referencias:

https://datos.covid-19.conacyt.mx/#DownZCSV

https://www.inegi.org.mx/contenidos/saladeprensa/aproposito/2020/Poblacion2020_Nal.pdf

https://python.quantecon.org/sir_model.html

https://www.mscbs.gob.es/biblioPublic/publicaciones/recursos_propios/resp/revista_cdrom/VOL94/C_ESP

https://www.medigraphic.com/pdfs/forense/mmf-2020/mmf203c.pdf