Proyecto 2 de IA

Determinación de parámetros de sistemas de ecuaciones diferenciales ordinarias.

Objetivo

Con la ayuda de algunos datos referentes a la pandemia provocada por el Sars-Cov2, determinar parámetros del modelo SIR con la ayuda del código proporcionado por el profesor.

In [1]:
```
#Importamos paquetería necesaria
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from pandas import DataFrame
from scipy.integrate import odeint
```

In [2]:
```
#Código del profesor
import PDEparams as pde
```

Definición del modelo SIR

El modelo SIR está dado por las siguientes ecuaciones:

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -b\,S\,I \tag{1}$$

$$\frac{\mathrm{d}I}{\mathrm{d}t} = b\,S\,I - c\,I \tag{2}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = c\,I, \tag{3}$$

donde S es la variable que representa a las personas suceptibles, I a las personas infectadas y R representa a las personas recuperadas. Aquí, ya estamos considerando a las cantidades S, I y R como normalizadas (divididas entre la problación total). Por otro lado, tanto b como c son

constantentes, donde b representa la rapidez de propagación de la enfermedad, mientras que c representa la rapidez de recuperación.

In [3]:
```
#Ecuación del modelo SIR
def SIR(z, t, b, c):

    S,I,R= z

    return [-b*S*I,b*S*I-c*I ,c*I ]
```

Datos utilizados

Los datos que se van a utilizar son los referentes a la pandemia en Chile, obtenidos de https://www.gob.cl/coronavirus/cifrasoficiales/#datos.

Infectados por día

Para la determinación de los parámetros de este ejercicio, tomaremos en consideración los datos reportados de infectados por día.

In [4]:
```
#Cargar datos
#Aquí estamos considerando los infectados por día, se guardan en la variable dfchpd.
dfchpd = pd.read_csv("dataperday.csv")
dfchpd["Día"]=dfchpd["Región"]
dfchpd["Infectados por día"]=dfchpd["Total"]
dfchpd=dfchpd.drop("Región", axis=1)
dfchpd=dfchpd.drop("Total",axis=1)
```

In [5]:
```
#Visualizamos el data frame
dfchpd.head()
```

Out[5]:

	Día	Infectados por día
0	03-Mar	0.0
1	04-Mar	2.0
2	05-Mar	1.0
3	06-Mar	1.0
4	07-Mar	2.0

In [6]:
```
dfchpd.tail()
```

Out[6]:

	Día	Infectados por día
303	31-Dic	3022.0
304	01-Ene	3588.0
305	02-Ene	3338.0
306	03-Ene	2289.0
307	04-Ene	2450.0

Cabe mencionar que aunque el gobierno informó sobre la detección del primer contagio el 3 de Marzo, no se contabilizó en esta base de datos

In [7]:
```
#Quitamos el primer renglón dado que en la tabla se informan que no hay infectados.
dfchpd=dfchpd.drop(0,axis=0)
dfchpd.head()
```

Out[7]:

	Día	Infectados por día
1	04-Mar	2.0
2	05-Mar	1.0
3	06-Mar	1.0
4	07-Mar	2.0
5	08-Mar	3.0

In [8]:
```
#Revisar si hay datos faltantes
dfchpd.loc[dfchpd["Infectados por día"].isnull()]
```

Out[8]:

	Día	Infectados por día

Ahora, normalicemos a los infectados por día. Según los datos del banco mundial(https://datos.bancomundial.org/indicator/SP.POP.TOTL?locations=CL), en el último censo llevado a cabo en Chile, se contaron 18 958 038 pobladores. Así, dividiremos a los infectados por día entre este valor.

In [9]:
```
#N=población total en Chile
```

```
N=18958038
dfchpd["Infectados por día"]=dfchpd["Infectados por día"]/N
dfchpd.head()
```

Out[9]:

	Día	Infectados por día
1	04-Mar	1.054961e-07
2	05-Mar	5.274807e-08
3	06-Mar	5.274807e-08
4	07-Mar	1.054961e-07
5	08-Mar	1.582442e-07

En un inicio, solamente hay dos personas infectadas que no se han recuperado. Adicionalmente SIR considera a todas las demás personas no infectadas y no recuperadas como suceptibles. Entonces la condición inicial para este caso es

$$(S(0), I(0), R(0)) = (\frac{N-2}{N}, \frac{2}{N}, 0).$$

In [10]:
```
#Cargamos la condición inicial
def cond_ini_S():
    return (N-2)/N
def cond_ini_I():
    return 2/N
def cond_ini_R():
    return 0
```

Generamos dos dataframe con el tiempo normalizado, uno para aplicarle los algoritmos llamado dft y otro para, posteriormente, graficar los datos llamado dftgraph.

In [11]:
```
dfchpd=dfchpd.reset_index()
del dfchpd['index']
dfchpd["t"]=pd.Series([i for i in range(0,308)])/306
```

In [12]:
```
dft=dfchpd.drop("Día",axis=1)
dft=dfchpd.reset_index()
dft=dft[["t","Infectados por día"]]
dftgraph=dft
```

In [13]:
```
dft=dft.drop(index=0)
dft=dft.reset_index()
del dft['index']
dft.head()
```

Out[13]:

	t	Infectados por día
0	0.003268	5.274807e-08
1	0.006536	5.274807e-08
2	0.009804	1.054961e-07
3	0.013072	1.582442e-07
4	0.016340	2.637404e-07

A continuación se eligen las cotas de los rangos utilizados en el algoritmo, dependiendo del error generado. Tomemos en cuenta que $c \in (0, 1]$. A la cota superior para b la llamaremos k y a la cota superior para c la llamaremos j.

In [14]:
```
y=[]
for k in np.linspace(0,10,100):
    for j in np.linspace(0,0.1,10):
        modelo1 = pde.PDEmodel(dft, SIR , [cond_ini_S, cond_ini_I, cond_ini_R], bounds=[(0,k), (0,j)],
                               param_names=[r'$b$', r'$c$'], nvars=3, ndims=0 , nreplicates=1 ,obsidx=[1], outfunc=No
        modelo1.fit()
        y.append((modelo1.best_error,k,j))
```

```
   $b$  $c$
0  0.0  0.0
   $b$       $c$
0  0.0  0.000169
   $b$       $c$
0  0.0  0.000674
   $b$       $c$
0  0.0  0.000633
   $b$       $c$
0  0.0  0.000051
   $b$      $c$
0  0.0  0.00098
   $b$       $c$
0  0.0  0.000113
```

```
   $b$       $c$
0  0.0  0.001745
   $b$       $c$
0  0.0  0.000804
   $b$       $c$
0  0.0  0.000646
        $b$  $c$
0  0.100574  0.0
        $b$       $c$
0  0.099902  0.002483
        $b$      $c$
0  0.098675  0.00401
        $b$       $c$
0  0.100073  0.006077
        $b$       $c$
0  0.100403  0.004207
        $b$       $c$
0  0.097842  0.004207
       $b$       $c$
0  0.09666  0.007655
        $b$       $c$
0  0.100078  0.015308
      $b$       $c$
0  0.0962  0.008802
        $b$       $c$
0  0.095832  0.002825
        $b$  $c$
0  0.201267  0.0
        $b$       $c$
0  0.200847  0.000595
       $b$       $c$
0  0.20175  0.000204
       $b$       $c$
0  0.19872  0.001249
        $b$       $c$
0  0.193334  0.013626
        $b$       $c$
0  0.190049  0.006432
       $b$       $c$
0  0.19637  0.002571
        $b$       $c$
0  0.194648  0.013067
        $b$      $c$
0  0.198899  0.01795
        $b$       $c$
```

```
0  0.195275  0.018573
       $b$  $c$
0  0.302866  0.0
       $b$       $c$
0  0.298821  0.006853
       $b$       $c$
0  0.298516  0.003496
       $b$       $c$
0  0.302124  0.013216
       $b$       $c$
0  0.297003  0.023461
       $b$       $c$
0  0.290273  0.009077
       $b$      $c$
0  0.300935  0.01132
       $b$       $c$
0  0.300174  0.008044
       $b$       $c$
0  0.287938  0.013756
       $b$       $c$
0  0.296745  0.004998
       $b$  $c$
0  0.393928  0.0
       $b$       $c$
0  0.399815  0.001921
       $b$      $c$
0  0.399259  0.00387
       $b$       $c$
0  0.397145  0.001906
       $b$       $c$
0  0.398691  0.014857
       $b$       $c$
0  0.388688  0.010806
       $b$       $c$
0  0.396078  0.015666
       $b$       $c$
0  0.401212  0.043053
       $b$       $c$
0  0.403758  0.005096
      $b$       $c$
0  0.39493  0.027664
      $b$  $c$
0  0.50419  0.0
       $b$       $c$
0  0.498895  0.001298
```

```
        $b$       $c$
0  7.422076  0.036149
        $b$      $c$
0  7.424166  0.02569
        $b$       $c$
0  7.475839  0.081401
```

In [15]:
```
#h es la serie con los datos referentes a la elección de k y j.
y=pd.DataFrame(y,columns=["Error","k","j"])
y=y.sort_values(by=["Error"], ascending=True)
h=y.iloc[0]
h
```

Out[15]:
```
Error    1.305829e-08
k        8.989899e+00
j        1.000000e-01
Name: 899, dtype: float64
```

Ahora, aplicamos el algoritmo con los datos de k y j encontrados y lo guardamos en modelo1.

In [16]:
```
modelo1 = pde.PDEmodel(dft, SIR , [cond_ini_S, cond_ini_I, cond_ini_R], bounds=[(0,h.loc["k"]), (0,h.loc["j"])],
                       param_names=[r'$b$', r'$c$'], nvars=3, ndims=0 , nreplicates=1 ,obsidx=[1], outfunc=None)
modelo1.fit()
```

```
        $b$       $c$
0  7.431626  0.034832
```

In [17]:
```
#Valores encontrados de b y c
modelo1.best_params
```

Out[17]:

	b	c
0	7.431626	0.034832

In [18]:
```
#Error generado
modelo1.best_error
```

Out[18]:
```
1.3058293063463893e-08
```

In [19]:
```
%%time
modelo1.likelihood_profiles()
```

```
Wall time: 36.8 s
```

In [20]:
```
modelo1.result_profiles
```

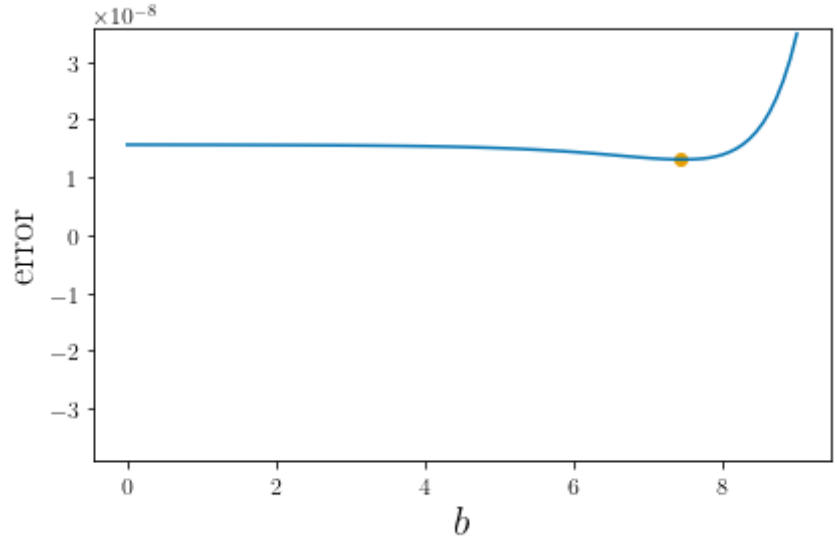
Out[20]:

	parameter	value	error
0	b	0.000000	1.559412e-08
1	b	0.090807	1.559311e-08
2	b	0.181614	1.559200e-08
3	b	0.272421	1.559086e-08
4	b	0.363228	1.558962e-08
...	...	...	...
195	c	0.095960	1.306061e-08
196	c	0.096970	1.305835e-08
197	c	0.097980	1.305850e-08
198	c	0.098990	1.305831e-08
199	c	0.100000	1.305829e-08

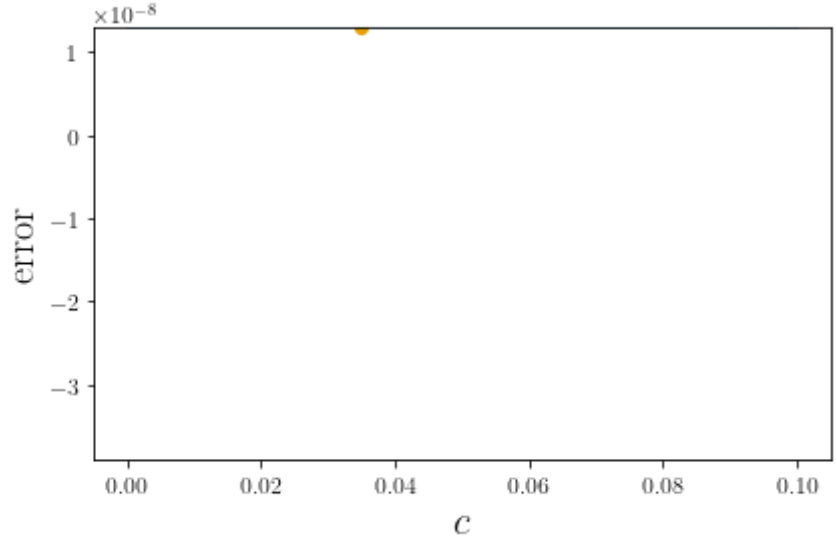
200 rows × 3 columns

In [21]:
```
modelo1.plot_profiles()
```

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

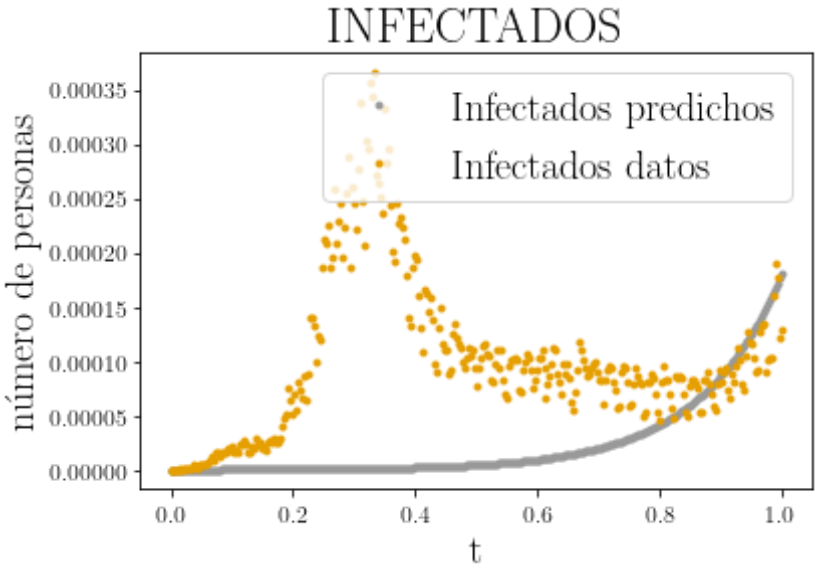


Con los valores obtenidos de b y c, resolvamos numéricamente el sistema SIR y comparemos esa solución con los datos observados.

In [22]:
```
infec_datos=np.array(list(dftgraph["Infectados por día"]))
```

In [23]:
```
b,c = modelo1.best_params.loc[0,"$b$"],modelo1.best_params.loc[0,"$c$"]

def dP_dt(P, t):
    return [-b*P[0]*P[1],b*P[0]*P[1]-c*P[1] ,c*P[1]]

ts = np.linspace(0, 1, 307)
P0 = [(N-2)/N,2/N,0]
Solucion = odeint(dP_dt, P0, ts)
Suceptibles = Solucion[:,0]
Infectados = Solucion[:,1]
Recuperados = Solucion[:,2]
```

In [24]:
```
plt.plot(ts, Infectados, ".", label="Infectados predichos")
plt.plot(ts,infec_datos, ".", label="Infectados datos")
plt.xlabel("t")
plt.ylabel("número de personas")
plt.title("INFECTADOS")
plt.legend();
```



Parámetros utilizando la estimación de activos y recuperados del gobierno chileno.

El gobierno Chileno tiene sus propias estimaciones de personas que pueden transmitir el virus (activos) por día y de recuperados acumulados (la estimación está descrita en la página de internet proporcionada anteriormente). Solamente como ejercicio, haremos el mismo procedimiento, solamente que utilizando dichas estimaciones.

Estimación utilizando solamente los datos de los Activos.

In [25]:
```
#Cargamos datos
dfar=pd.read_csv("dataactivos.csv")
dfar.head(40)
```

Out[25]:

	Unnamed: 0	Activos	Recuperados
0	22-Feb	1.000	0.000
1	23-Feb	2.000	0.000
2	24-Feb	2.000	0.000
3	25-Feb	9.000	0.000
4	26-Feb	10.000	0.000
5	27-Feb	6.000	0.000
6	28-Feb	6.000	0.000
7	29-Feb	3.000	0.000
8	01-Mar	12.000	0.000
9	02-Mar	17.000	0.000
10	03-Mar	14.000	0.000
11	04-Mar	53.000	1.000
12	05-Mar	73.000	2.000
13	06-Mar	85.000	3.000
14	07-Mar	90.000	11.000
15	08-Mar	112.000	13.000
16	09-Mar	141.000	17.000
17	10-Mar	203.000	19.000

	Unnamed: 0	Activos	Recuperados
18	11-Mar	252.000	20.000
19	12-Mar	311.000	31.000
20	13-Mar	419.000	37.000
21	14-Mar	561.000	45.000
22	15-Mar	722.000	54.000
23	16-Mar	974.000	75.000
24	17-Mar	1.210	88.000
25	18-Mar	1.455	101.000
26	19-Mar	1.712	125.000
27	20-Mar	2.028	158.000
28	21-Mar	2.261	222.000
29	22-Mar	2.506	272.000
30	23-Mar	2.810	342.000
31	24-Mar	3.006	456.000
32	25-Mar	3.179	606.000
33	26-Mar	3.343	776.000
34	27-Mar	3.403	1.049
35	28-Mar	3.473	1.298
36	29-Mar	3.491	1.556
37	30-Mar	3.580	1.837
38	31-Mar	3.545	2.186
39	01-Abr	3.687	2.483

In [26]:
```
#Arreglamos la escala de los datos
a=dfar.iloc[24:]["Activos"]*1000
b=dfar.iloc[0:24]["Activos"]
```

```
c=b.append(a)
d=dfar.iloc[34:]["Recuperados"]*1000
e=dfar.iloc[0:34]["Recuperados"]
f=e.append(d)
dfar["Activos"]=c
dfar["Recuperados"]=f
dfar.head(40)
```

Out[26]:

	Unnamed: 0	Activos	Recuperados
0	22-Feb	1.0	0.0
1	23-Feb	2.0	0.0
2	24-Feb	2.0	0.0
3	25-Feb	9.0	0.0
4	26-Feb	10.0	0.0
5	27-Feb	6.0	0.0
6	28-Feb	6.0	0.0
7	29-Feb	3.0	0.0
8	01-Mar	12.0	0.0
9	02-Mar	17.0	0.0
10	03-Mar	14.0	0.0
11	04-Mar	53.0	1.0
12	05-Mar	73.0	2.0
13	06-Mar	85.0	3.0
14	07-Mar	90.0	11.0
15	08-Mar	112.0	13.0
16	09-Mar	141.0	17.0
17	10-Mar	203.0	19.0
18	11-Mar	252.0	20.0
19	12-Mar	311.0	31.0

	Unnamed: 0	Activos	Recuperados
20	13-Mar	419.0	37.0
21	14-Mar	561.0	45.0
22	15-Mar	722.0	54.0
23	16-Mar	974.0	75.0
24	17-Mar	1210.0	88.0
25	18-Mar	1455.0	101.0
26	19-Mar	1712.0	125.0
27	20-Mar	2028.0	158.0
28	21-Mar	2261.0	222.0
29	22-Mar	2506.0	272.0
30	23-Mar	2810.0	342.0
31	24-Mar	3006.0	456.0
32	25-Mar	3179.0	606.0
33	26-Mar	3343.0	776.0
34	27-Mar	3403.0	1049.0
35	28-Mar	3473.0	1298.0
36	29-Mar	3491.0	1556.0
37	30-Mar	3580.0	1837.0
38	31-Mar	3545.0	2186.0
39	01-Abr	3687.0	2483.0

In [27]:
```
#Incluimos un tiempo normalizado
dfar["t"]=pd.Series([i for i in range(0,316)])/315
print(dfar.head())
print(dfar.tail())
```

```
  Unnamed: 0  Activos  Recuperados         t
```

```
0     22-Feb      1.0          0.0  0.000000
1     23-Feb      2.0          0.0  0.003175
2     24-Feb      2.0          0.0  0.006349
3     25-Feb      9.0          0.0  0.009524
4     26-Feb     10.0          0.0  0.012698
    Unnamed: 0  Activos  Recuperados         t
311     29-Dic  24657.0     575800.0  0.987302
312     30-Dic  24745.0     577610.0  0.990476
313     31-Dic  23847.0     579369.0  0.993651
314     01-Ene  21381.0     582085.0  0.996825
315     02-Ene  18988.0     584535.0  1.000000
```

In [28]:
```
dfar=dfar.drop("Unnamed: 0",axis=1)
dfar["Activos"]=dfar["Activos"]/N
dfar["Recuperados"]=dfar["Recuperados"]/N
dfar=dfar[["t","Activos","Recuperados"]]
dfargraph=dfar
dfar.head()
```

Out[28]:

	t	Activos	Recuperados
0	0.000000	5.274807e-08	0.0
1	0.003175	1.054961e-07	0.0
2	0.006349	1.054961e-07	0.0
3	0.009524	4.747327e-07	0.0
4	0.012698	5.274807e-07	0.0

En este caso los datos reportan que sí se comienza con un caso, por lo tanto la condición inicial queda de la forma

$$(S_0, I_0, R_0) = (\frac{N-1}{N}, \frac{1}{N}, 0)$$

In [29]:
```
#Condiciones iniciales
def S_0():
    return (N-1)/N
def I_0():
    return 1/N
def R_0():
    return 0
```

In [30]:
```
dfar=dfar.drop(index=0)
dfar.head()
```

Out[30]:

	t	Activos	Recuperados
1	0.003175	1.054961e-07	0.0
2	0.006349	1.054961e-07	0.0
3	0.009524	4.747327e-07	0.0
4	0.012698	5.274807e-07	0.0
5	0.015873	3.164884e-07	0.0

In [31]:
```
dfar=dfar.reset_index()
del dfar["index"]
dfar.head()
```

Out[31]:

	t	Activos	Recuperados
0	0.003175	1.054961e-07	0.0
1	0.006349	1.054961e-07	0.0
2	0.009524	4.747327e-07	0.0
3	0.012698	5.274807e-07	0.0
4	0.015873	3.164884e-07	0.0

In [32]:
```
dfar1=dfar.drop("Recuperados",axis=1)
dfar1.head()
```

Out[32]:

	t	Activos
0	0.003175	1.054961e-07
1	0.006349	1.054961e-07
2	0.009524	4.747327e-07
3	0.012698	5.274807e-07

	t	Activos
4	0.015873	3.164884e-07

In [33]:
```
z=[]
for k in np.linspace(0,10,100):
        modelo2 = pde.PDEmodel(dfar1, SIR , [S_0, I_0, R_0], bounds=[(0,k), (0,0.1)],
                               param_names=[r'$b$', r'$c$'], nvars=3, ndims=0,nreplicates=1,obsidx=[1], outfunc=None)
        modelo2.fit()
        z.append((modelo2.best_error,k))
```

```
   $b$       $c$
0  0.0  0.000497
        $b$       $c$
0  0.099336  0.001735
        $b$       $c$
0  0.195441  0.007966
        $b$       $c$
0  0.266667  0.002241
        $b$       $c$
0  0.403046  0.002109
        $b$      $c$
0  0.500459  0.00545
        $b$       $c$
0  0.597926  0.008636
       $b$       $c$
0  0.68609  0.028689
        $b$       $c$
0  0.794468  0.037601
      $b$       $c$
0  0.9079  0.043173
        $b$       $c$
0  0.994785  0.029831
        $b$       $c$
0  1.106469  0.008567
        $b$       $c$
0  1.209781  0.090184
        $b$       $c$
0  1.295427  0.038916
        $b$       $c$
0  1.391639  0.006173
        $b$       $c$
0  1.465478  0.033289
```

```
0  8.279219  0.009622
       $b$       $c$
0  8.436033  0.037418
       $b$      $c$
0  8.578996  0.01551
       $b$       $c$
0  8.669396  0.006242
       $b$       $c$
0  8.760131  0.012062
       $b$       $c$
0  8.884742  0.011133
      $b$       $c$
0  8.98308  0.022923
       $b$       $c$
0  9.076801  0.004029
       $b$      $c$
0  9.191032  0.01797
       $b$      $c$
0  9.282728  0.00829
       $b$       $c$
0  9.385119  0.010453
       $b$       $c$
0  9.480049  0.005963
      $b$       $c$
0  9.59363  0.000384
       $b$       $c$
0  9.694713  0.000133
       $b$      $c$
0  9.791362  0.00087
       $b$       $c$
0  9.889737  0.000234
      $b$       $c$
0  9.99924  0.010188
```

In [34]:
```
z=pd.DataFrame(z,columns=["Error","k"])
z=z.sort_values(by=["Error"], ascending=True)
x=z.iloc[0]
x
```

Out[34]:
```
Error     0.000002
k        10.000000
Name: 99, dtype: float64
```

In [35]:
```
%%time
```

```
modelo2 = pde.PDEmodel(dfar1, SIR , [S_0, I_0, R_0], bounds=[(0,x.loc["k"]), (0, 0.1)],
                       param_names=[r'$b$', r'$c$'], nvars=3, ndims=0,nreplicates=1,obsidx=[1], outfunc=None)
modelo2.fit()
```

```
        $b$       $c$
0  9.992306  0.008109
Wall time: 333 ms
```

In [36]:
```
modelo2.best_params
```

Out[36]:

	b	c
0	9.992306	0.008109

In [37]:
```
modelo2.best_error
```

Out[37]:
```
1.6479297144590835e-06
```

In [38]:
```
%%time
modelo2.likelihood_profiles()
```

```
Wall time: 50.2 s
```

In [39]:
```
modelo2.result_profiles
```

Out[39]:

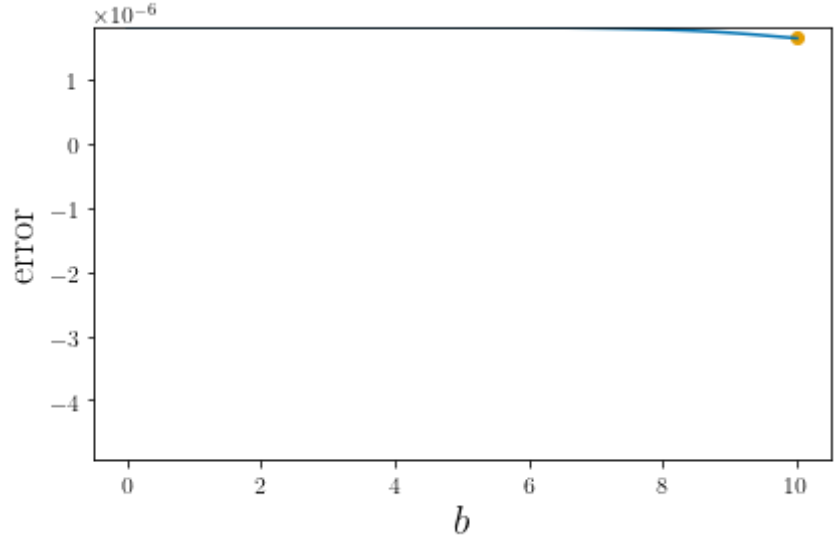
	parameter	value	error
0	b	0.00000	0.000002
1	b	0.10101	0.000002
2	b	0.20202	0.000002
3	b	0.30303	0.000002
4	b	0.40404	0.000002
...	...	...	...
195	c	0.09596	0.000002

	parameter	value	error
196	c	0.09697	0.000002
197	c	0.09798	0.000002
198	c	0.09899	0.000002
199	c	0.10000	0.000002

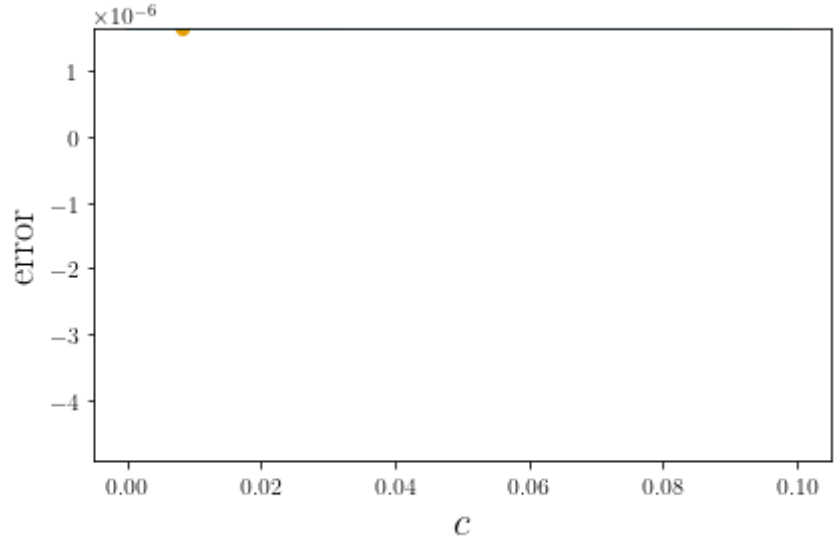
200 rows × 3 columns

In [40]:
```
modelo2.plot_profiles()
```

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



In [41]:
```
act_datos=np.array(list(dfargraph["Activos"]))
rec_datos=np.array(list(dfargraph["Recuperados"]))
dfargraph.tail()
```

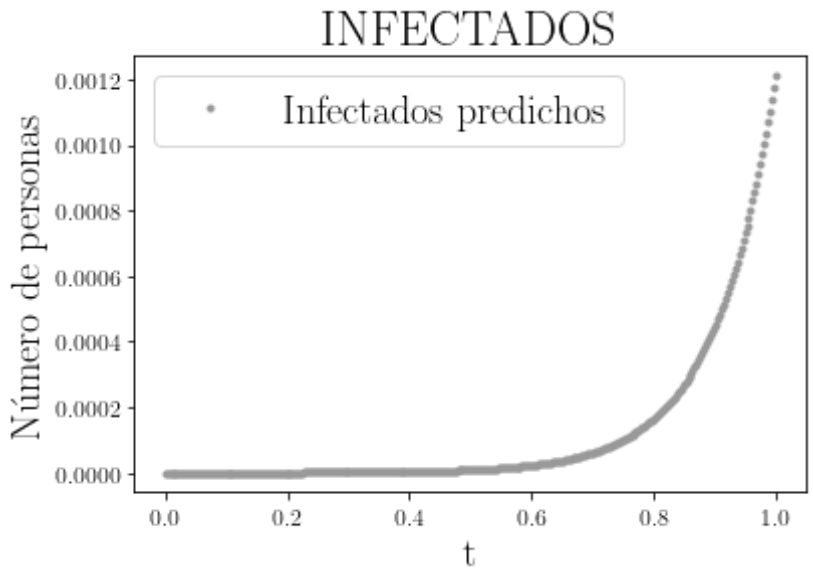
Out[41]:

	t	Activos	Recuperados
311	0.987302	0.001301	0.030372
312	0.990476	0.001305	0.030468
313	0.993651	0.001258	0.030561
314	0.996825	0.001128	0.030704
315	1.000000	0.001002	0.030833

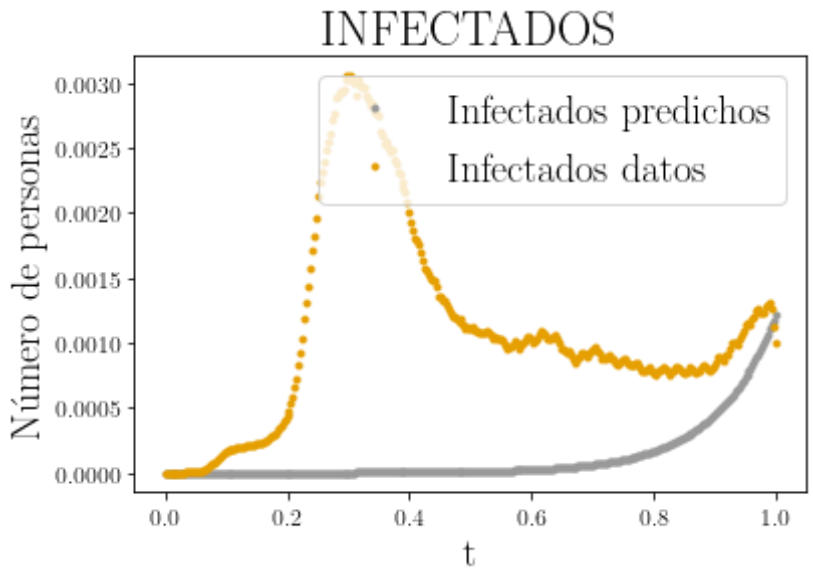
In [42]:
```
b,c = modelo2.best_params.loc[0,"$b$"],modelo2.best_params.loc[0,"$c$"]

def dP_dt(P, t):
    return [-b*P[0]*P[1],b*P[0]*P[1]-c*P[1] ,c*P[1]]

ts = np.linspace(0, 1, 316)
P0 = [(N-1)/N,1/N,0]
Solucion2 = odeint(dP_dt, P0, ts)
```

```
Suceptibles2 = Solucion2[:,0]
Infectados2 = Solucion2[:,1]
Recuperados2 = Solucion2[:,2]
```

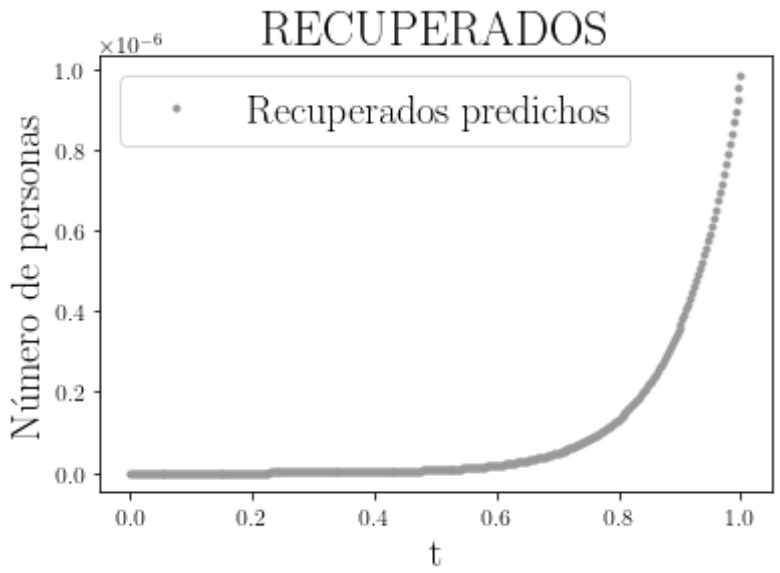
In [43]:
```
plt.plot(ts, Infectados2, ".", label="Infectados predichos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("INFECTADOS")
plt.legend();
```



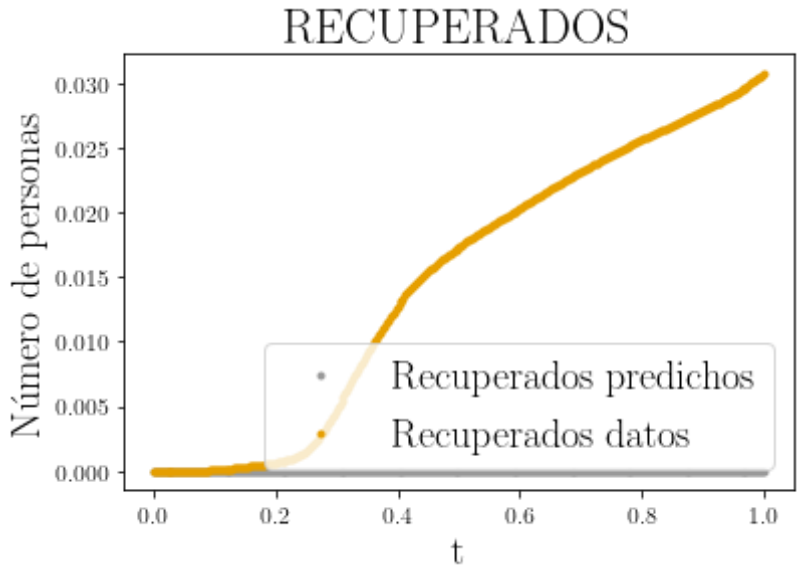
In [44]:
```
plt.plot(ts, Infectados2, ".", label="Infectados predichos")
plt.plot(ts,act_datos, ".", label="Infectados datos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("INFECTADOS")
plt.legend();
```



In [45]:
```
plt.plot(ts, Recuperados2, ".", label="Recuperados predichos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("RECUPERADOS")
plt.legend();
```



In [46]:
```
plt.plot(ts, Recuperados2, ".", label="Recuperados predichos")
plt.plot(ts, rec_datos, ".", label="Recuperados datos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("RECUPERADOS")
plt.legend();
```



Estimación de parámetros utilizando los datos de activos y recuperados.

In [47]:
```
m=[]
for k in np.linspace(0,10,100):
    modelo3 = pde.PDEmodel(dfar, SIR , [S_0, I_0, R_0], bounds=[(0,k), (0,0.1)],
                           param_names=[r'$b$', r'$c$'], nvars=3, ndims=0,nreplicates=1,obsidx=[1,2], outfunc=Nor
    modelo3.fit()
    m.append((modelo3.best_error,k))
```

```
   $b$      $c$
0  0.0  0.09911
        $b$       $c$
0  0.097505  0.098701
       $b$       $c$
0  0.16709  0.091391
        $b$     $c$
0  0.274746  0.0906
        $b$       $c$
0  0.357754  0.098108
        $b$       $c$
0  0.392107  0.094206
        $b$       $c$
0  0.558302  0.086454
```

```
        $b$       $c$
0  9.588741  0.092066
        $b$       $c$
0  9.658813  0.087769
        $b$       $c$
0  9.84646  0.083761
```

In [48]:
```
m=pd.DataFrame(m,columns=["Error","k"])
m=m.sort_values(by=["Error"], ascending=True)
l=m.iloc[0]
l
```

Out[48]:
```
Error     0.000168
k        10.000000
Name: 99, dtype: float64
```

In [49]:
```
%%time
modelo3 = pde.PDEmodel(dfar, SIR , [S_0, I_0, R_0], bounds=[(0,l.loc["k"]), (0,0.1)],
                       param_names=[r'$b$', r'$c$'], nvars=3, ndims=0,nreplicates=1,obsidx=[1,2], outfunc=None)
modelo3.fit()
```

```
        $b$       $c$
0  9.832183  0.016266
Wall time: 115 ms
```

In [50]:
```
modelo3.best_params
```

Out[50]:

	b	c
0	9.832183	0.016266

In [51]:
```
modelo3.best_error
```

Out[51]:
```
0.00016822170042460693
```

In [52]:
```
%%time
modelo3.likelihood_profiles()
```

```
Wall time: 15.8 s
```

In [53]:
```
modelo3.result_profiles
```

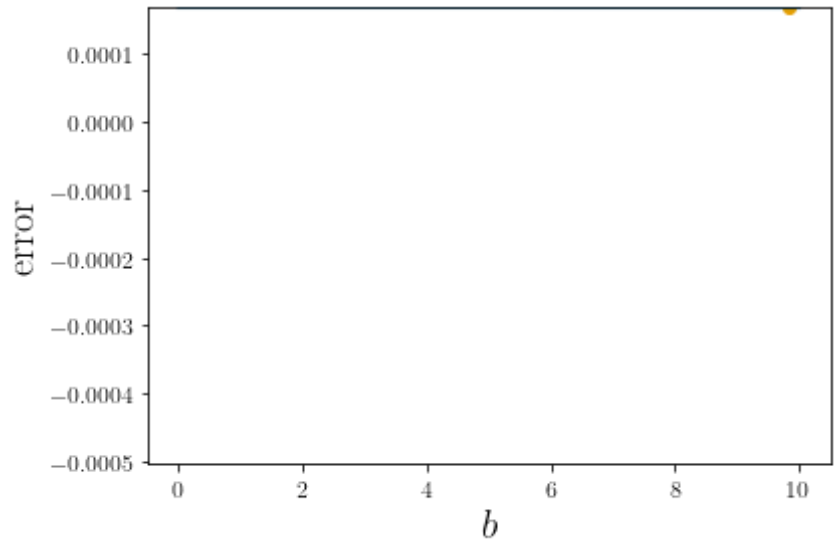
Out[53]:

	parameter	value	error
0	b	0.00000	0.000168
1	b	0.10101	0.000168
2	b	0.20202	0.000168
3	b	0.30303	0.000168
4	b	0.40404	0.000168
...	...	...	...
195	c	0.09596	0.000168
196	c	0.09697	0.000168
197	c	0.09798	0.000168
198	c	0.09899	0.000168
199	c	0.10000	0.000168

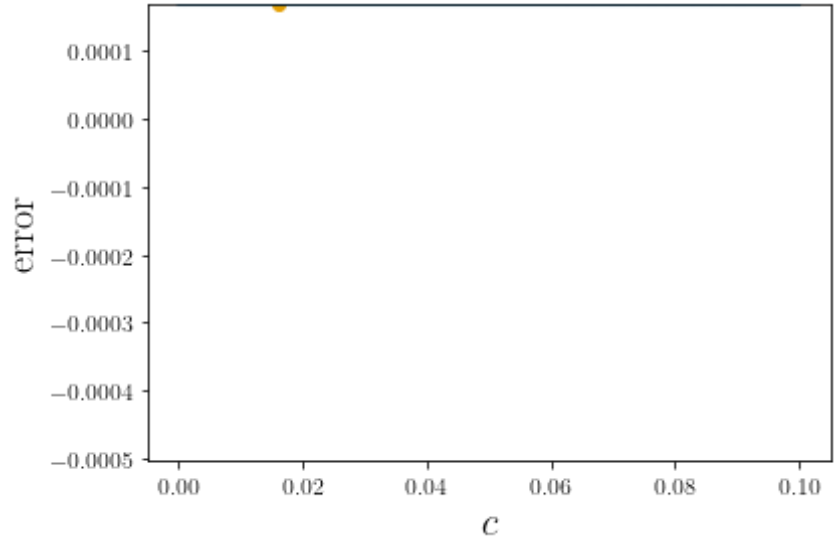
200 rows × 3 columns

In [54]:
```
modelo3.plot_profiles()
```

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



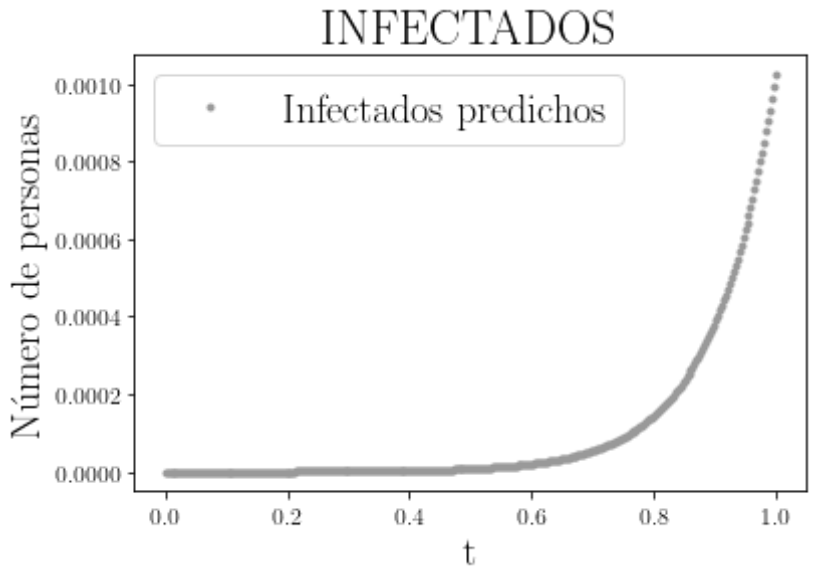

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have pre
cedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array wi
th a single row if you intend to specify the same RGB or RGBA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



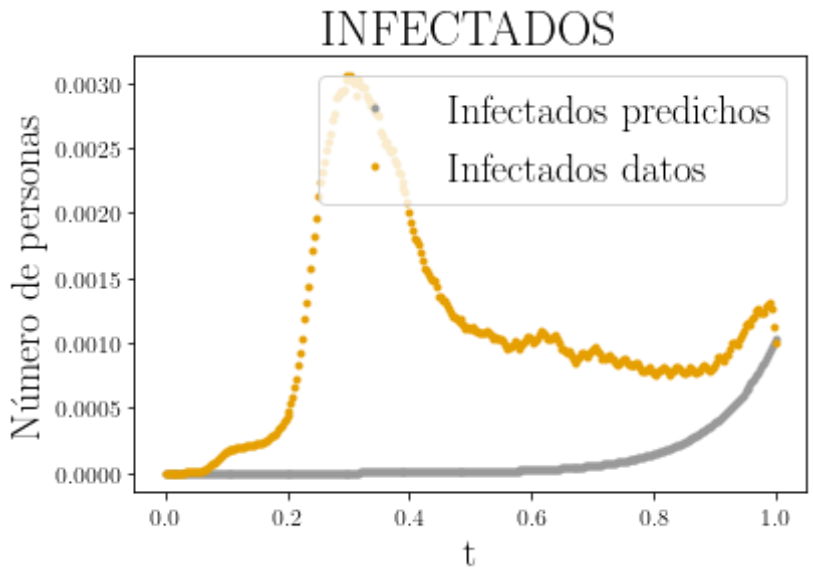
In [55]:
```
b,c = modelo3.best_params.loc[0,"$b$"],modelo3.best_params.loc[0,"$c$"]
```

```
def dP_dt(P, t):
    return [-b*P[0]*P[1],b*P[0]*P[1]-c*P[1] ,c*P[1]]

ts = np.linspace(0, 1, 316)
P0 = [(N-1)/N,1/N,0]
Solucion3 = odeint(dP_dt, P0, ts)
Suceptibles3 = Solucion3[:,0]
Infectados3 = Solucion3[:,1]
Recuperados3 = Solucion3[:,2]
```

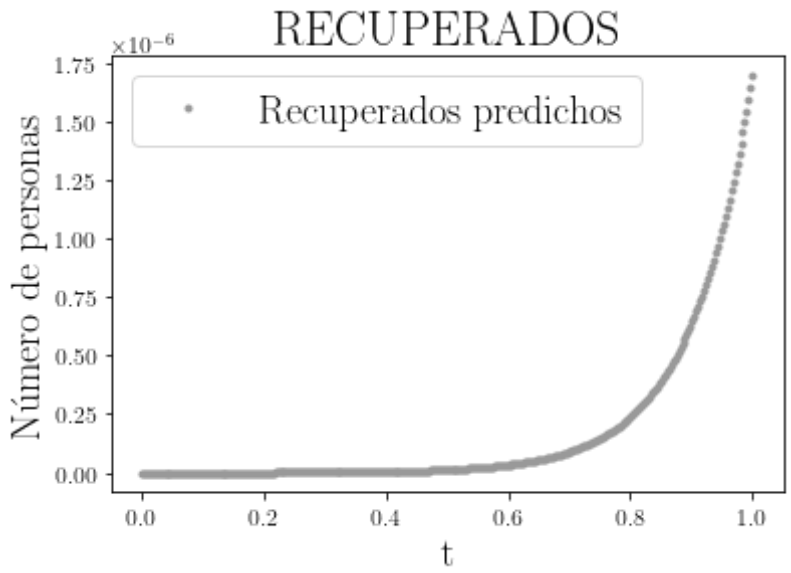
In [56]:
```
plt.plot(ts, Infectados3, ".", label="Infectados predichos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("INFECTADOS")
plt.legend();
```



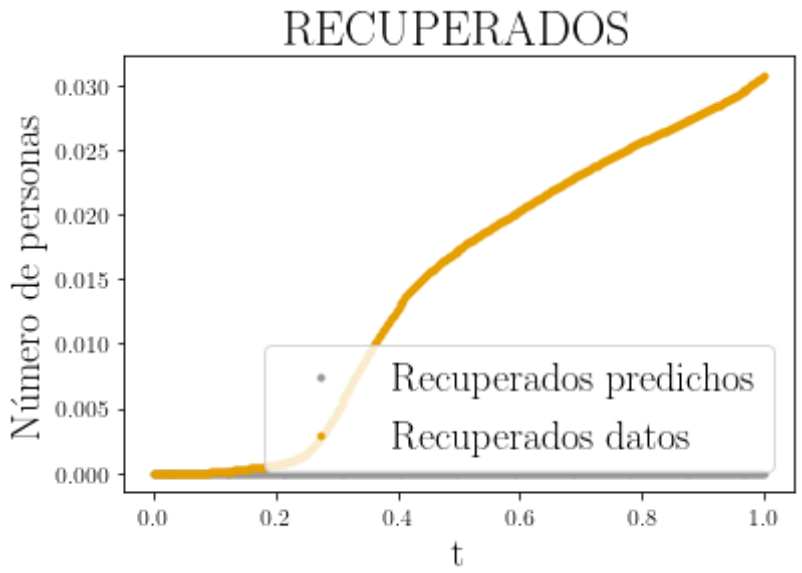
In [57]:
```
plt.plot(ts, Infectados3, ".", label="Infectados predichos")
plt.plot(ts,act_datos, ".", label="Infectados datos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("INFECTADOS")
plt.legend();
```



In [58]:
```
plt.plot(ts, Recuperados3, ".", label="Recuperados predichos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("RECUPERADOS")
plt.legend();
```



In [59]:
```
plt.plot(ts, Recuperados3, ".", label="Recuperados predichos")
plt.plot(ts, rec_datos, ".", label="Recuperados datos")
plt.xlabel("t")
plt.ylabel("Número de personas")
plt.title("RECUPERADOS")
plt.legend();
```



Referencias

https://www.gob.cl/coronavirus/cifrasoficiales/#datos

https://www.fcfm.buap.mx/assets/docs/docencia/tesis/matematicas/EmileneCarmelitaPliegoPliego.pdf

http://mat.uab.cat/matmat_antiga/PDFv2013/v2013n03.pdf

https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model