IMT Epidemic Models

Release v0.1

Mar 12, 2021

Modelos Analíticos

1 Alguns resultados do século XX... **3**

2 Previsões da COVID-19 **5**
- 2.1 Determinação do R_0 . . . 5
- 2.2 Previsões do consumo do sistema público . . . 6
- 2.3 Previsões dos picos epidêmicos da COVID-19 . . . 6

3 Autores **7**
- 3.1 **Modelo SIR** . . . 7
- 3.2 **Modelo Reed Frost** . . . 10
- 3.3 **SIR Dados Artificiais** . . . 14
- 3.4 **SIR Reino Unido 1944-1964** . . . 19
- 3.5 **SIR Reino Unido Modelos Espaciais** . . . 33
- 3.6 **SIR Parâmetros variantes** . . . 41
- 3.7 **SIR Brasil - Dados Reais** . . . 46
- 3.8 **SIR China - Dados Reais** . . . 55
- 3.9 **SIR Itália - Dados Reais** . . . 73
- 3.10 **SIR Alemanha - Dados Reais** . . . 90
- 3.11 epidemicModels . . . 107
- 3.12 Vanderlei Cunha Parro . . . 113
- 3.13 Marcelo Mendes Lafetá Lima . . . 113
- 3.14 Felipe Brandão Ippolito . . . 113
- 3.15 Felipe Antonio Silva de Andrade . . . 113

4 Indices e tabelas **115**

Python Module Index **117**

Index **119**

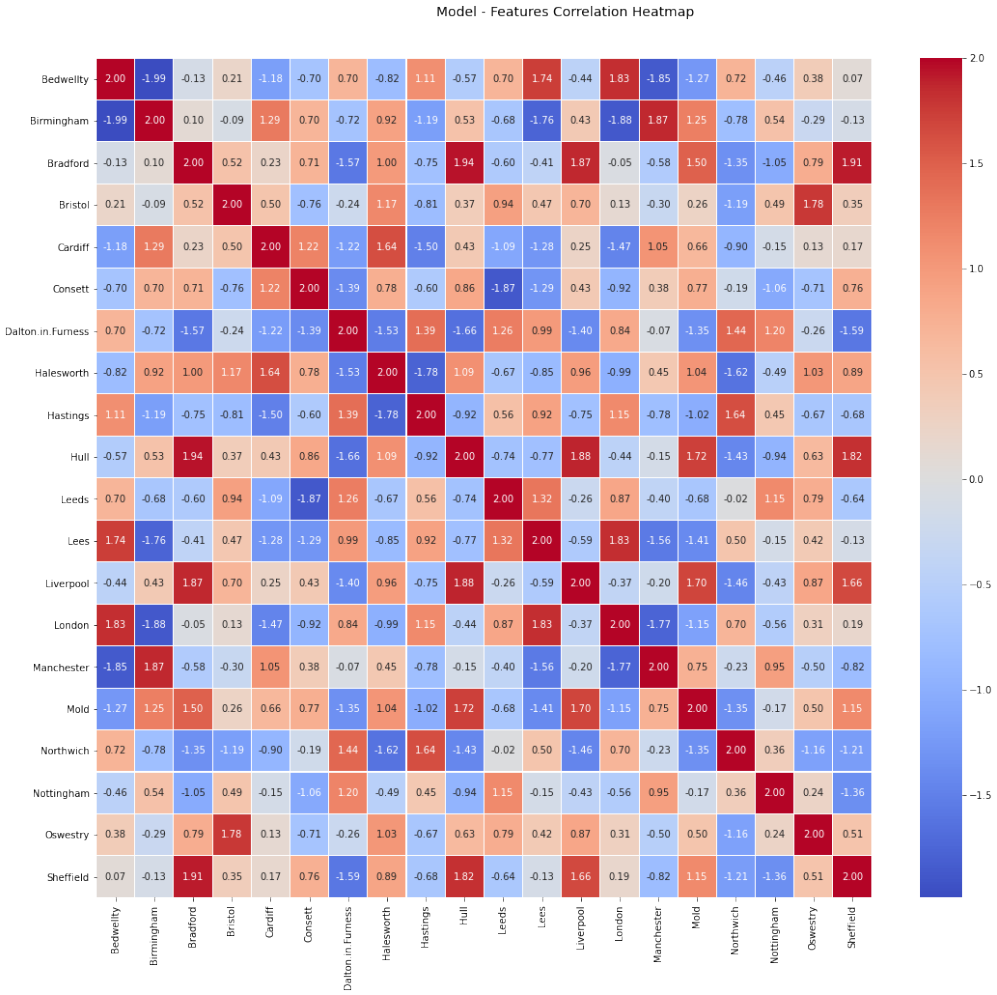
Este projeto tem o objetivo de divulgar como desenvolver modelos para epidemias, desde sua modelagem matemática até sua concepção computacional em Python. Sendo assim composto por vídeos explicativos, notebooks em Python, e diversas visualizações, para ajudar com o entendimento do conteúdo apresentado.

CHAPTER 1

Alguns resultados do século XX...

Para exemplificar o desempenho do conteúdo apresentado, foi feita uma análise com dados do século passado do Reino Unido (United Kingdom), e utilizando os modelos aqui desenvolvidos é possível obter previsões do comportamento dos dados. Essas previsões, juntamente com os dados reais, estão apresentadas na figura a seguir:

Assim como o estudo de correlação entre os modelos SIR obtidos para cada uma das cidades durante os períodos de epidemias do Reino Unido:


Model - Features Correlation Heatmap
Bedwellty 2.00 -1.99 -0.13 0.21 -1.18 -0.70 0.70 -0.82 1.11 -0.57 0.70 1.74 -0.44 1.83 -1.85 -1.27 0.72 -0.46 0.38 0.07
Birmingham -1.99 2.00 0.10 -0.09 1.29 0.70 -0.72 0.92 -1.19 0.53 -0.68 -1.76 0.43 -1.88 1.87 1.25 -0.78 0.54 -0.29 -0.13
Bradford -0.13 0.10 2.00 0.52 0.23 0.71 -1.57 1.00 -0.75 1.94 -0.60 -0.41 1.87 -0.05 -0.58 1.50 -1.35 -1.05 0.79 1.91
Bristol 0.21 -0.09 0.52 2.00 0.50 -0.76 -0.24 1.17 -0.81 0.37 0.94 0.47 0.70 0.13 -0.30 0.26 -1.19 0.49 1.78 0.35
Cardiff -1.18 1.29 0.23 0.50 2.00 1.22 -1.22 1.64 -1.50 0.43 -1.09 -1.28 0.25 -1.47 1.05 0.66 -0.90 -0.15 0.13 0.17
Consett -0.70 0.70 0.71 -0.76 1.22 2.00 -1.39 0.78 -0.60 0.86 -1.87 -1.29 0.43 -0.92 0.38 0.77 -0.19 -1.06 -0.71 0.76
Dalton.in.Furness 0.70 -0.72 -1.57 -0.24 -1.22 -1.39 2.00 -1.53 1.39 -1.66 1.26 0.99 -1.40 0.84 -0.07 -1.35 1.44 1.20 -0.26 -1.59
Halesworth -0.82 0.92 1.00 1.17 1.64 0.78 -1.53 2.00 -1.78 1.09 -0.67 -0.85 0.96 -0.99 0.45 1.04 -1.62 -0.49 1.03 0.89
Hastings 1.11 -1.19 -0.75 -0.81 -1.50 -0.60 1.39 -1.78 2.00 -0.92 0.56 0.92 -0.75 1.15 -0.78 -1.02 1.64 0.45 -0.67 -0.68
Hull -0.57 0.53 1.94 0.37 0.43 0.86 -1.66 1.09 -0.92 2.00 -0.74 -0.77 1.88 -0.44 -0.15 1.72 -1.43 -0.94 0.63 1.82
Leeds 0.70 -0.68 -0.60 0.94 -1.09 -1.87 1.26 -0.67 0.56 -0.74 2.00 1.32 -0.26 0.87 -0.40 -0.68 -0.02 1.15 0.79 -0.64
Lees 1.74 -1.76 -0.41 0.47 -1.28 -1.29 0.99 -0.85 0.92 -0.77 1.32 2.00 -0.59 1.83 -1.56 -1.41 0.50 -0.15 0.42 -0.13
Liverpool -0.44 0.43 1.87 0.70 0.25 0.43 -1.40 0.96 -0.75 1.88 -0.26 -0.59 2.00 -0.37 -0.20 1.70 -1.46 -0.43 0.87 1.66
London 1.83 -1.88 -0.05 0.13 -1.47 -0.92 0.84 -0.99 1.15 -0.44 0.87 1.83 -0.37 2.00 -1.77 -1.15 0.70 -0.56 0.31 0.19
Manchester -1.85 1.87 -0.58 -0.30 1.05 0.38 -0.07 0.45 -0.78 -0.15 -0.40 -1.56 -0.20 -1.77 2.00 0.75 -0.23 0.95 -0.50 -0.82
Mold -1.27 1.25 1.50 0.26 0.66 0.77 -1.35 1.04 -1.02 1.72 -0.68 -1.41 1.70 -1.15 0.75 2.00 -1.35 -0.17 0.50 1.15
Northwich 0.72 -0.78 -1.35 -1.19 -0.90 -0.19 1.44 -1.62 1.64 -1.43 -0.02 0.50 -1.46 0.70 -0.23 -1.35 2.00 0.36 -1.16 -1.21
Nottingham -0.46 0.54 -1.05 0.49 -0.15 -1.06 1.20 -0.49 0.45 -0.94 1.15 -0.15 -0.43 -0.56 0.95 -0.17 0.36 2.00 0.24 -1.36
Oswestry 0.38 -0.29 0.79 1.78 0.13 -0.71 -0.26 1.03 -0.67 0.63 0.79 0.42 0.87 0.31 -0.50 0.50 -1.16 0.24 2.00 0.51
Sheffield 0.07 -0.13 1.91 0.35 0.17 0.76 -1.59 0.89 -0.68 1.82 -0.64 -0.13 1.66 0.19 -0.82 1.15 -1.21 -1.36 0.51 2.00
Bedwellty
Birmingham
Bradford
Bristol
Cardiff
Consett
Dalton.in.Furness
Halesworth
Hastings
Hull
Leeds
Lees
Liverpool
London
Manchester
Mold
Northwich
Nottingham
Oswestry
Sheffield
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

CHAPTER 2

Previsões da COVID-19

Nesta análise utilizamos o modelo desenvolvido para tentar prever os comportamentos da COVID-19. Para isso, primeiramente utilizamos dados de países que já apresentam um comportamento característico da estrutura SIR e já estão no seu período de amortecimento da curva de infectados. Desta forma podemos validar o modelo com relação a sua capacidade de prever eventos futuros, mesmo que somente poucos dias de dados sejam utilizados. Desta forma algumas análises específicas, e de maior impacto, são apresentadas nessa primeira página:

- Determinação do número básico de reprodução R_0 no decorrer da epidemia
- Previsão da quantidade de infectados notificados no sistema público
- Previsão do momento de pico da epidemia

No caso, essas análises foram feitas para Itália , China e Alemanha , países que já estão em seu período de diminuição do nível de infectados. Para exemplificar a capacidade de previsão do algoritmo de aprendizado desenvolvido, nas visualizações a seguir nós mostramos a previsão feita pelo algoritmo para cada país a medida que os tempo da epidemia foi passando e mais dados foram utilizados para a aprendizagem:

Nosso modelo aprende com algoritmos de otimização os parâmetros da estrutura SIR, juntamente com a proporção da população que está sendo registrada pelo sistema de saúde. No caso, ele aprende três parâmetros fundamentais do modelo: β (contatos por dia), r (em que $1/r$ é o tempo médio de recuperação da doença), e o $S(0)$ (quantidade inicial de suscetíveis), para que a epidemia tenha o comportamento que os dados mostram. No caso os valores ajustados para cada país foram:

Caso queira tentar ajustar os parâmetros você mesmo, basta clicar no botão abaixo!

2.1 Determinação do R_0

Note que o parâmetro R_0, é determinado a partir dos dois outros parâmetros característicos do modelo SIR, $R_0 = \beta/r$. Aqui utilizamos um modelo SIR que pondera a quantidade de suscetíveis, uma vez que nem toda a população pode ser considerada suscetível, visto que nem todas as pessoas infectadas e recuperadas são notificadas ao sistema público. Além disso, nem toda a população é exposta ao vírus, devido a políticas públicas, isolamentos, etc.

Note: Desta forma, note que o modelo desenvolvido somente modela as pessoas notificadas pelo sistema de saúde, sendo assim, representativo de uma parte da verdade situação do país.

Dito isso, é possível definir os valores encontrados pelo algoritmo de aprendizagem para o parâmetro R_0 a medida que os dias da pandemia passaram:

2.2 Previsões do consumo do sistema público

Um dos parâmetros que nosso algoritmo aprende durante seu processo de treinamento é um parâmetro que pondera a quantidade da população de suscetíveis inicial (simplesmente uma técnica para melhorar o condicionamento númerico do algoritmo). Porém, como esse parâmetro tende sempre a estimar o valor de $S(0)$ igual ao valor de $R(\infty)$. Note que sempre é verdade $S(0) \geq R(\infty)$. Como os dados medidos são somente das pessoas notificadas e acompanhadas pelo sistema de saúde, podemos concluir que o valor de $R(\infty)$ é a quantidade de pessoas que foram contaminadas, frequentaram o sistema de saúde para o diagnóstico e por isso estão na base de dados. Nosso algoritmo prevê a quantidade de $S(0) = R(\infty)$, desta forma para cada novo dia de dados temos uma nova previsão de qual será a quantidade de pessoas absorvidas pelo sistema de saúde. Nos gráficos a seguir conseguimos mostrar o erro percentual entre o real valor de $R(\infty)$ e o valor estimado por nosso modelo a cada dia da epidemia:

2.3 Previsões dos picos epidêmicos da COVID-19

Nesta análise apresentamos o efeito da quantidade de dados na performance do modelo desenvolvido analisando a capacidade de prever o dia em que acontecerá o pico da quantidade de infectados da epidemia da COVID-19. Para isso estamos utilizando dados de países que já tiveram seu pico de contágio e atualmente estão no período de amortecimento da quantidade de infectados. Alguns dos países analisados foram a China , Itália e Alemanha , que possibilitaram as análises abaixo. Nestas figuras é mostrado o erro do modelo ao tentar prever o dia de pico, para cada dia da epidemia:

CHAPTER 3

Autores

3.1 Modelo SIR

[1]:
```
from IPython.display import IFrame

# Youtube
IFrame("https://www.youtube.com/embed/v-91XHpjugo", "100%", 500)
```

[1]:
```
<IPython.lib.display.IFrame at 0x106ea34d0>
```

3.1.1 Configurações

[2]:
```
import numpy as np
import pandas as pd
from scipy.integrate import odeint

from bokeh.palettes import brewer
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

output_notebook()
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

3.1.2 Suscetível, Infectado e Removido (Recuperado)

A proposta é introduzir um modelo determinístico para análise de evolução de uma epidemia. O modelo aqui representado foi inicialmente proposto por Kermack, W. and McKendrick, A., 1927. A contribution to the mathematical theory of epidemics. Proc. R. Soc. London A 115, 700-721. O modelo tem as seguintes premissas:

1. Uma posulação invariante no tempo com **N** individuos (fechada).
2. Taxas constantes: contato e remoção.
3. Desconsiderando variações demográficas: nascimentos e mortes.
4. População "bem misturada".

O modelo é fundamentado em três estados: **S** - suscetível, **I** - infectado e **R** - removido ou recuperado.

$$S(t) + I(t) + R(t) = N$$

[3]:
```
# Tamanho da populção - N
N = 500

# Valores iniciais
I0 = 1
R0 = 0
S0 = N - I0
```

O conjunto de equações diferenciais que caracteriza o modelo é descrito abaixo. No modelo

$$\beta - \text{representa a taxa de transmissão ou taxa efetiva de contato}$$

e

$$r - \text{a taxa de remoção ou recuperação.}$$

$$\frac{dS(t)}{dt} = -\beta S(t) I(t)$$
$$\frac{dI(t)}{dt} = \beta S(t) I(t) - rI(t)$$
$$\frac{dR(t)}{dt} = rI(t)$$

[4]:
```
# Definição do conjunto de equações diferencias não lineares que formam o modelo.

def SIRm(y, t, N, Beta, r):
    S, I, R = y
    Sdot = -(Beta * S * I)
    Idot = (Beta * S * I)  - r * I
    Rdot = r * I
    return Sdot, Idot, Rdot
```

3.1.3 Analisando a evolução de β e r

[5]:
```
# Beta -  taxa de contato,
# r - taxa média de recuperação (in 1/dia).
Beta = 0.0009
r = 1e-1

# Resolução da simulação - Escala temporal (dias)
t = np.linspace(0, 100, 1000)

# Vetor de condições iniciais
y0 = S0, I0, R0

# Integrando as Equações do modelo SIR
ret = odeint(SIRm, y0, t, args=(N, Beta, r))
S, I, R = ret.T

# Visualizando a evolução da Epidemia - S(t), I(t) e R(t)
p = figure(tools="pan,hover,lasso_select",
           x_range=(0, t[-1]), y_range=(0, N),
           plot_width=600, plot_height=400)

p.line(t, S, legend_label="Suscetíveis", color="#ffd885", line_width=3)
p.line(t, I, legend_label="Infectados",  color="#de425b", line_width=3)
p.line(t, R, legend_label="Removidos",   color="#99d594", line_width=3)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"
p.legend.click_policy = "mute"
p.legend.items.reverse()

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Salvando o modelo

[6]:
```
import pandas as pd

output = pd.DataFrame({"S": S, "I": I, "R": R})
output.to_csv('SIRpd', index=False)

print(output)
```

```
              S         I           R
0    499.000000  1.000000    0.000000
1    498.954252  1.035561    0.010187
```

(continues on next page)

(continued from previous page)

```
2    498.906883  1.072381    0.020736
3    498.857833  1.110506    0.031660
4    498.807046  1.149981    0.042973
..          ...       ...         ...
995    5.855617  0.232760  493.911622
996    5.855495  0.230564  493.913941
997    5.855374  0.228388  493.916238
998    5.855254  0.226232  493.918514
999    5.855135  0.224097  493.920768

[1000 rows x 3 columns]
```

3.1.4 Referências

- Material aberto para o modelo definido
- Referência para o modelo desenvolvido

3.2 Modelo Reed Frost

[1]:
```
from IPython.display import IFrame

# Youtube
IFrame("https://www.youtube.com/embed/D8S4GGmHUMw", "100%", 500)
```

[1]:
```
<IPython.lib.display.IFrame at 0x106077450>
```

3.2.1 Configurações

[2]:
```
import numpy as np
from scipy.integrate import odeint

from bokeh.palettes import brewer, Inferno10
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.layouts import gridplot

output_notebook()
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

3.2.2 Modelo para pequeno grupo

Há no senso comum o entendimento que há uma chance de "pegar" uma doença. Supondo que a probailidade que representa esta sensação seja $\mathbf{p}$ e que a probilidade de escapar desta doença seja $\mathbf{q}$. Vamos admitir um cenário onde tenhamos uma família de três pessoas $\mathbf{N} = 3$ e um dos individuos estaje **i**nfectado (**I**). Supondo que neste caso $\mathbf{p} = 0.4$ logo $\mathbf{q} = 0.6$. Admitindo este cenário pode-se explorar as seguintes progressões da doença (P_i - infectado e $\overline{P_i}$ - não infectado) :

Pensando em casos:

1. Apenas o primeiro **ifectado** desenvovle a doença.
2. O **infectado** transmite para um dos outros dois **suscetíveis** apenas.
3. Os dois **sucetíveis** são infectados um após o outro.
4. Ambos os **suscetíveis** são infectados simultaneamente.

Traduzindo em **cadeias** de transmissão:

1. $1 \rightarrow 0$
2. $1 \rightarrow 1 \rightarrow 0$
3. $1 \rightarrow 1 \rightarrow 1$
4. $1 \rightarrow 2$

Traduzindo em **probabilidades** de cada cadeia de transmissão:

1. $\overline{P_1}$ e $\overline{P_2}$ - $\mathbf{q^2}$
2. $\overline{P_1}$ e P_2 - $\mathbf{pq^2}$ **ou** P_1 e $\overline{P_2}$ - $\mathbf{pq^2}$
3. $\overline{P_1}$ e $P_2 \rightarrow P_1$ e P_2 - $\mathbf{p^2q}$ **ou** P_1 e $\overline{P_2} \rightarrow P_1$ e P_2 - $\mathbf{p^2q}$
4. P_1 e P_2 - $\mathbf{p^2}$

[3]:
```
p = 0.4
q = 1 - p

C1 = q**2
C2 = 2*p*(q**2)
C3 = 2*q*(p**2)
C4 = p**2

print(C1,C2,C3,C4, C3+C4, C1+C2+C3+C4)
```

```
0.36 0.288 0.19200000000000003 0.16000000000000003 0.3520000000000001 1.0
```

Considerando as probabilidades calculadas para cada uma das sequências, aplicada um cenário de 1000 famílias com três integrantes, pode-se estimar que cerca de **360** famílias, atigidas pela doença, terão apenas 1 infectado, **288** famílias terão pelo menos 1 infectado e **352** famílias todos serão infectados.

Se generalizarmos a análise pode-se dizer que o número de infectados $\mathbf{I_t}$ na próxima geração de infectados será $\mathbf{I_{t+1}}$. O termo $\mathbf{t}$ indica a geração da epidemia e indica com uma certa *imprecisão* a escala de tempo.

$$P(I_{t+1} = i_{t+1} | S_t = s_t, I_t = i_t) \sim \binom{s_t}{i_t} (1 - q_t^i)^{I_{t+1}} (q)^{i_t(s_t - i_{t+1})}, s_t \geq i_{t+1}$$

3.2.3 Generalizando o modelo

A partir da proposta que o contágio é dependente de uma probabilidade **p** de transmissão da doença, que corresponde a uma probabilidade **q** de escapar de ser contaminado, pode-se estabelecer um modelo que seja capaz de representar a cadeia de transmissão.

$$I_{t+1} \sim binomial(S(t), p)$$
$$S_{t+1} = S_t - I_{t+1}$$
$$R_{t+1} = R_t + I_t$$

Observe a semelhança com o modelo determinístico, exceto o fato do número de infectados ser um valor randômico, governado pela distribuição Binomial.

[4]:
```
# Modelo Reed-Frost
ngen  = 30;      # número de gerações
Sinit = 2000;    # população suscetível
Iinit = 1;       # infectados
Rinit = 0;       # removidos / recuperados

q = 0.999;

nsims = 2000;    # número de simulações

x, y = list(), list()
M = np.zeros((nsims-1,ngen-1))

TOOLS="zoom_in,zoom_out,save"
p = figure(tools=TOOLS, plot_width=600, plot_height=400)

for i in range(1,nsims-1):
    S = np.linspace(0,1,ngen) - np.linspace(0,1,ngen)
    I = np.linspace(0,1,ngen) - np.linspace(0,1,ngen)
    R = np.linspace(0,1,ngen) - np.linspace(0,1,ngen)
    S[1], I[1], R[1] = Sinit, Iinit, Rinit
    for j in range(1,ngen-1):
        # np.random.binomial(n, p, 1000)
        I[j+1] = np.random.binomial(S[j], 1 - q**I[j], 1);
        S[j+1] = S[j] - I[j+1]
        R[j+1] = R[j] + I[j]
        M[i,j] = I[j]

    p.line(range(ngen), I, line_width=2, color="#8e44ad", line_alpha=0.05)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Infectados"
p.xaxis.axis_label = "Gerações"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.2.4 Análise dos cenários para cada geração nova de infectados

Observe que para cada nova geração a distribuição estatística varia e pode-se observar o comportamento da distribuição Binomial. Neste sentido pode-se observar quem "governa" a incerteza da evolução da epidemia. A linha pontilhada em cada histograma representa a média para cada geração.

[5]:
```
num = 10
x = np.linspace(0, 100, 1000)
pallete = Inferno10

fig_list = []
for i in range(num):
    hist, edges = np.histogram(M[:,i], density=False, bins=50)
    p = figure(tools="pan,hover,lasso_select", plot_width=300, plot_height=300)
    p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:],
     fill_color=pallete[i], fill_alpha=0.5, line_color=pallete[i],
     legend_label="k = " + str(i) )
    p.grid.grid_line_alpha = 0
    p.ygrid.band_fill_alpha = 0.1
    p.ygrid.band_fill_color = "olive"
    p.yaxis.axis_label = "Frequ^encia"
    p.xaxis.axis_label = "Infectados"
    fig_list.append(p)

grid = gridplot(fig_list, ncols=2)
show(grid)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[6]:
```
num = 15

p = figure(tools="pan,hover,lasso_select", y_axis_type="log", plot_width=600, plot_
↪height=400)

for i in range(num):
    x = (i+1) * np.ones((len(M[:,i]),))
    try:
        p.scatter(x, M[:,i], size=5, fill_color=pallete[i], fill_alpha=0.05, line_
↪alpha=0)
    except:
        p.scatter(x, M[:,i], size=5, fill_color=pallete[i-10], fill_alpha=0.05, line_
↪alpha=0)
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_alpha = 0.1
p.ygrid.band_fill_color = "olive"
p.yaxis.axis_label = "Infectados"
p.xaxis.axis_label = "Gerações"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.2.5 Evolução da média e desvio padrão

Para uma visão mais clara podemos visualizar a média e o desvio padrão para cada geração.

[7]:
```
a = M.mean(0)
e = M.std(0)
x = np.linspace(0,ngen, ngen-1)

p = figure(tools="pan,hover,lasso_select", plot_width=600, plot_height=400)

p.varea(x=x, y1=a+e/2, y2=a-e/2, legend_label="Desvio Padrão", fill_alpha=0.3, fill_
↪color="#3498db")
p.line(x, a, legend_label='Média', line_cap="round", color='#e67e22', line_width=4)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_alpha = 0.1
p.ygrid.band_fill_color = "olive"
p.yaxis.axis_label = "Infectados"
p.xaxis.axis_label = "Geração"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.2.6 Referências

Inspirado no material disponível em:

- https://math.unm.edu/~sulsky/mathcamp/
- https://esc.fnwi.uva.nl/thesis/centraal/files/f70136284.pdf

3.3 SIR Dados Artificiais

[1]:
```
import pylab as pp
import numpy as np
import pandas as pd

from scipy import integrate, interpolate, optimize
from scipy.integrate import odeint

from bokeh.palettes import brewer
from bokeh.plotting import figure, show
from bokeh.io       import output_notebook
```

(continues on next page)

(continued from previous page)

```
output_notebook()
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

3.3.1 Dados artificiais

Uma prática interessante na análise de dados é testarmos se o algoritmo proposto e principalmente sua estrutura está consistente. Podemos então utilizar dados artificiais para avaliar se o processo de ajuste está estruturado corretamente.

[2]:
```
# Lendo o arquivo de dados no formato 'filename.csv'
data = pd.read_csv("./PG_IMT/DadosEpidemia/SIRpd")
# Preview das cinco primeiras linhas
data.head()
```

[2]:
```
            S         I         R
0  499.000000  1.000000  0.000000
1  498.949042  1.040746  0.010213
2  498.896013  1.083146  0.020841
3  498.840830  1.127267  0.031903
4  498.783405  1.173179  0.043415
```

[3]:
```
s_array = data[["S", "I", "R"]].to_numpy()

Sd = s_array[:,0]
Id = s_array[:,1]
Rd = s_array[:,2]
```

Gerando ruído gaussiano

[4]:
```
Sdn = np.random.normal(0, np.mean(Sd)/10, len(Sd))
Idn = np.random.normal(0, np.mean(Id)/5,  len(Sd))
Rdn = np.random.normal(0, np.mean(Rd)/10, len(Sd))

Sd = Sd + Sdn
Id = Id + Idn
Rd = Rd + Rdn
```

Visualizando os dados

[5]:
```
# Visualizando a evolução da Epidemia - S(t), I(t) e R(t)
t = np.linspace(0,len(Sd),len(Sd))

TOOLS="hover,crosshair,pan,wheel_zoom,zoom_in,zoom_out,box_zoom,undo,redo,reset,tap,
↪save,box_select,poly_select,lasso_select,"
```

(continues on next page)

(continued from previous page)

```
p = figure(tools=TOOLS, plot_width=600, plot_height=400)

p.scatter(t, Sd, legend_label="Suscetíveis - dados",
          radius=3.8, fill_color="#ffd885", fill_alpha=0.6, muted_color="#ffd885",␣
↪muted_alpha=0.2, line_color=None)
p.scatter(t, Id, legend_label="Infectados - dados",
          radius=3.8, fill_color="#de425b", fill_alpha=0.6, muted_color="#de425b",␣
↪muted_alpha=0.2, line_color=None)
p.scatter(t, Rd, legend_label="Removidos - dados",
          radius=3.8, fill_color="#99d594", fill_alpha=0.6, muted_color="#99d594",␣
↪muted_alpha=0.2, line_color=None)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"
p.legend.click_policy = "mute"
p.legend.items.reverse()

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.3.2 O problema

O conjunto de equações diferenciais que caracteriza o modelo é descrito abaixo. No modelo $:nbsphinx-math:*beta* '- *:nbsphinx-math:*text{representa a taxa de transmissão ou taxa efetiva de contato}' $ e r – a taxa de remoção ou recuperação.

$$\frac{dS(t)}{dt} = -\beta S(t)I(t)$$
$$\frac{dI(t)}{dt} = \beta S(t)I(t) - rI(t)$$
$$\frac{dR(t)}{dt} = rI(t)$$

Gostaríamos de identificar quais parâmetros β e r resultam num melhor ajuste do modelo para os dados de **S**,**I** e **R**

[6]:
```
def SIRmodel(y, t, Beta,r):
    S, I, R = y
    Sdot = -(Beta * S * I)
    Idot = (Beta * S * I)  - r * I
    Rdot = r * I
    return Sdot, Idot, Rdot

# Resolução da simulação - Escala temporal (dias)
```

3.3.3 Obtendo $y_s(\theta, k) = [S\ I\ R]$

O trecho a seguir retorna os valores sintetizados $y_s(\theta, k) = [S\ I\ R]$ representa o dado sintetizado a partir de um modelo sintetizado para uma determinada amostra k e θ representa o vetor ed parâmetros $\theta = [\beta\ \ r]^T$. A partir de uma condição inicial y_0.

[7]:
```
def SIRsim(y0,t,theta):
    Beta = theta[0]
    r = theta[1]
    ret = integrate.odeint(SIRmodel,y0,t,args=(Beta,r))
    S, I, R = ret.T
    return S, I, R
```

3.3.4 Condições inicias - y_0 e θ_0

[8]:
```
# Tamanho da populção - N
N = 500

# Valores iniciais
I0, R0 = 1, 0
S0 = N - I0

# Vetor de condições iniciais

y0 = S0, I0, R0

# Beta -  taxa de contato,
# r - taxa média de recuperação (in 1/dia).

theta0 = [1e-4,1e-2] # valores iniciais

# Definição do conjunto de equações diferencias não lineares que formam o modelo.

t = np.linspace(0, 1000, 1000)
```

3.3.5 Estimação de parâmetros

Para estimarmos os parâmetros do modelo β e **r**, vamos utilizar inicialmente o método de mínimos quadrados. Podemos então formular o problema a partir da Equação abaixo. Na Equação $y_m(k)$ representa o dado real em cada amostra k; $y_s(\theta, k)$ representa o **valor estimado** a partir da simulação do modelo para uma determinada amostra k e θ representa o vetor ed parâmetros $\theta = [\beta\ \ r]^T$.

$$min_\theta = \sum_{k=1}^{K} (y_m(k) - y_s(\theta, k))^2$$

A equação formula a pergunta: quais os valores de $beta$ e r que minizam o erro quadrático quando comparados com os dados reais.

[9]:
```
def ErroQuadratico(Sd,Id,Rd,y0,t,theta0):
    """ function to pass to optimize.leastsq
        The routine will square and sum the values returned by
        this function"""
    [S,I,R] = SIRsim(y0,t,theta0)
    erroS = S - Sd
    erroI = I - Id
    erroR = R - Rd
    EQ = np.concatenate([erroI,erroR])
    return EQ

def objetivo(p):
    return ErroQuadratico(Sd,Id,Rd,y0,t,p)
```

Minimização da função custo

[10]:
```
(c,kvg) = optimize.leastsq(objetivo,theta0)
print(c)
```

```
[9.93827391e-05 9.96628326e-03]
```

Visualização

[11]:
```
[Sa,Ia,Ra] = SIRsim(y0,t,c)

# Visualizando a evolução da Epidemia - S(t), I(t) e R(t)
p.line(t, Sa, legend_label="Suscetíveis", color="#f9bd3d", line_width=3)
p.line(t, Ia, legend_label="Infectados",  color="#f4193c", line_width=3)
p.line(t, Ra, legend_label="Removidos",   color="#45c83a", line_width=3)

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.3.6 Referências

- https://cmdlinetips.com/2018/02/how-to-subset-pandas-dataframe-based-on-values-of-a-column/
- https://cmdlinetips.com/2018/04/how-to-concatenate-arrays-in-numpy/
- https://stackoverflow.com/questions/11278836/fitting-data-to-system-of-odes-using-python-via-scipy-numpy

3.4 SIR Reino Unido 1944-1964

[1]:
```
import pylab as pp
import numpy as np
import pandas as pd
import scipy.signal as scs

from scipy import integrate, interpolate, optimize
from scipy.integrate import odeint
from pylab import *
import netCDF4

from bokeh.models   import ColumnDataSource, RangeTool, LinearAxis, Range1d
from bokeh.palettes import brewer, Inferno10
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook

output_notebook()
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

3.4.1 Dados Reais

Dados referentes a cidade de Londres entre 1944 e 1964. População de aproximadamente 2.5 milhões de habitantes na época.

[2]:
```
# Lendo o arquivo de dados no formato 'filename.csv'
data = pd.read_csv("./PG_IMT/DadosEpidemia/UKCities/London.csv")
# Preview das cinco primeiras linhas
data.head()

s_array = data[["cases", "births", "pop","time"]].to_numpy()

Id = s_array[:,0]
Bd = s_array[:,1]
Sd = s_array[:,2]
t  = s_array[:,3]
```

Visualizando os dados

[3]:
```
TOOLS="zoom_in,zoom_out,save"
p1 = figure(tools=TOOLS, plot_width=600, plot_height=300)
p2 = figure(tools=TOOLS, plot_width=600, plot_height=300)
p3 = figure(tools=TOOLS, plot_width=600, plot_height=300)

p1.line(data["time"], data["births"], line_width=4, color="#8e44ad", line_alpha=0.9)
```

(continues on next page)

(continued from previous page)

```
p1.grid.grid_line_alpha = 0
p1.ygrid.band_fill_color = "olive"
p1.ygrid.band_fill_alpha = 0.1
p1.yaxis.axis_label = "Nascimentos"
p1.xaxis.axis_label = "Data"

p2.line(data["time"], data["pop"], line_width=4, color="#8e44ad", line_alpha=0.9)

p2.grid.grid_line_alpha = 0
p2.ygrid.band_fill_color = "olive"
p2.ygrid.band_fill_alpha = 0.1
p2.yaxis.axis_label = "População"
p2.xaxis.axis_label = "Data"

p3.line(data["time"], data["cases"], line_width=4, color="#8e44ad", line_alpha=0.9)

p3.grid.grid_line_alpha = 0
p3.ygrid.band_fill_color = "olive"
p3.ygrid.band_fill_alpha = 0.1
p3.yaxis.axis_label = "Casos"
p3.xaxis.axis_label = "Data"

show(column(p1,p2,p3))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.4.2 Selecionando um ano específico para análise

[4]:

```
selected_year = 1948

is_1948 = (data['time'].astype(int) == selected_year)
LD48 = data[is_1948]
LD48.head()

p = figure(tools=TOOLS, plot_width=600, plot_height=300)
p.line(LD48["time"], LD48["cases"], line_width=4, color="#8e44ad", line_alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Casos"
p.xaxis.axis_label = "Data"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.4.3 Os dados

Conhecemos o número de casos, os nascimentos e a população total. Podemos determinar os demais valores a partir da relação $S(t) + I(t) + R(t) = N$ onde **N** vamos considerar a população.

[5]:
```
s_array = LD48[["cases", "births", "pop","time"]].to_numpy()

Id = s_array[:,0]
Bd = s_array[:,1]
Sd = s_array[:,2] - Id + Bd
t  = s_array[:,3]
```

3.4.4 O problema

O conjunto de equações diferenciais que caracteriza o modelo é descrito abaixo. No modelo β - *representa a taxa de transmissão ou taxa efetiva de contato* e r - *a taxa de remoção ou recuperação*.

$$\begin{aligned} \frac{dS(t)}{dt} &= -\beta S(t)I(t) \\ \frac{dI(t)}{dt} &= \beta S(t)I(t) - \gamma I(t) \\ \frac{dR(t)}{dt} &= \gamma I(t) \end{aligned} \tag{3.1}$$

Gostaríamos de identificar quais parâmetros β e r resultam num melhor ajuste do modelo para os dados de **S**, **I** e **R**

[6]:
```
def SIRmodel(y, t, Beta, r):
    S, I = y
    Sdot = -Beta * S * I
    Idot = Beta * S * I  - r * I
    return Sdot, Idot

# Resolução da simulação - Escala temporal (dias)
```

3.4.5 Obtendo $y_s(\theta, k) = [S\ I\ R]$

O trecho a seguir retorna os valores sintetizados $y_s(\theta, k) = [S\ I\ R]$ representa o dado sintetizado a partir de um modelo sintetizado para uma determinada amostra k e θ representa o vetor ed parâmetros $\theta = [\beta\ \ r]^T$. A partir de uma condição inicial y_0.

[7]:
```
def SIRsim(y0, t, theta):
    Beta, r = theta[0], theta[1]
    ret = integrate.odeint(SIRmodel, y0, t, args=(Beta,r))
    S, I = ret.T
    return S, I
```

3.4.6 Condições inicias - y_0 e θ_0

[8]:
```
# Valores iniciais
I0 = Id[0]
S0 = Sd[0]

# Vetor de condições iniciais

y0 = S0, I0

# Beta -  taxa de contato
# r - taxa média de recuperação (in 1/dia).

# theta = [Beta, r]
theta0 = [1e-8, 1e-1] # valores iniciais

# Definição do conjunto de equações diferencias não lineares que formam o modelo.
t = (t - 1948) * 365
```

3.4.7 Estimação de parâmetros

Para estimarmos os parâmetros do modelo β e **r**, vamos utilizar inicialmente o método de mínimos quadrados. Podemos então formular o problema a partir da Equação abaixo. Na Equação $y_m(k)$ representa o dado real em cada amostra k; $y_s(\theta, k)$ representa o **valor estimado** a partir da simulação do modelo para uma determinada amostra k e θ representa o vetor ed parâmetros $\theta = [\beta \;\; r]^T$.

$$min_\theta = \sum_{k=1}^{K} (y_m(k) - y_s(\theta, k))^2$$

A equação formula a pergunta: quais os valores de $beta$ e r que minizam o erro quadrático quando comparados com os dados reais.

[9]:
```
def ErroQuadratico(Sd,Id,y0,t,theta0,w):
    """ function to pass to optimize.leastsq
        The routine will square and sum the values returned by
        this function"""
    [S,I] = SIRsim(y0,t,theta0)
    erroS = w[0] * (S - Sd)
    erroI = w[1] * (I - Id)
    EQ = np.concatenate([erroS,erroI])
    if sum(np.isnan(EQ)) >= 1:
        print("Error!!!")
    return EQ

def objetivo(p, S, I, y0, t, w):
    return ErroQuadratico(S,I,y0,t,p,w)
```

Minimização da função custo

[10]:
```
# Criação das ponderações dos erros
wS = max(Id) / max(Sd)
w = [wS, 1]

(c,kvg) = optimize.leastsq(objetivo, theta0, args=(Sd, Id, y0, t, w))
print(c)
```

```
[2.10765916e-08 6.83427510e-02]
```

Visualização

[11]:
```
[Sa,Ia] = SIRsim(y0,t,c)

p = figure(tools=TOOLS, y_range=(0,5000), plot_width=600, plot_height=400)

p.scatter(t, Sd, legend_label="Suscetíveis - dados", size=8, fill_color="#ffd885",␣
↪fill_alpha=0.7, line_alpha=0)
p.scatter(t, Id, legend_label="Infectados - dados", size=8, fill_color="#de425b",␣
↪fill_alpha=0.7, line_alpha=0)

p.line(t, Sa, legend_label="Suscetíveis", line_width=4, color="#f9bd3d", line_cap=
↪'round', line_alpha=0.9)
p.line(t, Ia, legend_label="Infectados", line_width=4, color="#f4193c", line_cap=
↪'round', line_alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Reamostrando os dados

[12]:
```
Sd_res, t_res = scs.resample(Sd, 365, t=t)
Id_res, t_res = scs.resample(Id, 365, t=t)

p = figure(tools=TOOLS, y_range=(0,4000), plot_width=600, plot_height=400)

p.scatter(t, Id, legend_label="Infectados", size=10, fill_color="#f4193c", fill_
↪alpha=0.6, line_alpha=0)
p.scatter(t_res, Id_res, legend_label="Infectados - Resample", size=4, fill_color="
↪#8e44ad", line_alpha=0)
```

(continues on next page)

(continued from previous page)

```
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Restimando os parâmetros

[13]:

```
(c, kvg) = optimize.leastsq(objetivo, theta0, args=(Sd_res, Id_res, y0, t_res, w))
print(c)

[Sa,Ia] = SIRsim(y0, t_res, c)

p.line(t_res, Ia, legend_label="Infectados - Modelo", line_width=4, color="#f4193c",␣
↪line_cap='round', line_alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

show(p)
```

```
[2.15762357e-07 6.89608748e-01]
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.4.8 Outros métodos de estimação

[14]:

```
import warnings
warnings.filterwarnings("error")

from scipy.optimize import dual_annealing, shgo, basinhopping, differential_evolution

def cost_function(theta0,Sd,Id,y0,t,w):
    """ function to pass to optimize.leastsq
        The routine will square and sum the values returned by
        this function"""
```

(continues on next page)

(continued from previous page)

```
    try:
        [S, I] = SIRsim(y0, t, theta0)
        erroS = (w[0]*(S - Sd)**2)
        erroI = (w[1]*(I - Id)**2)
        final_error = sum(erroI)
    except:
        final_error = 10**14
    return final_error


beta_approx = 1 / data["pop"].max()
r_approx = 1 / 365

x0 = [beta_approx, r_approx]
lower, upper = [x0[0]/10, x0[1]/10], [10*x0[0], 1000*x0[1]]
bounds = zip(lower, upper)

#res = dual_annealing(cost_function, x0=x0, maxiter=2000, bounds=list(bounds),␣
↪args=(Sd_res, Id_res, y0, t_res, w))
#res = shgo(cost_function, bounds=list(bounds), args=(Sd_res, Id_res, y0, t_res, w))
#res = basinhopping(cost_function, x0=x0, niter=200, minimizer_kwargs=dict(args=(Sd_
↪res, Id_res, y0, t_res, w)))

res = differential_evolution(cost_function, list(bounds), args=(Sd_res, Id_res, y0, t_
↪res, w))
res
```

[14]:

```
     fun: 43830917.444261
     jac: array([2.89188102e+17, 3.06168899e+07])
 message: 'Optimization terminated successfully.'
    nfev: 906
     nit: 26
 success: True
       x: array([2.16790744e-07, 6.92576449e-01])
```

[15]:

```
[Sa,Ia] = SIRsim(y0, t_res, res.x)

p = figure(tools=TOOLS, y_range=(0,4000), plot_width=600, plot_height=400)

p.scatter(t, Id, legend_label="Infectados", size=10, fill_color="#f4193c", fill_
↪alpha=0.6, line_alpha=0)
p.scatter(t_res, Id_res, legend_label="Infectados - Resample", size=4, fill_color="
↪#8e44ad", line_alpha=0)

p.line(t_res, Ia, legend_label="Infectados - Modelo", line_width=4, color="#f4193c",␣
↪line_cap='round', line_alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.4.9 Análise anual

[16]:
```
years = data["time"].astype(int).unique().tolist()

p = figure(tools=TOOLS+",hover",
           x_range=(min(years), max(years)),
           plot_width=600, plot_height=300)

p.line(data["time"], data["cases"],
       legend_label="Casos", line_width=4, color="#f4511e", line_cap='round', line_
↪alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"
p.toolbar.autohide = True

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Detecção dos períodos de contágio

[17]:
```
from PyAstronomy import pyasl

# Filtrando os dados para o processo de derivação
filt_cases = pyasl.smooth(data["cases"], 13, 'hamming')


# Incluindo os dados filtrados no plot
p.line(data["time"], filt_cases, line_dash="dotted",
       legend_label="Casos Filtrado", line_width=3, color="#8e44ad", line_cap='round',
↪ line_alpha=0.9)

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[18]:
```
cases_variation = np.diff(filt_cases) # Calculando a taxa de variação

threshold = np.std(cases_variation)   # Calculando o threshold baseado no desvio
                                      # padrão da taxa de variação
threshold_vec = [threshold for k in cases_variation]

p = figure(tools=TOOLS+",hover",
           plot_width=600, plot_height=300)

p.line(data["time"][:-1], cases_variation,
       legend_label="Derivada dos casos", line_width=4, color="#f4511e", line_cap=
↪'round', line_alpha=0.9)
p.line(data["time"][:-1], threshold_vec,
       legend_label="Threshold", line_width=4, color="#8e44ad", line_cap='round',␣
↪line_alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"
p.toolbar.autohide = True

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[19]:
```
def findEpidemyBreaks(cases, threshold_prop, cases_before=10):
    """
    """

    # Filtering the data
    filt_cases = pyasl.smooth(cases, 11, 'hamming')
    # Compute the derivative and standard deviation
    cases_variation = np.diff(filt_cases).tolist()
    threshold = threshold_prop * np.std(cases_variation)

    start_points, end_points = [], []
    in_epidemy, out_epidemy = False, False
    for k, value in enumerate(cases_variation):
        if not in_epidemy:
            # Check value
            if value > threshold:
                in_epidemy = True
                # Find the start point
                start_index = 0 if k-cases_before < 0 else k-cases_before
                window = [abs(v) for v in cases_variation[start_index:k]]
                ref_index = window.index(min(window))
                start_points.append(k - (cases_before - ref_index))
        else:
            check_1 = (cases_variation[k-1] < 0)
            check_2 = (value >= 0)
```

(continues on next page)

(continued from previous page)

```
            if check_1 and check_2:
                in_epidemy = False
                end_points.append(k)
    return start_points, end_points
```

[20]:

```
start, end = findEpidemyBreaks(data["cases"], 1, 12)

p = figure(tools=TOOLS+",hover",
            x_range=(min(years), max(years)),
            y_range=(0, 8000),
            plot_width=600, plot_height=400)

p.line(data["time"], data["cases"],
       legend_label="Casos", line_width=4, color="#f4511e", line_cap='round', line_
↪alpha=0.9)



windowed_data = { "I":[], "S":[], "B":[], "t":[] }
for s, e in zip(start, end):
    windowed_data["B"].append(data["births"][s:e].to_numpy())
    windowed_data["S"].append(data["pop"][s:e].to_numpy())
    windowed_data["I"].append(data["cases"][s:e].to_numpy())
    windowed_data["t"].append(data["time"][s:e].to_numpy())

    p.line(windowed_data["t"][-1], windowed_data["I"][-1],
           legend_label="Casos na Epidemia", line_width=3, color="#8e44ad", line_cap=
↪'round', line_alpha=0.9)

    p.line([data["time"][s], data["time"][s]], [0, 10000], line_dash="dotted",
           legend_label="Início da Epidemia", line_width=1, color="#455a64", line_cap=
↪'round', line_alpha=0.9)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Ano"
p.toolbar.autohide = True


index = 5
days = (windowed_data["t"][index] - windowed_data["t"][index][0]) * 365

p1 = figure(tools=TOOLS+",hover",
            plot_width=600, plot_height=400)

p1.line(days, windowed_data["I"][index],
       legend_label="Casos", line_width=4, color="#8e44ad", line_cap='round', line_
↪alpha=0.9)

p1.grid.grid_line_alpha = 0
p1.ygrid.band_fill_color = "olive"
p1.ygrid.band_fill_alpha = 0.1
```

(continues on next page)

(continued from previous page)

```
p1.yaxis.axis_label = "Indivíduos"
p1.xaxis.axis_label = "Dias"
p1.toolbar.autohide = True

show(column(p,p1))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[21]:

```
final_data = {
    "data": {
        "original": [],
        "resampled": [],
        "simulated": []
    },
    "pars": {
        "beta": [],
        "r": []
    }
}

zipped_data = zip(
    windowed_data["S"],
    windowed_data["I"],
    windowed_data["B"],
    windowed_data["t"])

for S, I, B, t in zipped_data:
    year_ref = t[0]
    S = S - I + B # Calculando os suscetiveis
    t = (t - year_ref) * 365 # Tempo em dias
    # Condições iniciais
    y0 = S[0], I[0]
    theta0 = [1e-8, 1e-1]
    # Criando a ponderação
    # dos erros
    wS = max(I) / max(S)
    w = [wS, 1]
    # Reamostrando os dados
    Sd_res, t_res = scs.resample(S, int(t[-1]), t=t)
    Id_res, t_res = scs.resample(I, int(t[-1]), t=t)
    # Estimando os par^ametros
    bounds = zip([x0[0]/10, x0[1]/10], [10*x0[0], 1000*x0[1]])
    res = differential_evolution(cost_function, list(bounds), args=(Sd_res, Id_res,␣
↪y0, t_res, w))
    # Simulando os dados
    [Sa, Ia] = SIRsim(y0, t_res, res.x)
    # Save the year data
    final_data["data"]["original"].append(
        { "I": I, "B": B, "S": S, "t": t/365 + year_ref })
    final_data["data"]["resampled"].append(
        {"I": Id_res, "S": Sd_res, "t": t_res/365 + year_ref})
    final_data["data"]["simulated"].append(
```

(continues on next page)

(continued from previous page)

```
        {"I": Ia, "S": Sa, "t": t_res/365 + year_ref})
    final_data["pars"]["beta"].append( res.x[0] )
    final_data["pars"]["r"].append( res.x[1] )
```

[22]:

```
r = final_data["pars"]["r"]
beta = final_data["pars"]["beta"]
years = [int(t[0]) for t in windowed_data["t"]]



p = figure(tools=TOOLS+",hover",
           y_range=(min(beta), max(beta)),
           plot_width=600, plot_height=300)


p.line(years, beta,
       legend_label="beta", line_width=4, color="#c2185b", line_cap='round', line_
↪alpha=0.9)

p.extra_y_ranges = {"r_axis": Range1d(start=min(r), end=max(r))}
p.add_layout(LinearAxis(y_range_name="r_axis"), 'left')

p.line(years, r, y_range_name="r_axis", line_dash='dashed',
       legend_label="r", line_width=3, color="#8e44ad", line_cap='round', line_
↪alpha=0.9)


p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.xaxis.axis_label = "Ano"
p.toolbar.autohide = True



p1 = figure(tools=TOOLS+",hover",
            x_range=p.x_range,
            plot_width=600, plot_height=300)


p1.line(data["time"], data["cases"],
        legend_label="Casos", line_width=2, color="#f4511e", line_cap='round', line_
↪alpha=0.9)

for dataset in final_data["data"]["original"]:
    p1.line(dataset["t"], dataset["I"],
            legend_label="Casos", line_width=4, color="#f4511e", line_cap='round',␣
↪line_alpha=0.9)

for dataset in final_data["data"]["simulated"]:
    p1.line(dataset["t"], dataset["I"], line_dash='dashed',
            legend_label="Estimado", line_width=3, color="#0288d1", line_cap='round',␣
↪line_alpha=0.9)
```

(continues on next page)

(continued from previous page)

```
p1.grid.grid_line_alpha = 0
p1.ygrid.band_fill_color = "olive"
p1.ygrid.band_fill_alpha = 0.1
p1.yaxis.axis_label = "Indivíduos"
p1.xaxis.axis_label = "Ano"
p1.toolbar.autohide = True



show(column(p,p1))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[23]:

```
source = ColumnDataSource(data=dict(date=data["time"].to_numpy(), value=data["cases"].
↪to_numpy()))

p = figure(plot_height=300, plot_width=600, tools="xpan", toolbar_location=None,
           x_axis_location="above", x_range=(data["time"][5], data["time"][105]))

p.line('date', 'value', source=source,
       legend_label="Casos", line_width=2, color="#f4511e", line_cap='round', line_
↪alpha=0.9)

for dataset in final_data["data"]["resampled"]:
    rsource = ColumnDataSource(data=dict(date=dataset["t"], value=dataset["I"]))
    p.scatter('date', 'value', source=rsource,
       legend_label="Casos reamostrados", fill_color="#8e44ad", size=4, line_alpha=0)

p.yaxis.axis_label = 'Indivíduos'
p.grid.grid_line_alpha = 0
p.xgrid.band_fill_color = "navy"
p.xgrid.band_fill_alpha = 0.1
p.toolbar.autohide = True

select = figure(title="Drag the middle and edges of the selection box to change the␣
↪range above",
                plot_height=130, plot_width=600, y_range=p.y_range,
                y_axis_type=None, tools="", toolbar_location=None)

range_tool = RangeTool(x_range=p.x_range)
range_tool.overlay.fill_color = "navy"
range_tool.overlay.fill_alpha = 0.2

select.line('date', 'value', source=source,
            line_width=2, color="#f4511e", line_cap='round', line_alpha=0.9)
select.ygrid.grid_line_color = None
select.add_tools(range_tool)
select.toolbar.active_multi = range_tool
select.xaxis.axis_label = 'Ano'

show(column(p, select))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[24]:

```
p1 = figure(plot_height=300, plot_width=600, tools="xpan", toolbar_location=None,
            x_axis_location="above", x_range=p.x_range)

p1.line('date', 'value', source=source,
       legend_label="Casos", line_width=2, color="#f4511e", line_cap='round', line_
↪alpha=0.9)

for dataset in final_data["data"]["simulated"]:
    psource = ColumnDataSource(data=dict(date=dataset["t"], value=dataset["I"]))
    p1.line('date', 'value', source=psource, line_dash="dashed",
           legend_label="Modelo estimado", color="#0288d1",
           line_width=4, line_cap='round', line_alpha=0.9)

    window_bound_lower = [dataset["t"][0], dataset["t"][0]]
    window_bound_upper = [dataset["t"][-1], dataset["t"][-1]]
    window_limits = [0, 10000]

    p.line(window_bound_lower, window_limits, line_dash="dashed",
           legend_label="Janela da epidemia", color="#455a64",
           line_width=2, line_cap='round', line_alpha=0.2)
    p.line(window_bound_upper, window_limits, line_dash="dashed",
           legend_label="Janela da epidemia", color="#455a64",
           line_width=2, line_cap='round', line_alpha=0.2)
    p1.line(window_bound_lower, window_limits, line_dash="dashed",
           legend_label="Janela da epidemia", color="#455a64",
           line_width=2, line_cap='round', line_alpha=0.2)
    p1.line(window_bound_upper, window_limits, line_dash="dashed",
           legend_label="Janela da epidemia", color="#455a64",
           line_width=2, line_cap='round', line_alpha=0.2)

p1.xaxis.axis_line_alpha = 0
p1.xaxis.major_label_text_color = None
p1.xaxis.major_tick_line_color = None
p1.xaxis.minor_tick_line_color = None
p1.yaxis.axis_label = 'Indivíduos'
p1.grid.grid_line_alpha = 0
p1.xgrid.band_fill_color = "navy"
p1.xgrid.band_fill_alpha = 0.1
p1.toolbar.autohide = True

show(column(p, p1, select))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Salvando as variáveis

[25]:
```
import pickle

save_variable = {
    "data" : data,
    "final" : final_data
}

with open('UK_estimated_data.pickle', 'wb') as handle:
    pickle.dump(save_variable, handle)
```

3.5 SIR Reino Unido Modelos Espaciais

3.5.1 Os dados

Lendo os dados de todas as cidades disponíveis:

[1]:
```
import os
import pandas as pd


path = "./PG_IMT/DadosEpidemia/UKCities/"
files = os.listdir(path)

dataframes = []
for file in files:
    data = pd.read_csv(path + file).to_dict()
    keys = [k for k in data.keys()]
    size = len(data[keys[0]])
    # Create a column for the city name
    city = [file[:-4] for n in range(size)]
    data["city"] = dict(zip(range(size), city))
    dataframes.append(pd.DataFrame(data))

data = pd.concat(dataframes, ignore_index=True)
data = data.dropna()
data.head()
```

[1]:

	time	cases	births	pop	city
0	1944.016427	0	27.269231	28350.000000	Bedwellty
1	1944.054757	0	27.148291	28339.031079	Bedwellty
2	1944.093087	0	27.027352	28328.062157	Bedwellty
3	1944.131417	0	26.906413	28317.093236	Bedwellty
4	1944.169747	1	26.785473	28306.124314	Bedwellty

3.5.2 Determinando os parâmetros - SIR

Nesta seção, iremos utilizar a biblioteca `models`, que concentra todas as funções dos modelos SIR desenvolvidos nas seções anteriores.

Para uma única cidade

[2]:
```
from models import *

# Getting the data for a particular city
dataset = data.where(data["city"] == "London").dropna()

# Creating the data for trainning
N = int(dataset["pop"].mean())
B = dataset["births"].to_numpy()
I = dataset["cases"].to_numpy()
S = dataset["pop"].to_numpy() - I + B
t = dataset["time"].to_numpy()

# Creating the SIR model
model = ss.SIR(pop=N, focus=["I"])

# Fitting the model on data
fit_data = model.fit_multiple(S, I, B, t,
                              r_sens=[1000,10], beta_sens=[100,100])

# Plotting the model results
fig = model.result_summary(
      out_plot=True,
      plot_size=[600,400])


from bokeh.plotting import show

show(fig)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

```
Windows starting at:  [17, 46, 98, 124, 174, 222, 276, 331, 383, 429, 481]
Windows ending at:    [47, 72, 124, 153, 206, 260, 311, 361, 415, 468, 519]
Window start cases:   [41.0, 88.0, 113.0, 157.0, 308.0, 1521.0, 249.0, 183.0, 194.0,␣
↪256.0, 141.0]
New iter:::  1
          ├─ S(0) =   2550525.415201801   I(0) =   41.0
          ├─ beta =   1   r =   0.1
          ├─ beta bound =   0.01  =  10
          ├─ r bound =   0.001  =  1.0
          └─ Defined at:  0.7605155319336778  =  0.5771663611353828

New iter:::  2
          ├─ S(0) =   2949891.618829986   I(0) =   88.0
          ├─ beta =   1   r =   0.1
          ├─ beta bound =   0.01  =  10
          ├─ r bound =   0.001  =  1.0
          └─ Defined at:  0.652527725167639  =  0.5784638670213627

New iter:::  3
          ├─ S(0) =   3302261.375966816   I(0) =   113.0
          ├─ beta =   1   r =   0.1
```

(continues on next page)

(continued from previous page)

```
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.5930110943411019  −  0.5853040022414066

New iter:::  4
          ├─ S(0) −   3360609.294403033   I(0) −   157.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.5510788084784277  −  0.5541593857111112

New iter:::  5
          ├─ S(0) −   3379912.492617072   I(0) −   308.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.4418921319691951  −  0.437900484416566

New iter:::  6
          ├─ S(0) −   3361115.5108282953   I(0) −   1521.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.2746852290519879  −  0.27916031863574764

New iter:::  7
          ├─ S(0) −   3321231.492546763   I(0) −   249.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.36599758110134534  −  0.35877292971764685

New iter:::  8
          ├─ S(0) −   3272868.3492476433   I(0) −   183.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.4917754139555685  −  0.4781307010467401

New iter:::  9
          ├─ S(0) −   3226812.132892697   I(0) −   194.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.5780416048451077  −  0.5570092885781249

New iter:::  10
          ├─ S(0) −   3199397.624806639   I(0) −   256.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
          ├─ r bound −   0.001  −  1.0
          └─ Defined at:  0.3327289237627916  −  0.315264257523117

New iter:::  11
          ├─ S(0) −   3185602.2749261744   I(0) −   141.0
          ├─ beta −   1   r −   0.1
          ├─ beta bound −   0.01  −  10
```

(continues on next page)

(continued from previous page)

```
        ├─ r bound -   0.001   -  1.0
        └─ Defined at:  0.4244705235353749  -  0.4056067055948321
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[3]:

[3]: 3212255.4403247973

Estimando para todas cidades

[ ]:

```
from models import *

# Define the city names
cities = data["city"].unique()

par_data = dict()
for i, city in enumerate(cities):

    # Getting the data for a particular city
    dataset = data.where(data["city"] == city).dropna()

    # Creating the data for trainning
    N = dataset["pop"].mean()
    B = dataset["births"].to_numpy()
    I = dataset["cases"].to_numpy()
    S = dataset["pop"].to_numpy() - I + B
    t = dataset["time"].to_numpy()

    # Creating the SIR model
    model = ss.SIR(verbose=False)

    # Fitting the model on data
    par_data[city] = model.fit_multiple(S,I,B,t, out_type=1)

    # Save the summary on folder
    model.result_summary(
      save_results=True,
      plot_size=[700,500],
      folder_path="./estimation_summaries/",
      file_name=city+"_summary.png")

    print("Finished - ", city, " - ", i+1, " of ", len(cities))
```

3.5.3 Obtendo informações de Lat e Long

[ ]:
```
from geopy.geocoders import Nominatim

geolocator = Nominatim(user_agent="UK_EpidemicModels_App")

for city in cities:
    city_name = city.replace(".", " ")
    location = geolocator.geocode(city_name)
    par_data[city]["lat"] = location[-1][0]
    par_data[city]["lon"] = location[-1][1]
    print("- Located: ", location[-1], "  ", city_name)
```

3.5.4 Correlação dos modelos

Interpolando os modelos estimados

[ ]:
```
import numpy as np
import scipy.signal as scs
from scipy.interpolate import Rbf, InterpolatedUnivariateSpline

first_year = round(min([par_data[city]["time"][0] for city in cities]))
last_year = round(max([par_data[city]["time"][-1] for city in cities]))
points = int(last_year - first_year) - 1

data_struc = {
    "lat": [],
    "lon": [],
    "city": [],
    "year": [],
    "beta": [],
    "r": [],
}

points = 30
time = np.linspace(first_year, last_year, points)

for city in cities:

    # Selecionando os par^ametros estimados
    beta_res = par_data[city]["pars"]["beta"]
    r_res = par_data[city]["pars"]["r"]
    year = par_data[city]["time"]

    # Interpolando com reamostragem
    #beta_res, t_res = scs.resample(beta, points, t=year-year[0])
    #r_res, t_res = scs.resample(r, points, t=year-year[0])
    #t_res = t_res.astype(int)

    # Interpolando com Univariate Splines
    #beta_rbf = InterpolatedUnivariateSpline(year, beta_res)
    #r_rbf = InterpolatedUnivariateSpline(year, r_res)

    # Interpolando com Radial Basis Functions
    beta_rbf = Rbf(year, beta_res, function='gaussian', smooth=2)
```

(continues on next page)

(continued from previous page)

```
    r_rbf = Rbf(year, r_res, function='gaussian', smooth=2)

    # Salvando do dicionário cada campo
    data_struc["lat"] += [par_data[city]["lat"] for k in range(points)]
    data_struc["lon"] += [par_data[city]["lon"] for k in range(points)]
    data_struc["city"] += [city for k in range(points)]
    data_struc["beta"] += beta_rbf(time).tolist()
    data_struc["r"]    += r_rbf(time).tolist()
    data_struc["year"] += time.tolist()

# Criando um data frame
df = pd.DataFrame(data_struc)
```

Visualizando a interpolação

[ ]:

```
# Creating the parameters plot
#

from bokeh.models   import Legend, ColumnDataSource, RangeTool, LinearAxis, Range1d
from bokeh.palettes import brewer, Inferno256
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook


p_beta = figure(
    title="Beta Parameter",
    y_axis_type="log",
    plot_width=700,
    plot_height=500
)

p_r = figure(
    title="R Parameter",
    y_axis_type="log",
    plot_width=700,
    plot_height=500
)

legend_it, legend_it_r = [], []
for i, city in enumerate(cities):

    color = Inferno256[int((i/len(cities))*256)]

    df_filt = df.where(df["city"]==city).dropna()

    c = p_beta.line(
        df_filt["year"].to_list(),
        abs(df_filt["beta"]),
        line_width=4,
        line_cap="round",
        color=color
```

(continues on next page)

(continued from previous page)

```
    )

    cr = p_r.line(
        df_filt["year"],
        abs(df_filt["r"]),
        line_width=4,
        line_cap="round",
        color=color
    )

    legend_it.append((city, [c]))
    legend_it_r.append((city, [cr]))

p_beta.grid.grid_line_alpha = 0
p_beta.ygrid.band_fill_color = "olive"
p_beta.ygrid.band_fill_alpha = 0.1
p_beta.xaxis.axis_label = "Ano"

legend = Legend(items=legend_it, location=(0, -10))
legend.click_policy="mute"
p_beta.add_layout(legend, "right")
p_beta.legend.click_policy="hide"

p_r.grid.grid_line_alpha = 0
p_r.ygrid.band_fill_color = "olive"
p_r.ygrid.band_fill_alpha = 0.1
p_r.xaxis.axis_label = "Ano"

legend = Legend(items=legend_it_r, location=(0, -10))
legend.click_policy="mute"
p_r.add_layout(legend, "right")
p_r.legend.click_policy="hide"

show(column(p_beta, p_r))
```

Criando os data frames para matrix de correlação

[ ]:

```
r_data = {}
beta_data = {}

for city in cities:

    df_filt = df.where(df["city"]==city).dropna()

    data_dict = dict()
    zipped_data = zip(
        df_filt["year"].to_list(),
        df_filt["beta"].to_list())
    for y, b in zipped_data:
        data_dict[y] = b
    beta_data[city] = data_dict

    data_dict = dict()
```

(continues on next page)

(continued from previous page)

```
    zipped_data = zip(
        df_filt["year"].to_list(),
        df_filt["r"].to_list())
    for y, r in zipped_data:
        data_dict[y] = r
    r_data[city] = data_dict

r_df = pd.DataFrame(r_data)
r_df = r_df.sort_index()
r_df.index = (1000 * r_df.index.to_numpy()).astype(int)

beta_df = pd.DataFrame(beta_data)
beta_df = beta_df.sort_index()
beta_df.index = (1000 * beta_df.index.to_numpy()).astype(int)
beta_df.head()
```

Correlação do parâmetro β

[ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation Matrix Heatmap
f, ax = plt.subplots(figsize=(18, 15))

beta_corr = beta_df.corr()
hm = sns.heatmap(round(beta_corr,2),
                 annot=True,
                 ax=ax,
                 cmap="coolwarm",
                 fmt='.2f',
                 linewidths=.05)

f.subplots_adjust(top=0.93)

t = f.suptitle('Beta Parameter - Features Correlation Heatmap', fontsize=14)
```

Correlação do parâmetro r

[ ]:

```
# Correlation Matrix Heatmap
f, ax = plt.subplots(figsize=(18, 15))

r_corr = r_df.corr()
hm = sns.heatmap(round(r_corr,2),
                 annot=True,
                 ax=ax,
                 cmap="coolwarm",
                 fmt='.2f',
                 linewidths=.05)
```

(continues on next page)

(continued from previous page)

```
f.subplots_adjust(top=0.93)

t = f.suptitle('r Parameter - Features Correlation Heatmap', fontsize=14)
```

Correlação do modelo

[ ]:
```
f, ax = plt.subplots(figsize=(18, 15))

model_corr = 0.5 * ( r_corr +  beta_corr )
hm = sns.heatmap(round(model_corr,2),
                 annot=True,
                 ax=ax,
                 cmap="coolwarm",
                 fmt='.2f',
                 linewidths=.05)

f.subplots_adjust(top=0.93)

t = f.suptitle('Model - Features Correlation Heatmap', fontsize=14)
```

3.6 SIR Parâmetros variantes

3.6.1 Os Dados

[1]:
```
import pandas as pd

# reading csv file
BRdata = pd.read_csv("./PG_IMT/DadosEpidemia/CoVid19.csv")
BRdata.head()
```

[1]:
```
         date country  state    city  newDeaths  deaths  newCases  totalCases  \
0  2020-02-25  Brazil     SP  TOTAL          0       0         1           1
1  2020-02-25  Brazil  TOTAL  TOTAL          0       0         1           1
2  2020-02-26  Brazil     SP  TOTAL          0       0         0           1
3  2020-02-26  Brazil  TOTAL  TOTAL          0       0         0           1
4  2020-02-27  Brazil     SP  TOTAL          0       0         0           1

   deathsMS  totalCasesMS  deaths_per_100k_inhabitants  \
0         0             0                          0.0
1         0             0                          0.0
2         0             1                          0.0
3         0             1                          0.0
4         0             1                          0.0

   totalCases_per_100k_inhabitants  deaths_by_totalCases  recovered  suspects  \
0                          0.00218                   0.0        NaN       NaN
1                          0.00048                   0.0        NaN       NaN
```

(continues on next page)

(continued from previous page)

```
2                          0.00218                   0.0        NaN        NaN
3                          0.00048                   0.0        NaN        NaN
4                          0.00218                   0.0        NaN        NaN

   tests  tests_per_100k_inhabitants
0    NaN                         NaN
1    NaN                         NaN
2    NaN                         NaN
3    NaN                         NaN
4    NaN                         NaN
```

Filtrando e limpando os dados

[2]:
```
is_SP =  BRdata['state']=='TOTAL'
SPdata = BRdata[is_SP]

# SPdata = SPdata.fillna(0)

SPLimpo = SPdata[SPdata.recovered.notnull()]
SPLimpo.head()
```

[2]:
```
           date country  state   city  newDeaths  deaths  newCases  \
321  2020-03-23  Brazil  TOTAL  TOTAL          9      34       358
349  2020-03-24  Brazil  TOTAL  TOTAL         13      47       303
377  2020-03-25  Brazil  TOTAL  TOTAL         12      59       311
405  2020-03-26  Brazil  TOTAL  TOTAL         18      77       424
433  2020-03-27  Brazil  TOTAL  TOTAL         16      93       486

     totalCases  deathsMS  totalCasesMS  deaths_per_100k_inhabitants  \
321        1952        34          1891                      0.01618
349        2255        46          2201                      0.02237
377        2566        57          2433                      0.02808
405        2990        77          2915                      0.03664
433        3476        92          3417                      0.04425

     totalCases_per_100k_inhabitants  deaths_by_totalCases  recovered  \
321                          0.92887               0.01742        8.0
349                          1.07306               0.02084       20.0
377                          1.22105               0.02299       27.0
405                          1.42281               0.02575       42.0
433                          1.65408               0.02675       42.0

     suspects  tests  tests_per_100k_inhabitants
321   15867.0    NaN                         NaN
349   17700.0    NaN                         NaN
377   27227.0    NaN                         NaN
405   48793.0    NaN                         NaN
433   50684.0    NaN                         NaN
```

[3]:
```
from datetime import datetime

first_date = SPLimpo["date"].iloc[0]
```

(continues on next page)

(continued from previous page)

```
first_date = datetime.fromisoformat(first_date)

SPsir = SPLimpo[["totalCases", "deaths", "recovered"]].to_numpy()

SPsir[:,0:]
SPI = SPsir[:,0]
auxM = SPsir[:,1]
SPM = auxM
auxR = SPsir[:,2]
SPR = auxR
```

3.6.2 Visualizando a evolução

[4]:

```
from bokeh.models   import Legend, ColumnDataSource, RangeTool, LinearAxis, Range1d
from bokeh.palettes import brewer, Inferno256
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook

output_notebook()

from datetime import timedelta

# Criando o vetor de tempo
t_num = range(len(SPM))
date_vec = [ first_date + timedelta(days=k) for k in t_num]


# Criando a figura
p = figure(
    tools="hover",
    title="Evolução do COVID",
    x_axis_type='datetime',
    y_axis_type="log",
    plot_width=650,
    plot_height=500
)

# Incluindo os dados de mortes
p.line(
    date_vec, SPM,
    legend_label="Mortes",
    line_width=4,
    line_cap="round",
    color="#de425b"
)

# Incluindo os dados de infectados
p.line(
    date_vec, SPI,
    legend_label="Infectados",
    line_width=4,
    line_cap="round",
```

(continues on next page)

(continued from previous page)

```
    color="#ffd885"
)

# Incluindo os dados de recuperados
p.line(
    date_vec, SPR,
    legend_label="Removidos",
    line_width=4,
    line_cap="round",
    color="#99d594"
)


p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"
p.legend.click_policy="hide"
p.legend.location = "bottom_right"

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.6.3 Análise de um modelo com β variável no tempo

Uma outra forma de abordar a modelagme de dados é assumindo que a taxa de transmissão β é variante no tempo. Vamos, neste sentido, utilizar a abordagem elaborada por *Mark Pollicott, Hao Wang & Howard (Howie) Weiss em 2012 - Extracting the time-dependent transmission rate from infection data via solution of an inverse ODE problem*

Reescrevendo o conjunto de equações diferenciais que caracteriza o modelo é descrito abaixo. No modelo $:nbsphinx-math:*beta‘(t) - :nbsphinx-math:*text{representa a taxa de transmissão ou taxa efetiva de contato}‘ $ e $r -$ a taxa de remoção ou recuperação.

$$\frac{dS(t)}{dt} = -\beta(\mathbf{t})S(t)I(t)$$
$$\frac{dI(t)}{dt} = \beta(\mathbf{t})S(t)I(t) - rI(t)$$
$$\frac{dR(t)}{dt} = rI(t)$$

Gostaríamos de identificar quais parâmetros $\beta(t)$ e r resultam num melhor ajuste do modelo para os dados de **S**,**I** e **R**

Proposta de algoritmo para extração do $\beta(t)$ a partir dos dados

1. Interpolar os dados para suvizar eventuais variações gerando uma função $f(t)$

2. **Verificar a seguinte condição**: $\frac{f'(t)}{f(t)} > -r$. Neste algoritmo a taza de romação r é considerada constante.
3. Determinando a função $p(t) = \frac{f''(t)f(t) - f'^2(t)}{f(t)(f'(t) + rf(t))}$.
4. Escolhendo um valor inicial para $\beta(0)$ e determinando a integral $P(t) = \int_0^t p(\tau)d\tau$.
5. **Verificar a seguinte condição:** $\beta(0) < \frac{1}{\int_0^t e^{P(s)} f(s) ds}$.
6. O parâmetro $\beta(\mathbf{t})$ pode ser calculado por: $\beta(t) = \frac{1}{\frac{e^{-P(t)}}{\beta(0)} - e^{-P(t)} \int_0^t e^{P(s)} f(s) ds}$.

1) Interpolar os dados para suvizar eventuais variações gerando uma função $f(t)$

[5]:
```
import numpy as np
from scipy.interpolate import Rbf, UnivariateSpline

# Generating weights for polynomial function with degree =2
weights = np.polyfit(t_num, SPI, 2)
rbf = Rbf(t_num, SPI, function='gaussian')
spline = UnivariateSpline(t_num, SPI)
print(weights)


# Generating model with the given weights
model = np.poly1d(weights)

t_sim = np.linspace(t_num[0], t_num[-1], 100)

# Prediction on validation set
pred = model(t_sim)
pred_rbf = rbf(t_sim)
pred_spline = spline(t_sim)

# We will plot the graph for 70 observations only

# Criando a figura
p = figure(
    tools="hover",
    title="Dados interpolados",
    x_axis_type='datetime',
#     y_axis_type="log",
    plot_width=650,
    plot_height=500
)

# Incluindo os dados de mortes
p.line(
    t_num, SPI,
    legend_label="Medidas",
    line_width=4,
    line_cap="round",
    color="#de425b"
)

# Incluindo os dados de infectados
p.line(
```

(continues on next page)

(continued from previous page)

```
    t_sim, pred,
    legend_label="Interpolação",
    line_width=4,
    line_cap="round",
    color="#1e88e5"
)

p.line(
    t_sim, pred_rbf,
    legend_label="Interpolação - RBF",
    line_width=3,
    line_dash="dashed",
    line_cap="round",
    color="#512da8"
)

p.line(
    t_sim, pred_spline,
    legend_label="Interpolação - Spline",
    line_width=3,
    line_dash="dashed",
    line_cap="round",
    color="#00796b"
)

p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"
p.legend.click_policy="hide"
p.legend.location = "bottom_right"

show(p)
```

```
[  67.47775863 -501.31086778 4785.98379386]
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.7 SIR Brasil - Dados Reais

3.7.1 Os dados

Lendo e visualizandos os dados do Brasil do COVID fornecidos por

- Wesley Cota

Os dados são atualizados diariamente... Então normalmente a cada dia esse pipeline pode mudar seus resultados.

[1]:
```
import pandas as pd

# Lendo os dados online - wcota
data_path = 'https://raw.githubusercontent.com/wcota/covid19br/master/cases-brazil-
↪states.csv'

online_data = pd.read_csv(data_path, delimiter=",")
online_data.head()
```

[1]:
```
         date country  state    city  newDeaths  deaths  newCases  totalCases  \
0  2020-02-25  Brazil     SP  TOTAL          0       0         1           1
1  2020-02-25  Brazil  TOTAL  TOTAL          0       0         1           1
2  2020-02-26  Brazil     SP  TOTAL          0       0         0           1
3  2020-02-26  Brazil  TOTAL  TOTAL          0       0         0           1
4  2020-02-27  Brazil     SP  TOTAL          0       0         0           1

   deathsMS  totalCasesMS  deaths_per_100k_inhabitants  \
0         0             0                          0.0
1         0             0                          0.0
2         0             1                          0.0
3         0             1                          0.0
4         0             1                          0.0

   totalCases_per_100k_inhabitants  deaths_by_totalCases  recovered  suspects  \
0                          0.00218                   0.0        NaN       NaN
1                          0.00048                   0.0        NaN       NaN
2                          0.00218                   0.0        NaN       NaN
3                          0.00048                   0.0        NaN       NaN
4                          0.00218                   0.0        NaN       NaN

   tests  tests_per_100k_inhabitants
0    NaN                         NaN
1    NaN                         NaN
2    NaN                         NaN
3    NaN                         NaN
4    NaN                         NaN
```

Filtrando e limpando os dados

[2]:
```
selected_state = "SP"

at_state = online_data['state']==selected_state
local_data = online_data[at_state]
local_data = local_data[local_data.recovered.notnull()]
#local_data = local_data.fillna(method="backfill")

local_data.head()
```

[2]:
```
           date country state    city  newDeaths  deaths  newCases  totalCases  \
347  2020-03-24  Brazil    SP  TOTAL         10      40        65         810
375  2020-03-25  Brazil    SP  TOTAL          8      48        52         862
403  2020-03-26  Brazil    SP  TOTAL         10      58       191        1053
```

(continues on next page)

(continued from previous page)

```
431  2020-03-27  Brazil    SP  TOTAL         10      68       170         1223
459  2020-03-28  Brazil    SP  TOTAL         16      84       183         1406

     deathsMS  totalCasesMS  deaths_per_100k_inhabitants  \
347        40           810                      0.08711
375        48           862                      0.10453
403        58          1052                      0.12631
431        68          1223                      0.14809
459        84          1406                      0.18293

     totalCases_per_100k_inhabitants  deaths_by_totalCases  recovered  \
347                          1.76397               0.04938        1.0
375                          1.87722               0.05568        1.0
403                          2.29317               0.05508        1.0
431                          2.66338               0.05560        1.0
459                          3.06191               0.05974        1.0

     suspects  tests  tests_per_100k_inhabitants
347    4572.0    NaN                         NaN
375    4300.0    NaN                         NaN
403   14312.0    NaN                         NaN
431   14312.0    NaN                         NaN
459   14312.0    NaN                         NaN
```

[3]:

```
import numpy as np
from datetime import datetime

first_date = local_data["date"].iloc[0]
first_date = datetime.fromisoformat(first_date)

if selected_state == "SP":
#     N = 11869660
    N = 44.01e6
elif selected_state == "TOTAL":
    N = 220e6

I = list()                                          # <- I(t)
R = local_data["recovered"].iloc[1:].to_numpy()  # <- R(t)
M = local_data["newDeaths"].iloc[1:].to_numpy()  # <- M(t)
nR = np.diff(local_data["recovered"].to_numpy()) # <- dR(t)/dt
nC = local_data["newCases"].iloc[1:].to_numpy()  # <- nC(t)/dt

I = [ local_data["totalCases"].iloc[1] ]          # I(0)

# I(t) <- I(t-1) + newCases(t) - newMortes(t) - newRecovered(t)
for t in range(len(M)-1):
    I.append(I[-1] + nC[t] - M[t] - nR[t])
I = np.array(I)
```

Visualizando a evolução

[18]:

(continues on next page)

(continued from previous page)

```
from bokeh.models   import Legend, ColumnDataSource, RangeTool, LinearAxis, Range1d,␣
↪HoverTool
from bokeh.palettes import brewer, Inferno256
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook

output_notebook()

from datetime import timedelta

# Criando o vetor de tempo
date_vec = [ first_date + timedelta(days=k) for k in range(len(M))]

# Criando os valores para legenda no plot
year =  [str(int(d.year)) for d in date_vec ]
month = [("0"+str(int(d.month)))[-2:] for d in date_vec ]
day =   [("0"+str(int(d.day)))[-2:] for d in date_vec ]



# Criando a fonte de dados
source = ColumnDataSource(data={
    'Data'       : date_vec,
    'd': day, 'm': month, 'y': year,
    'Infectados' : I,
    'Removidos'  : R,
    'Mortes'     : M,
})


# Criando a figura
p = figure(plot_height=500,
           plot_width=600,
           x_axis_type="datetime",
           tools="",
           toolbar_location=None,
           y_axis_type="log",
           title="Evolução do COVID - São Paulo")

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

# Incluindo as curvas
i_p = p.line(x='Data', y='Infectados', legend_label="Infectados", line_cap="round",␣
↪line_width=3, color="#ffd885", source=source)
m_p = p.line(x='Data', y='Mortes', legend_label="Mortes", line_cap="round", line_
↪width=3, color="#de425b", source=source)
r_p = p.line(x='Data', y='Removidos', legend_label="Removidos", line_cap="round",␣
↪line_width=3, color="#99d594", source=source)

# Colocando as legendas
p.legend.click_policy="hide"
```

(continues on next page)

(continued from previous page)

```
p.legend.location = "top_left"

# Incluindo a ferramenta de hover
p.add_tools(HoverTool(
    tooltips=[
        ( 'Indivíduos', '$y{i}'),
        ( 'Data',       '@d/@m/@y' ),
    ],
    renderers=[
        r_p, i_p, m_p
    ]
))

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.7.2 O problema

O conjunto de equações diferenciais que caracteriza o modelo é descrito abaixo. No modelo $:nbsphinx-math:*beta* '- *:nbsphinx-math:*text{representa a taxa de transmissão ou taxa efetiva de contato}' $ e r – a taxa de remoção ou recuperação.

$$\frac{dS(t)}{dt} = -\beta S(t)I(t)$$
$$\frac{dI(t)}{dt} = \beta S(t)I(t) - rI(t)$$
$$\frac{dR(t)}{dt} = rI(t)$$

Gostaríamos de identificar quais parâmetros β e r resultam num melhor ajuste do modelo para os dados de **S**,**I** e **R**

[5]:
```
# Importando o modelo SIR
from models import *

sir_model = ss.SIR(pop=N, focus=["I", "R"])
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

3.7.3 Estimando os parâmetros

Para estimarmos os parâmetros do modelo β e **r**, vamos utilizar inicialmente o método de mínimos quadrados. Podemos então formular o problema a partir da Equação abaixo. Na Equação $y_m(k)$ representa o dado real em cada amostra k; $y_s(\theta, k)$ representa o **valor estimado** a partir da simulação do modelo para uma determinada amostra k e θ representa o vetor ed parâmetros $\theta = [\beta \ \ r]^T$.

$$min_\theta = \sum_{k=1}^{K} (y_m(k) - y_s(\theta, k))^2$$

A equação formula a pergunta: quais os valores de $beta$ e r que minizam o erro quadrático quando comparados com os dados reais.

[12]:
```
import numpy as np

S = N - I - R

time = np.linspace(0, len(I), len(I))

# Estimando os par^ametros
sir_model.fit(S, I, R, time, sample_ponder=True, resample=True, beta_sens=[1000,10],␣
↪r_sens=[1000,10])

r_included = True
```

```
         ├─ Resample from sizes —  65 65 65 65
         └─ Resample to sizes —    1540 1540 1540 1540
         ├─ S(0) — I(0) — R(0) —  [44009136.41827327, 862.5810526571099, 1.
↪0006740750082481]
         ├─ beta —   1   r —   0.14285714285714285
         ├─ beta bound —   0.001  —  10
         ├─ r bound —   0.00014285714285714284  —  1.4285714285714284
         ├─ equation weights —   [0.0016004861866103122, 1, 1]
         └─ Defined at:  0.08388235287485474  —  0.010752489662406836
```

[13]:
```
sir_model.parameters[0]/sir_model.parameters[1] # Ro <- \beta * (1 / \r)
```

[13]:
```
7.801202838457649
```

[19]:
```
# Criando a figura
p1 = figure(plot_height=500,
            plot_width=600,
            x_axis_type="datetime",
            tools="",
            toolbar_location=None,
#             y_axis_type="log",
            title="Evolução do COVID - São Paulo")

# Preparando o estilo
p1.grid.grid_line_alpha = 0
p1.ygrid.band_fill_color = "olive"
p1.ygrid.band_fill_alpha = 0.1
p1.yaxis.axis_label = "Indivíduos"
```

(continues on next page)

(continued from previous page)

```
p1.xaxis.axis_label = "Dias"

# Incluindo as curvas
p1.line(time, I, legend_label="Infectados", line_cap="round", line_width=3, color="
↪#ffd885")
#p1.line(time, , legend_label="Mortes", line_cap="round", line_width=3, color="#de425b
↪")
p1.line(time, R, legend_label="Removidos", line_cap="round", line_width=3, color="
↪#99d594")

p1.scatter(sir_model.pipeline["resample"]["after"]["t"],
           sir_model.pipeline["resample"]["after"]["I"],
           marker="circle",line_color="#6666ee", fill_color="#ee6666", fill_alpha=0.5,
↪ size=3)

p1.scatter(sir_model.pipeline["resample"]["after"]["t"],
           sir_model.pipeline["resample"]["after"]["R"],
           marker="circle",line_color="#6666ee", fill_color="#ee6666", fill_alpha=0.5,
↪ size=3)

# Colocando as legendas
p1.legend.click_policy="hide"
p1.legend.location = "top_left"


show(p1)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

[20]:

```
if r_included:
    initial = [S[0], I[0], R[0]]
else:
    initial = [S[0], I[0]]

results = sir_model.predict(initial, time)
```

[21]:

```
# Incluindo os dados de infectados
im_p = p.line(
    date_vec, results[1],
    legend_label="Infectados - Modelo",
    line_width=4,
    line_dash="dashed",
    line_cap="round",
    color="#f57f17"
)

# Incluindo os dados de recuperados
if r_included:
```

(continues on next page)

(continued from previous page)

```
    rm_p = p.line(
        date_vec, results[2],
        legend_label="Removidos - Modelo",
        line_dash="dashed",
        line_width=4,
        line_cap="round",
        color="#1b5e20"
    )

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.7.4 Predições utilizando o modelo

[17]:

```
# Criando os valores de tempo para previsão - 120 dias
t_sim = np.linspace(0, len(I) + 120, len(I) + 120)
date_vec_sim = [first_date + timedelta(days=k) for k in t_sim]

# Prevendo para os valores selecionados
prediction = sir_model.predict(initial, t_sim)



# Criando o gráfico com as predições

# Criando os valores para legenda no plot
year_sim =  [str(int(d.year)) for d in date_vec_sim ]
month_sim = [("0"+str(int(d.month)))[-2:] for d in date_vec_sim ]
day_sim =   [("0"+str(int(d.day)))[-2:] for d in date_vec_sim ]

if r_included:
    accum_Infect = [0]
    for i in N - prediction[1] - prediction[2]:
        accum_Infect.append(accum_Infect[-1]+i)

    accum_Infect = sir_model.parameters[1] * np.array(accum_Infect)

# Criando a fonte de dados
if r_included:
    source = ColumnDataSource(data={
        'Data'       : date_vec,
        'd': day, 'm': month, 'y': year,
        'Infectados' : I,
        'Removidos'  : R,
        'Mortes'     : M,
        'InfecModelo' : prediction[1],
        'RemovModelo' : prediction[2],
        'AccumInfect' : accum_Infect,
        'SucetModelo' : N - prediction[1] - prediction[2],
```

(continues on next page)

(continued from previous page)

```
        'DataModelo'   : date_vec_sim,
        'ds': day_sim, 'ms': month_sim, 'ys': year_sim
    })
else:
    source = ColumnDataSource(data={
        'Data'       : date_vec,
        'd': day, 'm': month, 'y': year,
        'Infectados' : I,
        'Removidos'  : R,
        'Mortes'     : M,
        'InfecModelo' : prediction[1],
        'DataModelo'   : date_vec_sim,
        'ds': day_sim, 'ms': month_sim, 'ys': year_sim
    })


# Criando a figura
p = figure(plot_height=700,
           plot_width=800,
           x_axis_type="datetime",
           tools="",
           toolbar_location=None,
           y_axis_type="log",
           title="Previsão do COVID - Brasil")

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

# Incluindo as curvas
i_p = p.line(x='Data', y='Infectados', legend_label="Infectados", line_cap="round",␣
↪line_width=3, color="#ffd885", source=source)
m_p = p.line(x='Data', y='Mortes', legend_label="Mortes", line_cap="round", line_
↪width=3, color="#de425b", source=source)
r_p = p.line(x='Data', y='Removidos', legend_label="Removidos", line_cap="round",␣
↪line_width=3, color="#99d594", source=source)

mp_p = p.line(x='DataModelo', y='InfecModelo', legend_label="Infectados - Modelo",␣
↪line_dash="dashed", line_cap="round", line_width=4, color="#f57f17", source=source)

renders = [i_p, m_p, r_p, mp_p]

if r_included:
    rp_p = p.line(x='DataModelo', y='RemovModelo', legend_label="Removidos - Modelo",␣
↪line_dash="dashed", line_cap="round", line_width=4, color="#1b5e20", source=source)
    renders.append(rp_p)

# Colocando as legendas
p.legend.click_policy="hide"
p.legend.location = "top_left"

# Incluindo a ferramenta de hover
p.add_tools(HoverTool(
    tooltips=[
```

(continues on next page)

(continued from previous page)

```
        ( 'Indivíduos', '$y{0.00 a}' ),
        ( 'Data',       '@ds/@ms/@ys'),
    ],
    renderers=renders
))

show(p)
```

```
BokehUserWarning: ColumnDataSource's columns must be of the same length. Current␣
↪lengths: ('AccumInfect', 186), ('Data', 65), ('DataModelo', 185), ('InfecModelo',␣
↪185), ('Infectados', 65), ('Mortes', 65), ('RemovModelo', 185), ('Removidos', 65), (
↪'SucetModelo', 185), ('d', 65), ('ds', 185), ('m', 65), ('ms', 185), ('y', 65), ('ys
↪', 185)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.7.5 Referências

- Predictive Monitoring of COVID-19
- Apple mobility data
- Corona Virus - Brazil Data
- Fitting model to Corona Virus

3.8 SIR China - Dados Reais

3.8.1 Os dados

Para coletar os dados de outros países, foram utilizadas as APIs dos sites:

- About Corona
- Europe RestFul

[1]:

```
import requests
import pandas as pd

covid_api = 'https://corona-api.com/countries/'
rest_countries = 'https://restcountries.eu/rest/v2/alpha/'
country = 'CN' # Alpha-2 ISO3166

data_json =  requests.get(covid_api + country).json()
country = requests.get(covid_api + country).json()

N = country['data']['population']

print(country['data']['name'])
```

```
China
```

Organizando os dados

[2]:
```
from datetime import datetime

df = pd.DataFrame(data_json['data']['timeline'])
df = df.sort_values('date').reset_index()

from datetime import datetime, timedelta
df['date'] = [datetime.fromisoformat(f) for f in df['date']]
df = df.drop_duplicates(subset='date', keep = 'last')

# Criando o vetor de tempo
first_date = df['date'].iloc[0]
size_days = (df['date'].iloc[-1] - df['date'].iloc[0]).days
date_vec = [first_date + timedelta(days=k) for k in range(size_days)]

new_df = pd.DataFrame(date_vec, columns=['date'])
new_df = pd.merge(new_df, df, how='left', on= 'date')
new_df = new_df.drop(columns= ['index',  'updated_at', 'is_in_progress'])

for col in new_df.columns[1:]:
    new_df[col] = new_df[col].interpolate(method='polynomial', order=1)
df = new_df.dropna()

df.head()
```

[2]:
```
        date  deaths  confirmed  active  recovered  new_confirmed  \
0 2020-01-21      17        548     503         28            548
1 2020-01-22      18        643     595         30             95
2 2020-01-23      26        920     858         36            277
3 2020-01-24      42       1406    1325         39            486
4 2020-01-25      56       2075    1970         49            669

   new_recovered  new_deaths
0             28          17
1              2           1
2              6           8
3              3          16
4             10          14
```

Visualizando os dados

[3]:
```
from bokeh.models   import Legend, ColumnDataSource, RangeTool, LinearAxis, Range1d,␣
↪HoverTool
from bokeh.palettes import brewer, Inferno256
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook
```

(continues on next page)

(continued from previous page)

```
output_notebook()

import numpy as np

# Criando os valores para legenda no plot
year =  [str(int(d.year)) for d in df['date'] ]
month = [("0"+str(int(d.month)))[-2:] for d in df['date'] ]
day =   [("0"+str(int(d.day)))[-2:] for d in df['date'] ]

# Criando a fonte de dados
source = ColumnDataSource(data={
    'Data'       : df['date'].values,
    'd': day, 'm': month, 'y': year,
    'Infectados Acc' : df['confirmed'].values,
    'Mortes'     : df['deaths'].values,
    'Ativo'     : df['active'].values,
    'Recuperados': df['recovered'].values
})


# Criando a figura
p = figure(plot_height=500,
           plot_width=600,
           x_axis_type="datetime",
           tools="",
           #y_axis_type="log",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

# Incluindo as curvas
i_p = p.line(x='Data', y='Ativo',
             legend_label="Infectados",
             line_cap="round", line_width=5, color="#c62828", source=source)
m_p = p.line(x='Data', y='Mortes',
             legend_label="Mortes",
             line_cap="round", line_width=5, color="#512da8", source=source)
c_p = p.line(x='Data', y='Infectados Acc',
             legend_label="Infectados Acc",
             line_cap="round", line_width=5, color="#0288d1", source=source)
r_p = p.line(x='Data', y='Recuperados',
             legend_label="Recuperados",
             line_cap="round", line_width=5, color="#388e3c", source=source)

# Colocando as legendas
p.legend.click_policy="hide"
# p.legend.location = "top_left"
p.legend.location = "top_left"

# Incluindo a ferramenta de hover
p.add_tools(HoverTool(
```

(continues on next page)

(continued from previous page)

```
    tooltips=[
        ( 'Indivíduos', '$y{i}'),
        ( 'Data',       '@d/@m/@y' ),
    ],
    renderers=[
        m_p, i_p, c_p, r_p
    ]
))

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Verificando os dados

[4]:

```
dif_I = np.diff(df['active'])
cum = []
cum.append(dif_I[0])

for k, i in enumerate(dif_I):
    cum.append(cum[-1] + i)

cum = np.array(cum)
cum += df['deaths'].to_numpy() + df['recovered'].to_numpy()

print("Erro entre casos acumulados e valores de confirmados: {}".format(
        round(sum((cum - df['confirmed'].values)**2 / len(cum)),2) ) )
```

```
Erro entre casos acumulados e valores de confirmados: 168921.0
```

Criando os dados SIR

[5]:

```
I = df['active'].to_numpy()
R = df['recovered'].to_numpy()
M = df['deaths'].to_numpy()
S = N - R - I

# Creating the time vector
t = np.linspace(0, len(I), len(I))
```

(continues on next page)

(continued from previous page)

```
Sd, Id, Md, Rd, td = S, I, M, R, t
```

[7]:
```
Ro = (np.log(S[0]/N) - np.log(S[-1]/N))/(S[0]/N - S[-1]/N)
print("Calculo do Ro baseado nos dados:", Ro)
```

```
Calculo do Ro baseado nos dados: 1.0000300994523579
```

3.8.2 Estimando utilizando todos os dados

[8]:
```
from models import *

dataset = dict(S=Sd, I=Id, R=Rd)

# Create the model
sir_model = ss.SIR(pop=N, focus=["S", "I", "R"])

# Adjust the parameters
sir_model.fit(dataset, td,
              search_pop=True,
              pop_sens=[0.00001,0.05],
              beta_sens=[100,10000],
              r_sens=[100,100])


# Predict the model
sim_res = sir_model.predict((Sd[0],Id[0],Rd[0]), td)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

```
        ├─ S(0) - I(0) - R(0) -  [1330043469, 503, 28]
        ├─ beta -   1   r -   0.14285714285714285
        ├─ beta bound -   0.01   -  10000
        ├─ r bound -   0.0014285714285714286   -  14.285714285714285
        ├─ equation weights -   [7.51893117169047e-10, 7.90340642628301e-05, 1.
↪8154828650455473e-05]
        ├─ Running on -  differential_evolution SciPy Search Algorithm
        └─ Defined at:  5100.406034312756  -  0.04454555270373974
```

[9]:
```
print("Par^ametros estimados: ", sir_model.parameters)
print("Suposto Ro: ", sir_model.parameters[0] * sir_model.parameters[-1] / sir_model.
↪parameters[1])
print("Dias contaminados: ", 1 / sir_model.parameters[1])
```

```
Par^ametros estimados:  [5.10040603e+03 4.45455527e-02 6.14351607e-05]
Suposto Ro:  7.034243490574524
Dias contaminados:  22.448930124421754
```

[14]:
```
p = figure(plot_height=500,
           plot_width=800,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

p.line(t, I,
       legend_label="Infectados", color="#ff6659", line_width=4)
p.line(t, R,
       legend_label="Removidos", color="#76d275", line_width=4)

# Show the results
p.line(td, sim_res[1],
       legend_label="Infectados - Modelo", line_dash="dashed", color="#d32f2f", line_
↪width=3)
p.line(td, sim_res[2],
    legend_label="Removidos - Modelo", line_dash="dashed", color="#43a047", line_
↪width=3)
p.line(td, sim_res[0],
      legend_label="Suscetíveis Ponderados - Modelo", line_dash="dashed", color="
↪#1e88e5", line_width=3)
p.add_layout(p.legend[0], 'right')

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.8.3 Monte Carlo

Nesta parte, faremos um teste aumentando a quantidade de amostras de treinamento e prevendo o momento do pico da epidemia a medida que mais dias são utilizados para treinamento. Esse estudo vai possibilitar a análise da certeza da previsão do pico da epidemia antes desse acontecer.

[15]:
```
saved_param = {'r':[], 'beta':[], 'pop':[]}
saved_prediction = []

start_day = 10
pred_t = np.array(range(int(td[-1])))

for i in range(start_day, len(I)):

    dataset = dict(S=Sd[:i], I=Id[:i], R=Rd[:i])
    td_ = td[:i]
```

(continues on next page)

(continued from previous page)

```
    # Create the model
    sir_model = ss.SIR(pop=N, focus=["S", "I", "R"], verbose = False)

    # Adjust the parameters
    sir_model.fit(dataset, td_,
                  search_pop=True,
                  pop_sens=[0.00001,0.05],
                  beta_sens=[100,10000],
                  r_sens=[100,100])

    saved_param['beta'].append(sir_model.parameters[0])
    saved_param['r'].append(sir_model.parameters[1])
    saved_param['pop'].append(sir_model.parameters[2])

    saved_prediction.append(sir_model.predict((Sd[0],Id[0], Rd[0]), pred_t))
```

```
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/opt/anaconda3/lib/python3.7/site-packages/scipy/integrate/odepack.py:248:␣
↪ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with␣
↪full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
```

```
[17]: import pickle

      with open("./China_II_mc_runs.pickle", "wb") as handle:
          pickle.dump({"pars":saved_param, "pred":saved_prediction}, handle)
```

3.8.4 Análise do uso do sistema de saúde

Nesta análise, mostramos o erro percentual do quanto antes do pico, conseguimos prever a quantidade de pessoas que realmente serão identificadas como infectadas, uma vez que $R(\infty)$ é a quantidade de pessoas recuperadas totais, daquelas que foram noficadas como infectadas no sistema de saúde. Desta forma segue o erro proporcional do erro a medida em que novos dados diários foram incluidos no modelo:

[34]:
```
x = range(start_day, len(I))
usage_error = [ 100 * abs(p*N - Rd[-1]) / Rd[-1] for p in saved_param['pop']]

fig9 = go.Figure()

fig9.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=usage_error[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="ϵ(%) = %{y:.0f}, <br> com %{x:.0f} dias de dados."))
fig9.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=usage_error[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="ϵ(%) = %{y:.0f}, <br> com %{x:.0f} dias de dados."))
fig9.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[usage_error[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um ϵ(%) = %{y:.0f}."))
fig9.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[usage_error[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para ϵ(%)␣
↪= %{y:.4f}."))
fig9.update_layout(template='xgridoff', yaxis_range=[-1,100],
                   legend_orientation="h", legend=dict(x=0.10, y=1.05),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Erro (%) do R(∞) - notificados',
                   title_text="Erro da previsão do uso do sistema de saúde -  " +␣
↪country['data']['name'])

fig9.show(renderer="png")
```

3.8.5 Visualizando as previsões de $I(t)$

Vamos analisar as previsões quando somente os dados antes do pico são fornecidos ao modelo, e as previões utilizando os dados após o pico:

[19]:
```
visual_peak = 30

p2 = figure(plot_height=500,
           plot_width=600,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p2.grid.grid_line_alpha = 0
p2.ygrid.band_fill_color = "olive"
p2.ygrid.band_fill_alpha = 0.1
p2.yaxis.axis_label = "Indivíduos"
p2.xaxis.axis_label = "Dias"

# Incluindo as curvas
p2.line(td, Id,
       legend_label="Infectados",
       line_cap="round", line_width=3, color="#c62828")

for data in saved_prediction[visual_peak-start_day:]:
```

(continues on next page)

(continued from previous page)

```
    p2.line(pred_t[:len(td)], -data[1][:len(td)],
            legend_label="Previsão Infectados - Depois do pico",
            line_cap="round", line_dash="dashed", line_width=4, color="#ffa000", line_
↪alpha = 0.1)

for data in saved_prediction[:visual_peak-start_day]:
    p2.line(pred_t[:len(td)], -data[1][:len(td)],
            legend_label="Previsão Infectados - Antes do pico",
            line_cap="round", line_dash="dashed", line_width=4, color="#42a5f5", line_
↪alpha = 0.3)

# Colocando as legendas
p2.legend.click_policy="hide"
p2.legend.location = "top_right"

show(p2)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.8.6 Visualizando os ajustes dos grupos

Aqui vamos analisar as previsões obtidas para cada grupo $S(t)$, $I(t)$ e $R(t)$, a medida que mais dias foram informados ao modelo.

[26]:
```
p3 = figure(plot_height=350,
            plot_width=600,
            tools="",
            toolbar_location=None,
            title="Evolução do COVID - I(t) e R(t) - " + country['data']['name'])

p4 = figure(plot_height=350,
            plot_width=600,
            tools="",
            toolbar_location=None,
            title="Evolução do COVID - S(t) - " + country['data']['name'])

plot_all = True

# Preparando o estilo
p3.grid.grid_line_alpha = 0
p3.ygrid.band_fill_color = "olive"
p3.ygrid.band_fill_alpha = 0.1
p3.yaxis.axis_label = "Indivíduos"
p3.xaxis.axis_label = "Dias"

p4.grid.grid_line_alpha = 0
p4.ygrid.band_fill_color = "olive"
p4.ygrid.band_fill_alpha = 0.1
p4.yaxis.axis_label = "Indivíduos"
p4.xaxis.axis_label = "Dias"
```

(continues on next page)

(continued from previous page)

```
# Incluindo as curvas
for data in saved_prediction[20:]:
    p3.line(pred_t, data[1],
           legend_label="Previsão Infectados",
           line_cap="round", line_dash = 'dashed',
           line_width=4, color="#42a5f5", line_alpha = 0.1)

    p3.line(pred_t, data[2],
           legend_label="Previsão Recuperados",
           line_cap="round", line_dash = 'dashed', line_width=4,
           color="#9c27b0", line_alpha = 0.07)


p3.line(td, Id,
       legend_label="Infectados",
       line_cap="round", line_width=3, color="#005cb2")

p3.line(td, Rd,
           legend_label="Recuperados",
           line_cap="round", line_width=3, color="#5e35b1")

if plot_all:
    for data in saved_prediction:
        p4.line(pred_t, data[0] + N*(1-saved_param['pop'][-1]),
               legend_label="Previsão Suscetiveis",
               line_cap="round", line_dash = 'dashed',
               line_width=4, color="#ff5722", line_alpha = 0.07)
    p4.line(td, Sd,
           legend_label="Suscetiveis",
           line_cap="round", line_width=3, color="#b71c1c")

# Colocando as legendas
p3.legend.click_policy="hide"
p3.legend.location = "top_left"

p4.legend.click_policy="hide"
p4.legend.location = "top_right"

show(column(p3,p4))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.8.7 Análise de variação do R_0

[33]:
```
x = range(start_day, len(I))
beta_norm = [b*p for b,p in zip(saved_param['beta'],saved_param['pop'])]
Ro = [b*p/r for b,r,p in zip(saved_param['beta'],saved_param['r'],saved_param['pop'])]


fig1 = go.Figure()
```

(continues on next page)

(continued from previous page)

```
fig1.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=beta_norm[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="β = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig1.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=beta_norm[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="β = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[beta_norm[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um β = %{y:.4f}."))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[beta_norm[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para β =
↪%{y:.4f}."))
fig1.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.10, y=1.05),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro β',
                   title_text="Estimativa do par^ametro β -  " + country['data']['name
↪'])

fig1.show(renderer="png")
```

[32]:

```
fig2 = go.Figure()

fig2.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=saved_param['r'][:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig2.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=saved_param['r'][peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig2.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[saved_param['r'][peak_pos-start_
↪day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um r = %{y:.4f}."))
fig2.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[saved_param['r'][-1]],
```

(continues on next page)

(continued from previous page)

```
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para r =
↪%{y:.4f}."))
fig2.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.07, y=1.06),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro r',
                   title_text="Estimativa do par^ametro r -  " + country['data']['name
↪'])

fig2.show(renderer="png")
```

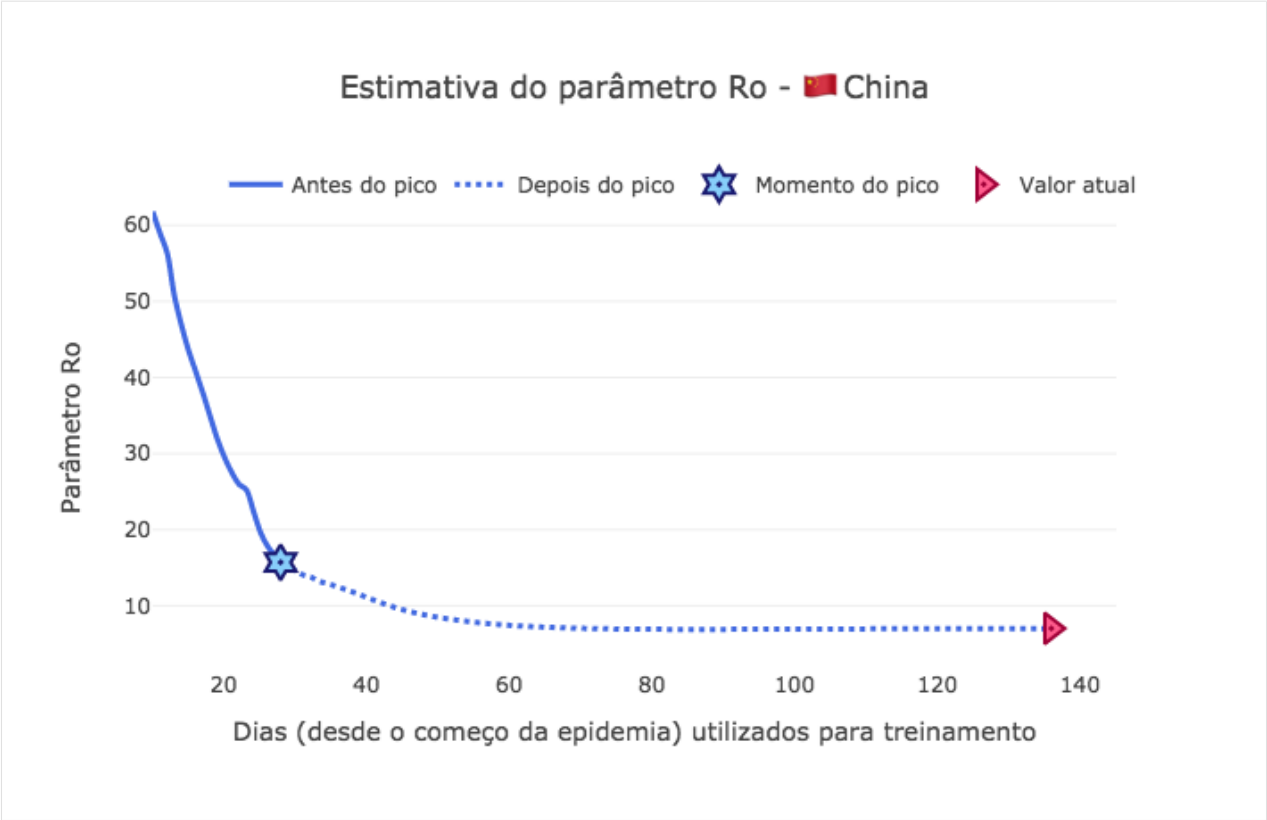
[31]:

```
fig3 = go.Figure()

fig3.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=Ro[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig3.add_trace(go.Scatter(
```

(continues on next page)

(continued from previous page)

```
                    x=td[peak_pos:-1],
                    y=Ro[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="Ro = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig3.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[Ro[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um Ro = %{y:.4f}."))
fig3.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[Ro[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para Ro␣
↪= %{y:.4f}."))
fig3.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.07, y=1.06),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro Ro',
                   title_text="Estimativa do par^ametro Ro -  " + country['data']['name
↪'])

fig3.show(renderer="png")
```



3.8.8 Análise de confiança

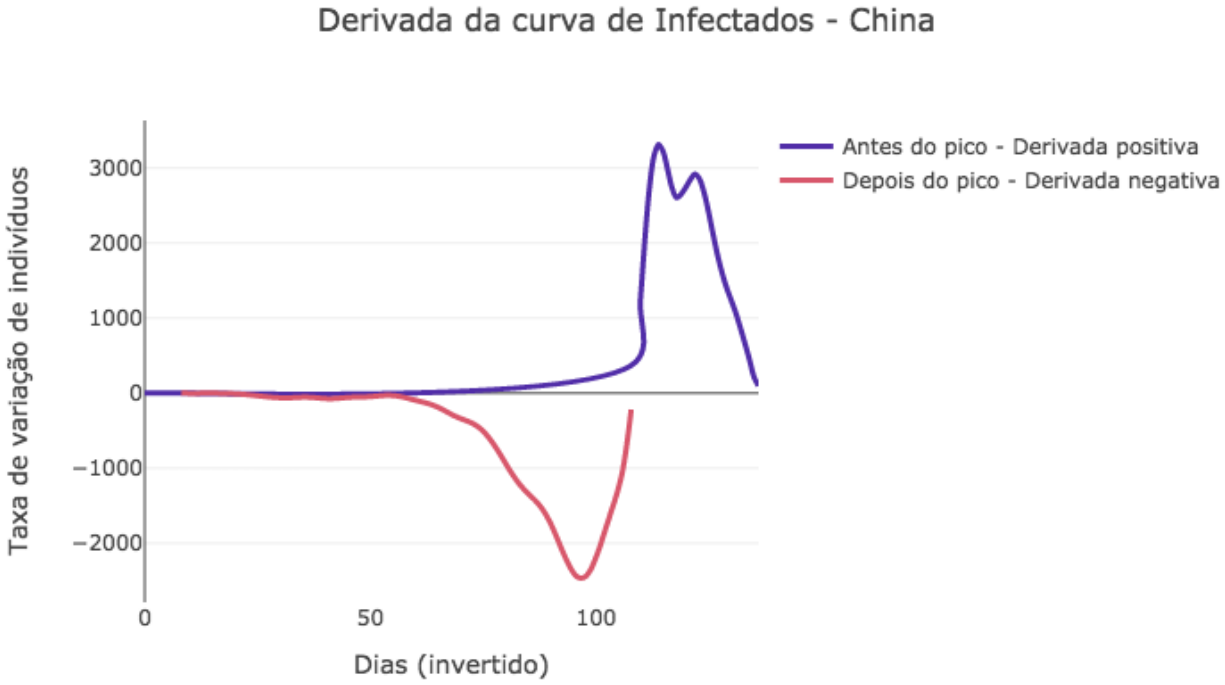
[20]:

```
from PyAstronomy import pyasl

dI = np.gradient(pyasl.smooth(I, 13, "hamming"))
t = np.linspace(0, len(dI), len(dI))

signal = np.array([di >= 0 for di in dI[::-1]])


import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(
                    x=t[signal],
                    y=dI[::-1][signal],
                    mode='lines',
                    name='Antes do pico - Derivada positiva',
                    line_shape='spline',
                    line = dict(color='#512da8', width=3)))
fig.add_trace(go.Scatter(
                    x=t[~signal],
                    y=dI[::-1][~signal],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico - Derivada negativa',
```

(continues on next page)

(continued from previous page)

```
                    line = dict(color='#d8576b', width=3)))

fig.update_layout(template='xgridoff',
                  xaxis_title='Dias (invertido)',
                  yaxis_title='Taxa de variação de indivíduos',
                  title_text="Derivada da curva de Infectados - " + country['data']['name'])
# 'ggplot2', 'seaborn', 'simple_white', 'plotly',
# 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
# 'ygridoff', 'gridon', 'none'

fig.show(renderer="png")
```



[22]:

```
peak_pos = np.argmax(~signal[::-1])
print("O pico da epidemia acontece no dia", peak_pos)

final_day = int(td[-1])

estimated_peaks = []
for data in saved_prediction:
    dI = np.gradient(data[1][:final_day])
    signal_pred = np.array([di >= 0 for di in dI[::-1]])
    estimated_peaks.append(len(Id) - np.argmax(signal_pred))
estimated_peaks = np.array(estimated_peaks)
```

```
O pico da epidemia acontece no dia 28
```

[23]:

```
import plotly.graph_objects as go

peak_error = np.abs(estimated_peaks - peak_pos)

fig1 = go.Figure()
fig1.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=peak_error[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="Erro de %{y} dias, <br> com %{x:.0f} dias de dados.
↪"))
fig1.add_trace(go.Scatter(
                    x=td[peak_pos:],
                    y=peak_error[peak_pos-start_day:],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="Erro de %{y} dias, <br> com %{x:.0f} dias de dados.
↪"))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[peak_error[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x} depois do começo da epidemia."))

fig1.update_layout(yaxis_range=[-1,31],
                   template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.20, y=1.0),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Erro (em dias) da estimativa do pico',
                   title_text="Erro da estimativa do pico - " + country['data']['name
↪'])
# 'ggplot2', 'seaborn', 'simple_white', 'plotly',
# 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
# 'ygridoff', 'gridon', 'none'

fig1.show(renderer="png")
```

3.9 SIR Itália - Dados Reais

3.9.1 Os dados

Para coletar os dados de outros países, foram utilizadas as APIs dos sites:

- About Corona
- Europe RestFul

[1]:
```
import requests
import pandas as pd

covid_api = 'https://corona-api.com/countries/'
rest_countries = 'https://restcountries.eu/rest/v2/alpha/'
country = 'IT' # Alpha-2 ISO3166

data_json =  requests.get(covid_api + country).json()
country = requests.get(covid_api + country).json()

N = country['data']['population']

print(country['data']['name'])
```

```
Italy
```

Organizando os dados

[2]:
```
from datetime import datetime

df = pd.DataFrame(data_json['data']['timeline'])
df = df.sort_values('date').reset_index()

from datetime import datetime, timedelta
df['date'] = [datetime.fromisoformat(f) for f in df['date']]
df = df.drop_duplicates(subset='date', keep = 'last')

# Criando o vetor de tempo
first_date = df['date'].iloc[0]
size_days = (df['date'].iloc[-1] - df['date'].iloc[0]).days
date_vec = [first_date + timedelta(days=k) for k in range(size_days)]

new_df = pd.DataFrame(date_vec, columns=['date'])
new_df = pd.merge(new_df, df, how='left', on= 'date')
new_df = new_df.drop(columns= ['index',  'updated_at', 'is_in_progress'])

for col in new_df.columns[1:]:
    new_df[col] = new_df[col].interpolate(method='polynomial', order=1)
df = new_df.dropna()

df.head()
```

[2]:
```
        date  deaths  confirmed    active  recovered  new_confirmed  \
0 2020-01-30     0.0   2.000000  2.000000        0.0       2.000000
1 2020-01-31     0.0   2.000000  2.000000        0.0       0.000000
2 2020-02-01     0.0   2.142857  2.142857        0.0       0.142857
3 2020-02-02     0.0   2.285714  2.285714        0.0       0.285714
4 2020-02-03     0.0   2.428571  2.428571        0.0       0.428571

   new_recovered  new_deaths
0            0.0         0.0
1            0.0         0.0
2            0.0         0.0
3            0.0         0.0
4            0.0         0.0
```

Visualizando os dados

[3]:
```
from bokeh.models   import Legend, ColumnDataSource, RangeTool, LinearAxis, Range1d,␣
↪HoverTool
from bokeh.palettes import brewer, Inferno256
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook

output_notebook()

import numpy as np
```

(continues on next page)

(continued from previous page)

```
# Criando os valores para legenda no plot
year =  [str(int(d.year)) for d in df['date'] ]
month = [("0"+str(int(d.month)))[-2:] for d in df['date'] ]
day =   [("0"+str(int(d.day)))[-2:] for d in df['date'] ]

# Criando a fonte de dados
source = ColumnDataSource(data={
    'Data'       : df['date'].values,
    'd': day, 'm': month, 'y': year,
    'Infectados Acc' : df['confirmed'].values,
    'Mortes'     : df['deaths'].values,
    'Ativo'     : df['active'].values,
    'Recuperados': df['recovered'].values
})


# Criando a figura
p = figure(plot_height=500,
           plot_width=600,
           x_axis_type="datetime",
           tools="",
           #y_axis_type="log",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

# Incluindo as curvas
i_p = p.line(x='Data', y='Ativo',
             legend_label="Infectados",
             line_cap="round", line_width=5, color="#c62828", source=source)
m_p = p.line(x='Data', y='Mortes',
             legend_label="Mortes",
             line_cap="round", line_width=5, color="#512da8", source=source)
c_p = p.line(x='Data', y='Infectados Acc',
             legend_label="Infectados Acc",
             line_cap="round", line_width=5, color="#0288d1", source=source)
r_p = p.line(x='Data', y='Recuperados',
             legend_label="Recuperados",
             line_cap="round", line_width=5, color="#388e3c", source=source)

# Colocando as legendas
p.legend.click_policy="hide"
# p.legend.location = "top_left"
p.legend.location = "top_left"

# Incluindo a ferramenta de hover
p.add_tools(HoverTool(
    tooltips=[
        ( 'Indivíduos', '$y{i}'),
        ( 'Data',       '@d/@m/@y' ),
    ],
```

(continues on next page)

(continued from previous page)

```
    renderers=[
        m_p, i_p, c_p, r_p
    ]
))

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Verificando os dados

[4]:

```
dif_I = np.diff(df['active'])
cum = []
cum.append(dif_I[0])

for k, i in enumerate(dif_I):
    cum.append(cum[-1] + i)

cum = np.array(cum)
cum += df['deaths'].to_numpy() + df['recovered'].to_numpy()

print("Erro entre casos acumulados e valores de confirmados: {}".format(
        round(sum((cum - df['confirmed'].values)**2 / len(cum)),2) ) )
```

```
Erro entre casos acumulados e valores de confirmados: 4.0
```

Criando os dados SIR

[5]:

```
I = df['active'].to_numpy()
R = df['recovered'].to_numpy()
M = df['deaths'].to_numpy()
S = N - R - I

# Creating the time vector
t = np.linspace(0, len(I), len(I))


Sd, Id, Md, Rd, td = S, I, M, R, t
```

3.9.2 Estimando utilizando todos os dados

[6]:
```
from models import *

dataset = dict(S=Sd, I=Id, R=Rd)

# Create the model
sir_model = ss.SIR(pop=N, focus=["S", "I", "R"])

# Adjust the parameters
sir_model.fit(dataset, td,
              search_pop=True,
              pop_sens=[0.001,0.01],
              beta_sens=[100,100],
              r_sens=[10000,100])

# Predict the model
sim_res = sir_model.predict((Sd[0],Id[0], Rd[0]), td)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

```
         ├─ S(0) - I(0) - R(0) -  [60340326.0, 2.0, 0.0]
         ├─ beta -   1   r -   0.14285714285714285
         ├─ beta bound -   0.01   -  100
         ├─ r bound -   1.4285714285714285e-05   -  14.285714285714285
         ├─ equation weights -   [1.6598704101560237e-08, 1.9910826361712874e-05, 2.
↪2503737227789617e-05]
         ├─ Running on -  differential_evolution SciPy Search Algorithm
         └─ Defined at:  65.58644881554696  -  0.023308094571579452
```

[7]:
```
print("Par^ametros estimados: ", sir_model.parameters)
print("Suposto Ro: ", sir_model.parameters[0] * sir_model.parameters[-1] / sir_model.
↪parameters[1])
print("Dias contaminados: ", 1 / sir_model.parameters[1])
```

```
Par^ametros estimados:  [6.55864488e+01 2.33080946e-02 3.10705183e-03]
Suposto Ro:  8.742906691463745
Dias contaminados:  42.90354996325364
```

[56]:
```
p = figure(plot_height=500,
           plot_width=700,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
```

(continues on next page)

(continued from previous page)

```
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

p.line(t, I,
       legend_label="Infectados", color="#ff6659", line_width=4)
p.line(t, R,
       legend_label="Removidos", color="#76d275", line_width=4)

# Show the results
p.line(td, sim_res[1],
       legend_label="Infectados - Modelo", line_dash="dashed", color="#d32f2f", line_
↪width=3)
p.line(td, sim_res[2],
    legend_label="Removidos - Modelo", line_dash="dashed", color="#43a047", line_
↪width=3)
p.line(td, sim_res[0],
      legend_label="Suscetíveis Ponderados - Modelo", line_dash="dashed", color="
↪#1e88e5", line_width=3)
p.add_layout(p.legend[0], 'right')

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.9.3 Monte Carlo

Nesta parte, faremos um teste aumentando a quantidade de amostras de treinamento e prevendo o momento do pico da epidemia a medida que mais dias são utilizados para treinamento. Esse estudo vai possibilitar a análise da certeza da previsão do pico da epidemia antes desse acontecer.

[13]:

```
saved_param = {'r':[], 'beta':[], 'pop':[]}
saved_prediction = []

start_day = 35
pred_t = np.array(range(int(td[-1]) + 120))

for i in range(start_day,len(I),1):

    dataset = dict(S=Sd[:i], I=Id[:i], R=Rd[:i])
    td_ = td[:i]

    # Create the model
    sir_model = ss.SIR(pop=N, focus=["S", "I", "R"], verbose = False)

    # Adjust the parameters
    sir_model.fit(dataset, td_,
                  search_pop=True,
                  pop_sens=[0.001,0.01],
                  beta_sens=[100000,100],
                  r_sens=[100000,100])
```

(continues on next page)

(continued from previous page)

```
    saved_param['beta'].append(sir_model.parameters[0])
    saved_param['r'].append(sir_model.parameters[1])
    saved_param['pop'].append(sir_model.parameters[2])

    saved_prediction.append(sir_model.predict((Sd[0],Id[0],Rd[0]), pred_t))
```

[27]:
```
import pickle

with open("./Italy_mc_runs.pickle", "wb") as handle:
    pickle.dump({"pars":saved_param, "pred":saved_prediction}, handle)
```

3.9.4 Análise do uso do sistema de saúde

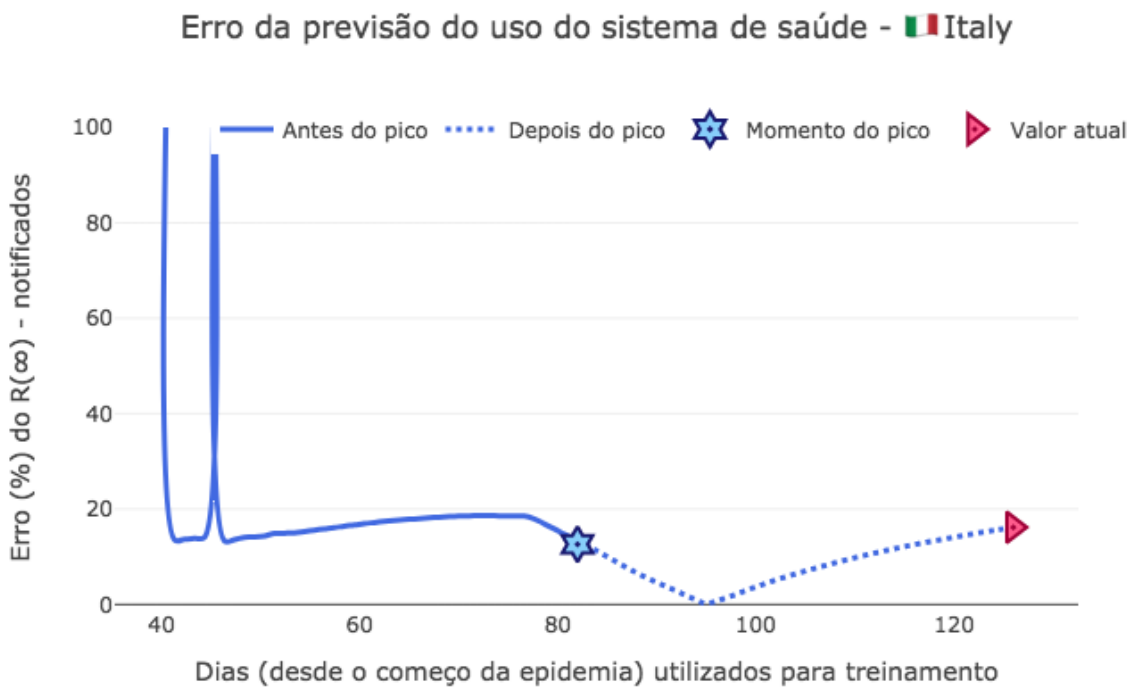
Nesta análise, mostramos o erro percentual do quanto antes do pico, conseguimos prever a quantidade de pessoas que realmente serão identificadas como infectadas, uma vez que $R(\infty)$ é a quantidade de pessoas recuperadas totais, daquelas que foram noficadas como infectadas no sistema de saúde. Desta forma segue o erro proporcional do erro a medida em que novos dados diários foram incluidos no modelo:

[71]:
```
x = range(start_day, len(I))

usage_error = [ 100 * abs(p*N - Rd[-1]) / Rd[-1] for p in saved_param['pop']]

fig9 = go.Figure()

fig9.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=usage_error[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="ϵ(%) = %{y:.0f}, <br> com %{x:.0f} dias de dados."))
fig9.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=usage_error[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="ϵ(%) = %{y:.0f}, <br> com %{x:.0f} dias de dados."))
fig9.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[usage_error[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um ϵ(%) = %{y:.0f}."))
fig9.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[usage_error[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
```

(continues on next page)

(continued from previous page)

```
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para ϵ(%)␣
↪= %{y:.4f}."))
fig9.update_layout(template='xgridoff', yaxis_range=[-1,100],
                   legend_orientation="h", legend=dict(x=0.10, y=1.05),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Erro (%) do R(∞) - notificados',
                   title_text="Erro da previsão do uso do sistema de saúde -  " +␣
↪country['data']['name'])

fig9.show(renderer="png")
```



3.9.5 Visualizando as previsões de $I(t)$

Vamos analisar as previsões quando somente os dados antes do pico são fornecidos ao modelo, e as previões utilizando os dados após o pico:

[53]:

```
p2 = figure(plot_height=500,
            plot_width=600,
            tools="",
            toolbar_location=None,
            title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p2.grid.grid_line_alpha = 0
```

(continues on next page)

(continued from previous page)

```
p2.ygrid.band_fill_color = "olive"
p2.ygrid.band_fill_alpha = 0.1
p2.yaxis.axis_label = "Indivíduos"
p2.xaxis.axis_label = "Dias"

# Incluindo as curvas
p2.line(td, Id,
       legend_label="Infectados",
       line_cap="round", line_width=3, color="#c62828")

for data in saved_prediction[45:]:
    p2.line(pred_t[:len(td)], -data[1][:len(td)],
           legend_label="Previsão Infectados - Depois do pico",
           line_cap="round", line_dash="dashed", line_width=4, color="#ffa000", line_
↪alpha = 0.1)

for data in saved_prediction[:45]:
    p2.line(pred_t[:len(td)], -data[1][:len(td)],
           legend_label="Previsão Infectados - Antes do pico",
           line_cap="round", line_dash="dashed", line_width=4, color="#42a5f5", line_
↪alpha = 0.1)

# Colocando as legendas
p2.legend.click_policy="hide"
p2.legend.location = "bottom_left"

show(p2)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.9.6 Visualizando os ajustes dos grupos

Aqui vamos analisar as previsões obtidas para cada grupo $S(t)$, $I(t)$ e $R(t)$, a medida que mais dias foram informados ao modelo.

[54]:

```
p3 = figure(plot_height=350,
           plot_width=600,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - I(t) e R(t) - " + country['data']['name'])

p4 = figure(plot_height=350,
           plot_width=600,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - S(t) - " + country['data']['name'])

plot_all = True

# Preparando o estilo
```

(continues on next page)

(continued from previous page)

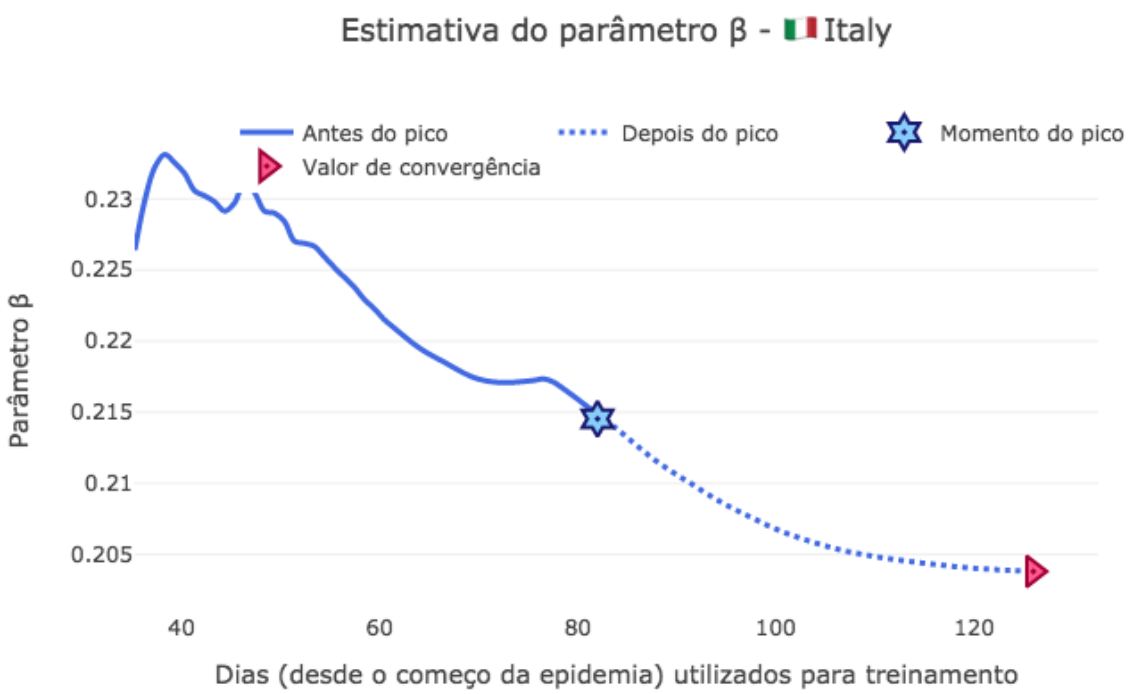
```
p3.grid.grid_line_alpha = 0
p3.ygrid.band_fill_color = "olive"
p3.ygrid.band_fill_alpha = 0.1
p3.yaxis.axis_label = "Indivíduos"
p3.xaxis.axis_label = "Dias"

p4.grid.grid_line_alpha = 0
p4.ygrid.band_fill_color = "olive"
p4.ygrid.band_fill_alpha = 0.1
p4.yaxis.axis_label = "Indivíduos"
p4.xaxis.axis_label = "Dias"

# Incluindo as curvas
for data in saved_prediction[10:]:
    p3.line(pred_t, data[1],
           legend_label="Previsão Infectados",
           line_cap="round", line_dash = 'dashed',
           line_width=4, color="#42a5f5", line_alpha = 0.1)

    p3.line(pred_t, data[2],
           legend_label="Previsão Recuperados",
           line_cap="round", line_dash = 'dashed', line_width=4,
           color="#9c27b0", line_alpha = 0.07)


p3.line(td, Id,
       legend_label="Infectados",
       line_cap="round", line_width=3, color="#005cb2")

p3.line(td, Rd,
           legend_label="Recuperados",
           line_cap="round", line_width=3, color="#5e35b1")

if plot_all:
    for data in saved_prediction[10:]:
        p4.line(pred_t, data[0] + N*(1-saved_param['pop'][-1]),
               legend_label="Previsão Suscetiveis",
               line_cap="round", line_dash = 'dashed',
               line_width=4, color="#ff5722", line_alpha = 0.07)
    p4.line(td, Sd,
           legend_label="Suscetiveis",
           line_cap="round", line_width=3, color="#b71c1c")

# Colocando as legendas
p3.legend.click_policy="hide"
p3.legend.location = "top_left"

p4.legend.click_policy="hide"
p4.legend.location = "top_right"

show(column(p3,p4))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.9.7 Análise de variação do R_0

[67]:

```
x = range(start_day, len(I))
beta_norm = [b*p for b,p in zip(saved_param['beta'],saved_param['pop'])]
Ro = [b*p/r for b,r,p in zip(saved_param['beta'],saved_param['r'],saved_param['pop'])]


fig1 = go.Figure()

fig1.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=beta_norm[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="β = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig1.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=beta_norm[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="β = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[beta_norm[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um β = %{y:.4f}."))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[beta_norm[-1]],
                    marker_symbol="triangle-right-dot", name="Valor de converg^encia",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> converge para β = %
↪{y:.4f}."))
fig1.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.10, y=1.05),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro β',
                   title_text="Estimativa do par^ametro β -  " + country['data']['name
↪'])

fig1.show(renderer="png")
```



[68]:

```
fig2 = go.Figure()

fig2.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=saved_param['r'][:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig2.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=saved_param['r'][peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig2.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[saved_param['r'][peak_pos-start_
↪day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um r = %{y:.4f}."))
fig2.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[saved_param['r'][-1]],
```

(continues on next page)

(continued from previous page)

```
                    marker_symbol="triangle-right-dot", name="Valor de converg^encia",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> converge para r = %
↪{y:.4f}."))
fig2.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.07, y=1.06),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro r',
                   title_text="Estimativa do par^ametro r -  " + country['data']['name
↪'])

fig2.show(renderer="png")
```

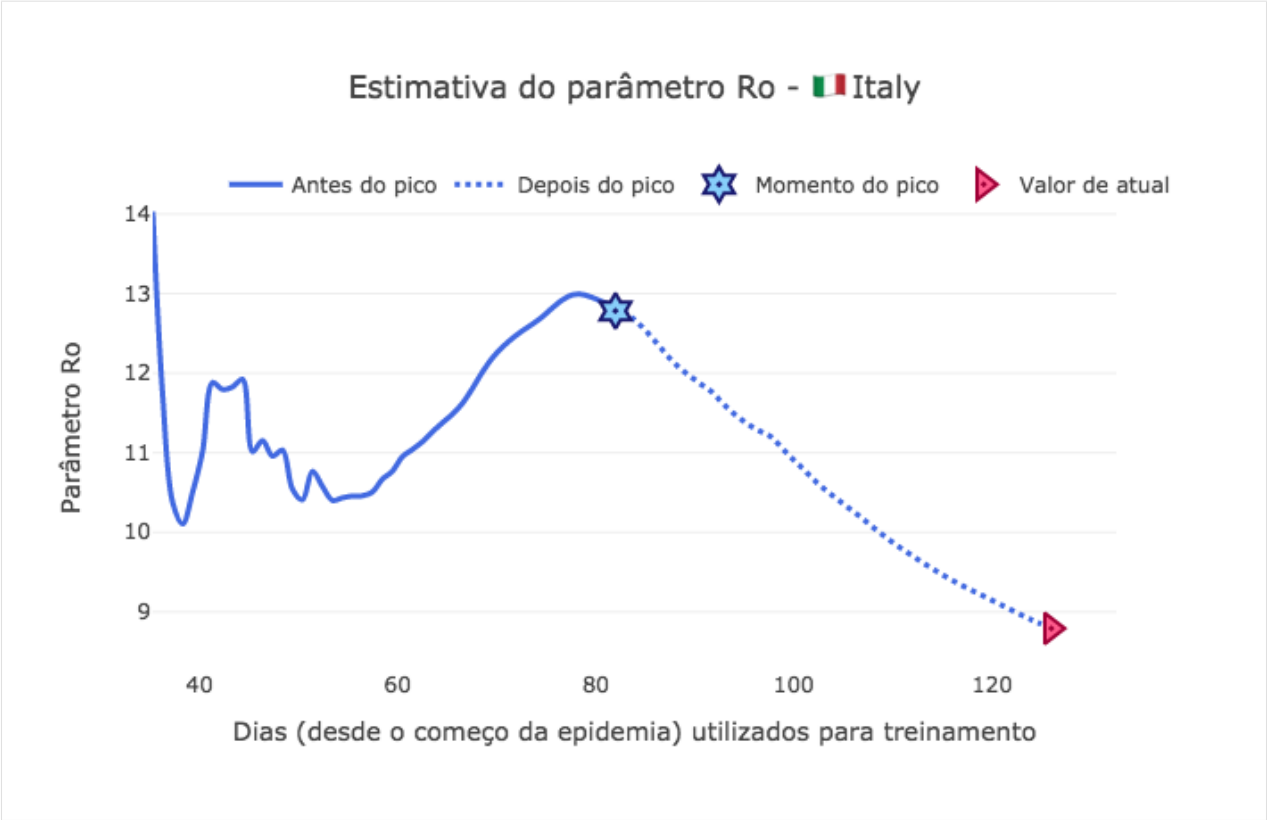
[69]:

```
fig3 = go.Figure()

fig3.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=Ro[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig3.add_trace(go.Scatter(
```

(continues on next page)

(continued from previous page)

```
                    x=td[peak_pos:-1],
                    y=Ro[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="Ro = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig3.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[Ro[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um Ro = %{y:.4f}."))
fig3.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[Ro[-1]],
                    marker_symbol="triangle-right-dot", name="Valor de atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para Ro␣
↪= %{y:.4f}."))
fig3.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.07, y=1.06),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro Ro',
                   title_text="Estimativa do par^ametro Ro -  " + country['data']['name
↪'])

fig3.show(renderer="png")
```



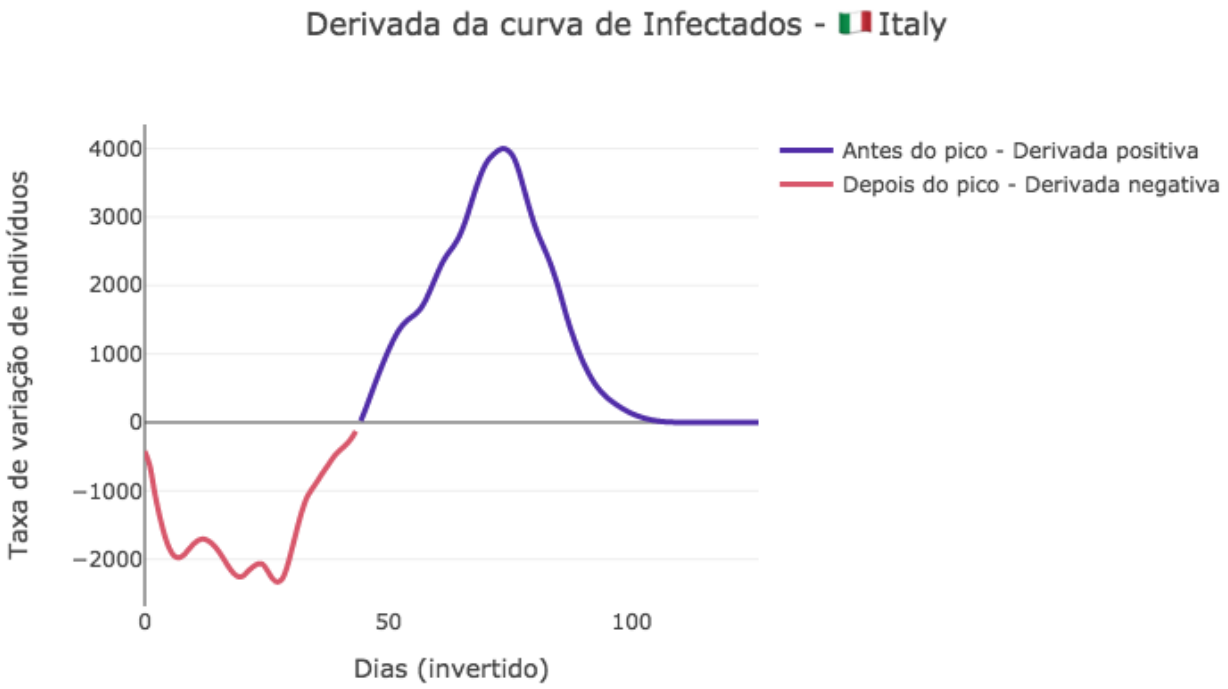
3.9.8 Análise de confiança

[70]:
```
from PyAstronomy import pyasl

dI = np.gradient(pyasl.smooth(I, 13, "hamming"))
t = np.linspace(0, len(dI), len(dI))

signal = np.array([di >= 0 for di in dI[::-1]])


import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(
                    x=t[signal],
                    y=dI[::-1][signal],
                    mode='lines',
                    name='Antes do pico - Derivada positiva',
                    line_shape='spline',
                    line = dict(color='#512da8', width=3)))
fig.add_trace(go.Scatter(
                    x=t[~signal],
                    y=dI[::-1][~signal],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico - Derivada negativa',
```

(continues on next page)

(continued from previous page)

```
                    line = dict(color='#d8576b', width=3)))

fig.update_layout(template='xgridoff',
                  xaxis_title='Dias (invertido)',
                  yaxis_title='Taxa de variação de indivíduos',
                  title_text="Derivada da curva de Infectados -  " + country['data']['
↪'name'])
# 'ggplot2', 'seaborn', 'simple_white', 'plotly',
# 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
# 'ygridoff', 'gridon', 'none'

fig.show(renderer="png")
```



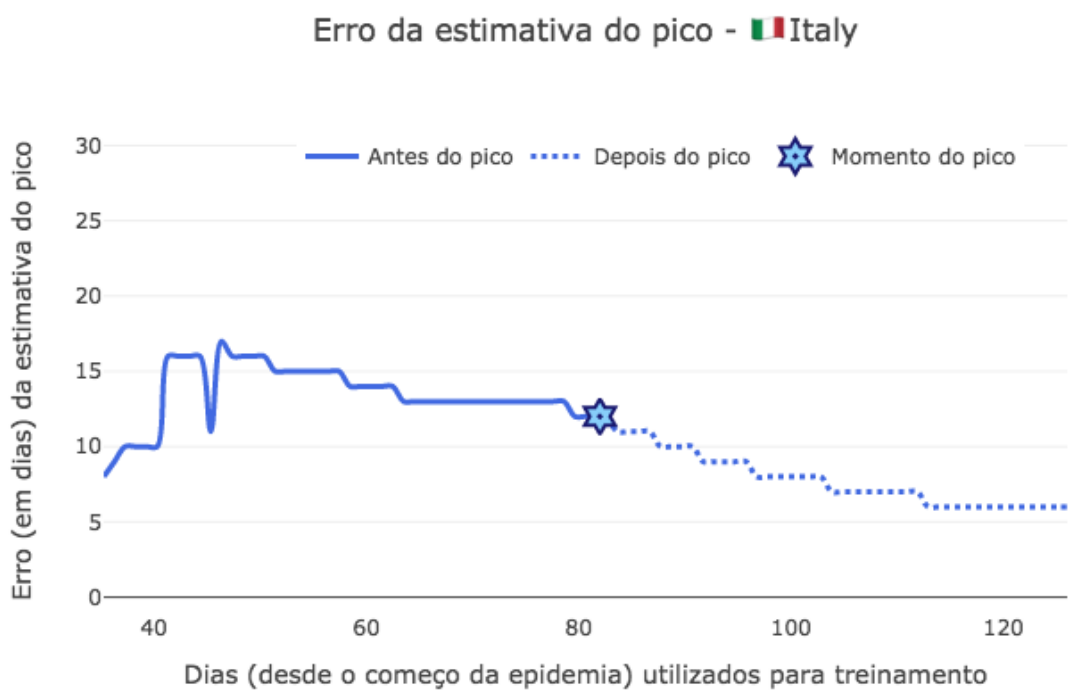
[22]:

```
peak_pos = len(Id) - np.argmax(signal)
print("O pico da epidemia acontece no dia", peak_pos)

start_day = 35
final_day = int(td[-1])

estimated_peaks = []
for data in saved_prediction:
    dI = np.gradient(data[1][:final_day])
    signal_pred = np.array([di >= 0 for di in dI[::-1]])
    estimated_peaks.append(len(Id) - np.argmax(signal_pred))
estimated_peaks = np.array(estimated_peaks)
```

```
O pico da epidemia acontece no dia 82
```

[47]:

```
import plotly.graph_objects as go

peak_error = np.abs(estimated_peaks - peak_pos)

fig1 = go.Figure()
fig1.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=peak_error[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="Erro de %{y} dias, <br> com %{x:.0f} dias de dados.
↪"))
fig1.add_trace(go.Scatter(
                    x=td[peak_pos:],
                    y=peak_error[peak_pos-start_day:],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="Erro de %{y} dias, <br> com %{x:.0f} dias de dados.
↪"))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[peak_error[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x} depois do começo da epidemia."))

fig1.update_layout(yaxis_range=[-1,31], template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.20, y=1.0),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Erro (em dias) da estimativa do pico',
                   title_text="Erro da estimativa do pico -  " + country['data']['name
↪'])
# 'ggplot2', 'seaborn', 'simple_white', 'plotly',
# 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
# 'ygridoff', 'gridon', 'none'

fig1.show(renderer="png")
```



3.10 SIR Alemanha - Dados Reais

3.10.1 Os dados

Para coletar os dados de outros países, foram utilizadas as APIs dos sites:

- About Corona
- Europe RestFul

[1]:
```
import requests
import pandas as pd

covid_api = 'https://corona-api.com/countries/'
rest_countries = 'https://restcountries.eu/rest/v2/alpha/'
country = 'DE' # Alpha-2 ISO3166

data_json =  requests.get(covid_api + country).json()
country = requests.get(covid_api + country).json()

N = country['data']['population']

print(country['data']['name'])
```

```
Germany
```

Organizando os dados

[2]:
```
from datetime import datetime

df = pd.DataFrame(data_json['data']['timeline'])
df = df.sort_values('date').reset_index()

from datetime import datetime, timedelta
df['date'] = [datetime.fromisoformat(f) for f in df['date']]
df = df.drop_duplicates(subset='date', keep = 'last')

# Criando o vetor de tempo
first_date = df['date'].iloc[0]
size_days = (df['date'].iloc[-1] - df['date'].iloc[0]).days
date_vec = [first_date + timedelta(days=k) for k in range(size_days)]

new_df = pd.DataFrame(date_vec, columns=['date'])
new_df = pd.merge(new_df, df, how='left', on= 'date')
new_df = new_df.drop(columns= ['index',  'updated_at', 'is_in_progress'])

for col in new_df.columns[1:]:
    new_df[col] = new_df[col].interpolate(method='polynomial', order=1)
df = new_df.dropna()

df.head()
```

[2]:
```
        date  deaths  confirmed  active  recovered  new_confirmed  \
0 2020-01-27     0.0        4.0     4.0        0.0            4.0
1 2020-01-28     0.0        4.0     4.0        0.0            0.0
2 2020-01-29     0.0        4.0     4.0        0.0            0.0
3 2020-01-30     0.0        5.0     5.0        0.0            1.0
4 2020-01-31     0.0        8.0     8.0        0.0            3.0

   new_recovered  new_deaths
0            0.0         0.0
1            0.0         0.0
2            0.0         0.0
3            0.0         0.0
4            0.0         0.0
```

Visualizando os dados

[3]:
```
from bokeh.models   import Legend, ColumnDataSource, RangeTool, LinearAxis, Range1d,␣
↪HoverTool
from bokeh.palettes import brewer, Inferno256
from bokeh.plotting import figure, show
from bokeh.layouts  import column
from bokeh.io       import output_notebook

output_notebook()

import numpy as np
```

(continues on next page)

(continued from previous page)

```
# Criando os valores para legenda no plot
year =  [str(int(d.year)) for d in df['date'] ]
month = [("0"+str(int(d.month)))[-2:] for d in df['date'] ]
day =   [("0"+str(int(d.day)))[-2:] for d in df['date'] ]

# Criando a fonte de dados
source = ColumnDataSource(data={
    'Data'       : df['date'].values,
    'd': day, 'm': month, 'y': year,
    'Infectados Acc' : df['confirmed'].values,
    'Mortes'     : df['deaths'].values,
    'Ativo'     : df['active'].values,
    'Recuperados': df['recovered'].values
})


# Criando a figura
p = figure(plot_height=500,
           plot_width=600,
           x_axis_type="datetime",
           tools="",
           #y_axis_type="log",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

# Incluindo as curvas
i_p = p.line(x='Data', y='Ativo',
             legend_label="Infectados",
             line_cap="round", line_width=5, color="#c62828", source=source)
m_p = p.line(x='Data', y='Mortes',
             legend_label="Mortes",
             line_cap="round", line_width=5, color="#512da8", source=source)
c_p = p.line(x='Data', y='Infectados Acc',
             legend_label="Infectados Acc",
             line_cap="round", line_width=5, color="#0288d1", source=source)
r_p = p.line(x='Data', y='Recuperados',
             legend_label="Recuperados",
             line_cap="round", line_width=5, color="#388e3c", source=source)

# Colocando as legendas
p.legend.click_policy="hide"
# p.legend.location = "top_left"
p.legend.location = "top_left"

# Incluindo a ferramenta de hover
p.add_tools(HoverTool(
    tooltips=[
        ( 'Indivíduos', '$y{i}'),
        ( 'Data',       '@d/@m/@y' ),
    ],
```

(continues on next page)

(continued from previous page)

```
    renderers=[
        m_p, i_p, c_p, r_p
    ]
))

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

Verificando os dados

[4]:

```
dif_I = np.diff(df['active'])
cum = []
cum.append(dif_I[0])

for k, i in enumerate(dif_I):
    cum.append(cum[-1] + i)

cum = np.array(cum)
cum += df['deaths'].to_numpy() + df['recovered'].to_numpy()

print("Erro entre casos acumulados e valores de confirmados: {}".format(
        round(sum((cum - df['confirmed'].values)**2 / len(cum)),2) ) )
```

```
Erro entre casos acumulados e valores de confirmados: 16.0
```

Criando os dados SIR

[5]:

```
I = df['active'].to_numpy()
R = df['recovered'].to_numpy()
M = df['deaths'].to_numpy()
S = N - R - I

# Creating the time vector
t = np.linspace(0, len(I), len(I))


Sd, Id, Md, Rd, td = S, I, M, R, t
```

3.10.2 Estimando utilizando todos os dados

[6]:
```
from models import *

dataset = dict(S=Sd, I=Id, R=Rd)

# Create the model
sir_model = ss.SIR(pop=N, focus=["S", "I", "R"])

# Adjust the parameters
sir_model.fit(dataset, td,
              search_pop=True,
              pop_sens=[0.001,0.05],
              beta_sens=[100,1000],
              r_sens=[100,100])


# Predict the model
sim_res = sir_model.predict((Sd[0],Id[0],Rd[0]), td)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

```
         ├─ S(0) — I(0) — R(0) —  [81802253.0, 4.0, 0.0]
         ├─ beta —   1   r —   0.14285714285714285
         ├─ beta bound —   0.01  —  1000
         ├─ r bound —   0.0014285714285714286  —  14.285714285714285
         ├─ equation weights —   [1.223645216108376e-08, 4.8013887093497817e-05, 1.
↪7124646680667163e-05]
         ├─ Running on —  differential_evolution SciPy Search Algorithm
         └─ Defined at:  105.13818075597061  —  0.05759270422192064
```

[7]:
```
print("Par^ametros estimados: ", sir_model.parameters)
print("Suposto Ro: ", sir_model.parameters[0] * sir_model.parameters[-1] / sir_model.
↪parameters[1])
print("Dias contaminados: ", 1 / sir_model.parameters[1])
```

```
Par^ametros estimados:  [1.05138181e+02 5.75927042e-02 2.08772990e-03]
Suposto Ro:  3.811248776553262
Dias contaminados:  17.36331039686421
```

[19]:
```
p = figure(plot_height=500,
           plot_width=800,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - " + country['data']['name'])

# Preparando o estilo
p.grid.grid_line_alpha = 0
p.ygrid.band_fill_color = "olive"
```

(continues on next page)

(continued from previous page)

```
p.ygrid.band_fill_alpha = 0.1
p.yaxis.axis_label = "Indivíduos"
p.xaxis.axis_label = "Dias"

p.line(t, I,
       legend_label="Infectados", color="#ff6659", line_width=4)
p.line(t, R,
       legend_label="Removidos", color="#76d275", line_width=4)

# Show the results
p.line(td, sim_res[1],
       legend_label="Infectados - Modelo", line_dash="dashed", color="#d32f2f", line_
↪width=3)
p.line(td, sim_res[2],
    legend_label="Removidos - Modelo", line_dash="dashed", color="#43a047", line_
↪width=3)
p.line(td, sim_res[0],
      legend_label="Suscetíveis Ponderados - Modelo", line_dash="dashed", color="
↪#1e88e5", line_width=3)
p.add_layout(p.legend[0], 'right')

show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.10.3 Monte Carlo

Nesta parte, faremos um teste aumentando a quantidade de amostras de treinamento e prevendo o momento do pico da epidemia a medida que mais dias são utilizados para treinamento. Esse estudo vai possibilitar a análise da certeza da previsão do pico da epidemia antes desse acontecer.

[12]:

```
saved_param = {'r':[], 'beta':[], 'pop':[]}
saved_prediction = []

start_day = 45
pred_t = np.array(range(int(td[-1]) + 120))

for i in range(start_day, len(I)):

    dataset = dict(S=Sd[:i], I=Id[:i], R=Rd[:i])
    td_ = td[:i]

    # Create the model
    sir_model = ss.SIR(pop=N, focus=["S", "I", "R"], verbose = False)

    # Adjust the parameters
    sir_model.fit(dataset, td_,
                  search_pop=True,
                  pop_sens=[0.001,0.04],
                  beta_sens=[100,1000],
```

(continues on next page)

(continued from previous page)

```
                    r_sens=[100,100])

    saved_param['beta'].append(sir_model.parameters[0])
    saved_param['r'].append(sir_model.parameters[1])
    saved_param['pop'].append(sir_model.parameters[2])

    saved_prediction.append(sir_model.predict((Sd[0],Id[0], Rd[0]), pred_t))
```

[13]:
```
import pickle

with open("./Germany_mc_runs.pickle", "wb") as handle:
    pickle.dump({"pars":saved_param, "pred":saved_prediction}, handle)
```

3.10.4 Análise do uso do sistema de saúde

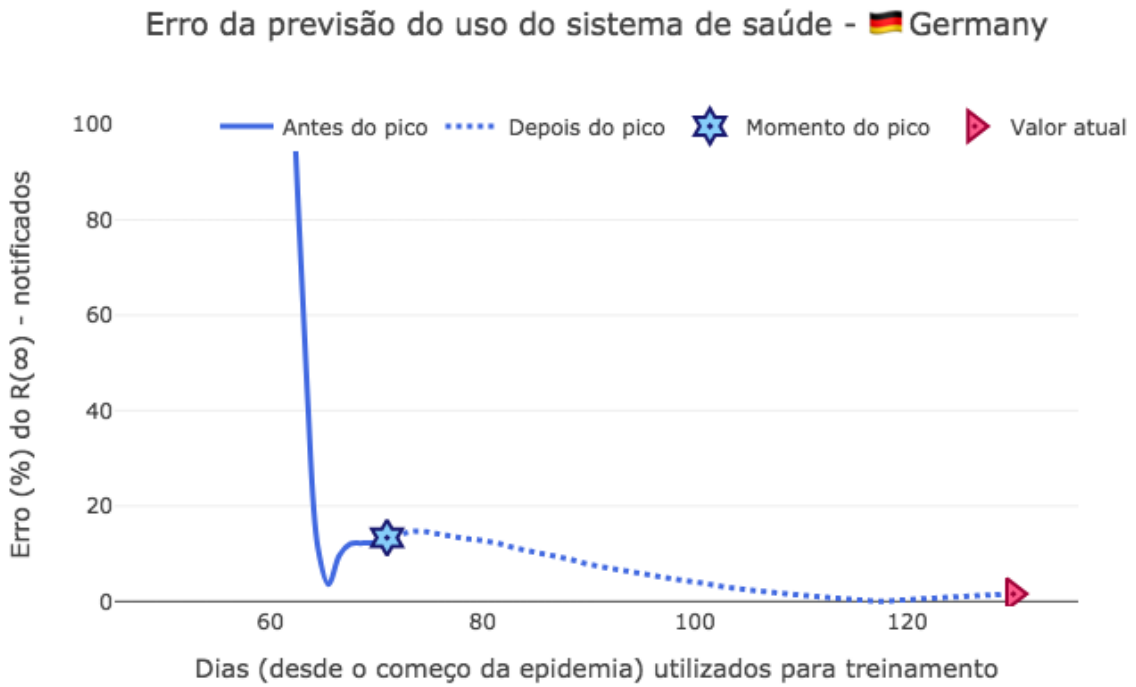
Nesta análise, mostramos o erro percentual do quanto antes do pico, conseguimos prever a quantidade de pessoas que realmente serão identificadas como infectadas, uma vez que $R(\infty)$ é a quantidade de pessoas recuperadas totais, daquelas que foram noficadas como infectadas no sistema de saúde. Desta forma segue o erro proporcional do erro a medida em que novos dados diários foram incluidos no modelo:

[41]:
```
x = range(start_day, len(I))
usage_error = [ 100 * abs(p*N - Rd[-1]) / Rd[-1] for p in saved_param['pop']]

fig9 = go.Figure()

fig9.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=usage_error[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="ϵ(%) = %{y:.0f}, <br> com %{x:.0f} dias de dados."))
fig9.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=usage_error[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="ϵ(%) = %{y:.0f}, <br> com %{x:.0f} dias de dados."))
fig9.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[usage_error[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um ϵ(%) = %{y:.0f}."))
fig9.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[usage_error[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
```

(continues on next page)

(continued from previous page)

```
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para ϵ(%)␣
↪= %{y:.4f}."))
fig9.update_layout(template='xgridoff', yaxis_range=[-1,100],
                   legend_orientation="h", legend=dict(x=0.10, y=1.05),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Erro (%) do R(∞) - notificados',
                   title_text="Erro da previsão do uso do sistema de saúde -  " +␣
↪country['data']['name'])

fig9.show(renderer="png")
```



3.10.5 Visualizando as previsões de $I(t)$

Vamos analisar as previsões quando somente os dados antes do pico são fornecidos ao modelo, e as previões utilizando os dados após o pico:

[17]:

```
visual_peak = 70

p2 = figure(plot_height=500,
            plot_width=600,
            tools="",
            toolbar_location=None,
            title="Evolução do COVID - " + country['data']['name'])
```

(continues on next page)

(continued from previous page)

```
# Preparando o estilo
p2.grid.grid_line_alpha = 0
p2.ygrid.band_fill_color = "olive"
p2.ygrid.band_fill_alpha = 0.1
p2.yaxis.axis_label = "Indivíduos"
p2.xaxis.axis_label = "Dias"

# Incluindo as curvas
p2.line(td, Id,
       legend_label="Infectados",
       line_cap="round", line_width=3, color="#c62828")

for data in saved_prediction[visual_peak-start_day:]:
    p2.line(pred_t[:len(td)], -data[1][:len(td)],
           legend_label="Previsão Infectados - Depois do pico",
           line_cap="round", line_dash="dashed", line_width=4, color="#ffa000", line_
↪alpha = 0.1)

for data in saved_prediction[20:visual_peak-start_day]:
    p2.line(pred_t[:len(td)], -data[1][:len(td)],
           legend_label="Previsão Infectados - Antes do pico",
           line_cap="round", line_dash="dashed", line_width=4, color="#42a5f5", line_
↪alpha = 0.1)

# Colocando as legendas
p2.legend.click_policy="hide"
p2.legend.location = "bottom_left"

show(p2)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.10.6 Visualizando os ajustes dos grupos

Aqui vamos analisar as previsões obtidas para cada grupo $S(t)$, $I(t)$ e $R(t)$, a medida que mais dias foram informados ao modelo.

[33]:

```
p3 = figure(plot_height=350,
           plot_width=600,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - I(t) e R(t) - " + country['data']['name'])

p4 = figure(plot_height=350,
           plot_width=600,
           tools="",
           toolbar_location=None,
           title="Evolução do COVID - S(t) - " + country['data']['name'])
```

(continues on next page)

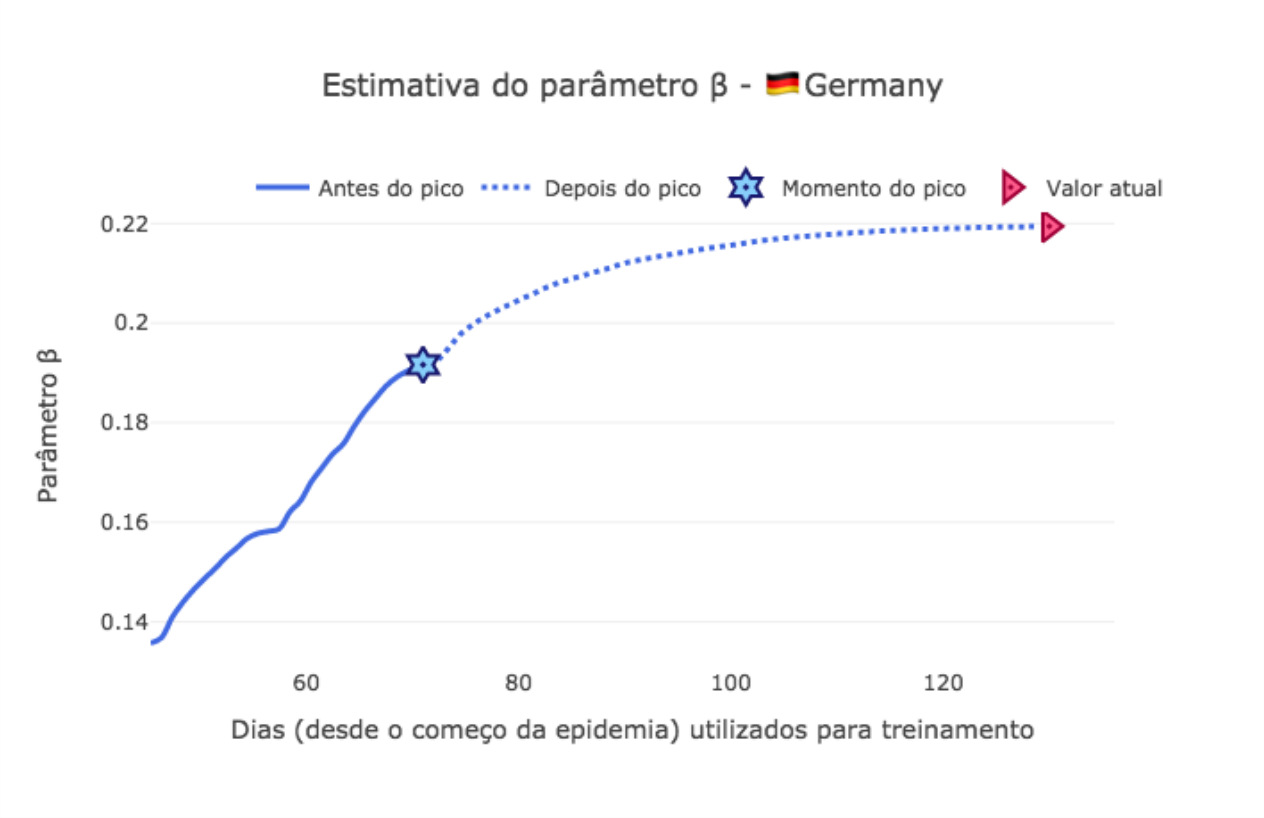
(continued from previous page)

```
plot_all = True

# Preparando o estilo
p3.grid.grid_line_alpha = 0
p3.ygrid.band_fill_color = "olive"
p3.ygrid.band_fill_alpha = 0.1
p3.yaxis.axis_label = "Indivíduos"
p3.xaxis.axis_label = "Dias"

p4.grid.grid_line_alpha = 0
p4.ygrid.band_fill_color = "olive"
p4.ygrid.band_fill_alpha = 0.1
p4.yaxis.axis_label = "Indivíduos"
p4.xaxis.axis_label = "Dias"

# Incluindo as curvas
for data in saved_prediction[20:]:
    p3.line(pred_t, data[1],
           legend_label="Previsão Infectados",
           line_cap="round", line_dash = 'dashed',
           line_width=4, color="#42a5f5", line_alpha = 0.1)

    p3.line(pred_t, data[2],
           legend_label="Previsão Recuperados",
           line_cap="round", line_dash = 'dashed', line_width=4,
           color="#9c27b0", line_alpha = 0.07)


p3.line(td, Id,
       legend_label="Infectados",
       line_cap="round", line_width=3, color="#005cb2")

p3.line(td, Rd,
           legend_label="Recuperados",
           line_cap="round", line_width=3, color="#5e35b1")

if plot_all:
    for data in saved_prediction:
        p4.line(pred_t, data[0] + N*(1-saved_param['pop'][-1]),
               legend_label="Previsão Suscetiveis",
               line_cap="round", line_dash = 'dashed',
               line_width=4, color="#ff5722", line_alpha = 0.07)
    p4.line(td, Sd,
           legend_label="Suscetiveis",
           line_cap="round", line_width=3, color="#b71c1c")

# Colocando as legendas
p3.legend.click_policy="hide"
p3.legend.location = "top_left"

p4.legend.click_policy="hide"
p4.legend.location = "top_right"

show(column(p3,p4))
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

3.10.7 Análise de variação do R_0

[42]:
```
x = range(start_day, len(I))
beta_norm = [b*p for b,p in zip(saved_param['beta'],saved_param['pop'])]
Ro = [b*p/r for b,r,p in zip(saved_param['beta'],saved_param['r'],saved_param['pop'])]


fig1 = go.Figure()

fig1.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=beta_norm[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="β = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig1.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=beta_norm[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="β = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[beta_norm[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um β = %{y:.4f}."))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[beta_norm[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para β =
↪%{y:.4f}."))
fig1.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.10, y=1.05),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro β',
                   title_text="Estimativa do par^ametro β -  " + country['data']['name
↪'])

fig1.show(renderer="png")
```



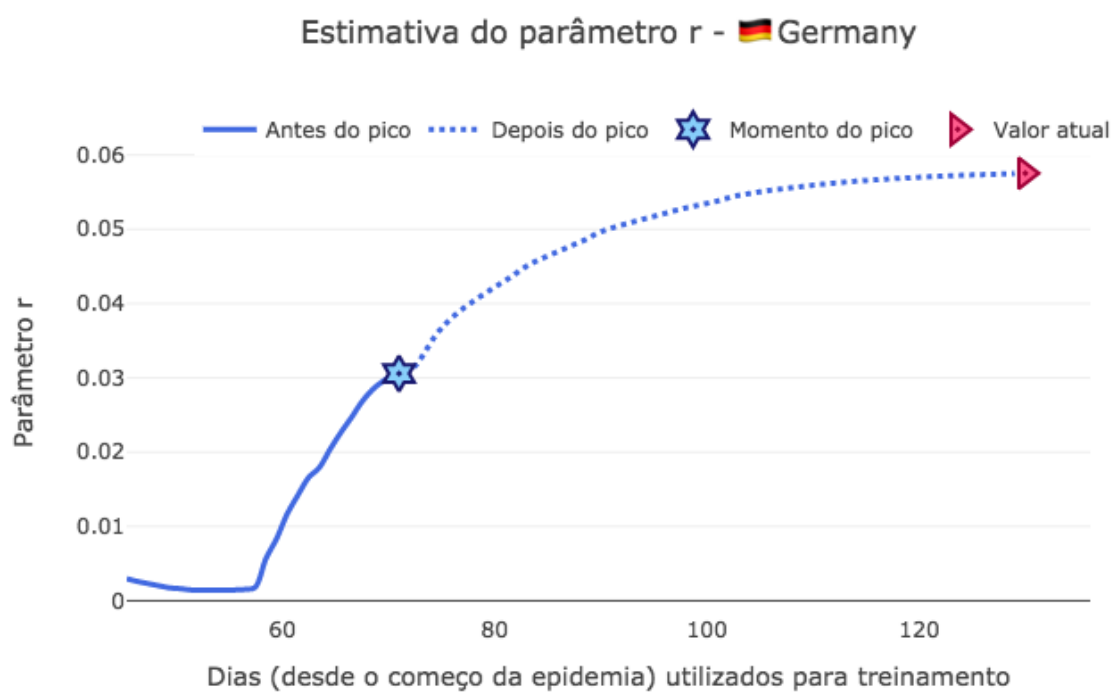
[43]:

```
fig2 = go.Figure()

fig2.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=saved_param['r'][:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig2.add_trace(go.Scatter(
                    x=td[peak_pos:-1],
                    y=saved_param['r'][peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig2.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[saved_param['r'][peak_pos-start_
↪day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um r = %{y:.4f}."))
fig2.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[saved_param['r'][-1]],
```

(continues on next page)

(continued from previous page)

```
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para r =
↪%{y:.4f}."))
fig2.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.07, y=1.06),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro r',
                   title_text="Estimativa do par^ametro r -  " + country['data']['name
↪'])

fig2.show(renderer="png")
```



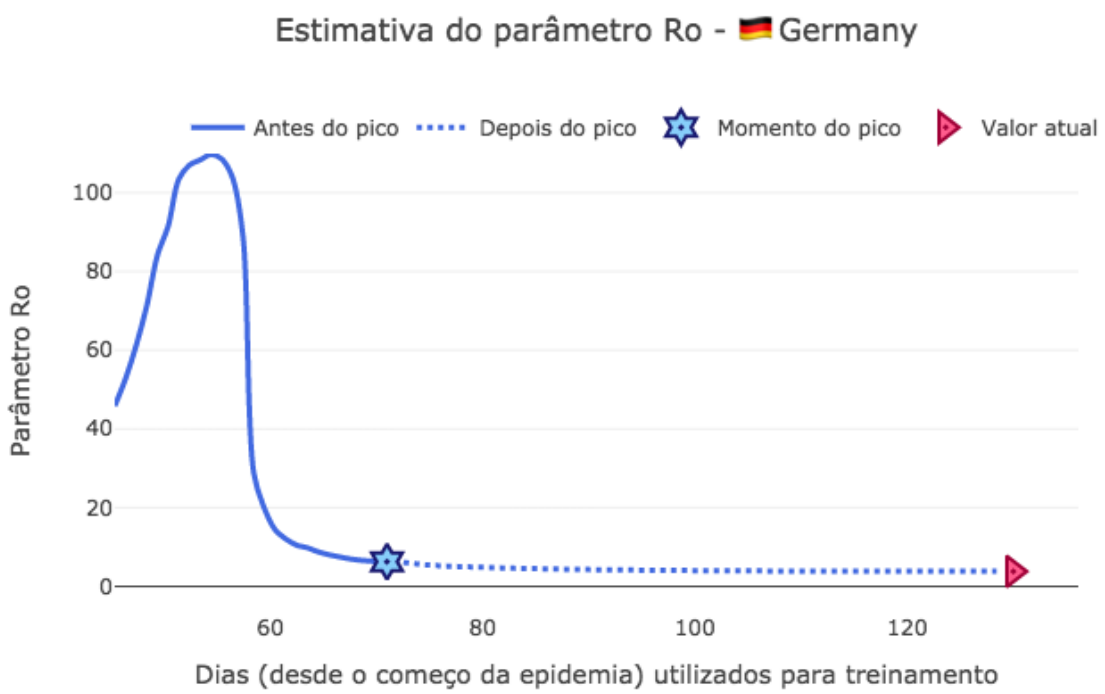
[44]:

```
fig3 = go.Figure()

fig3.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=Ro[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="r = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig3.add_trace(go.Scatter(
```

(continues on next page)

(continued from previous page)

```
                    x=td[peak_pos:-1],
                    y=Ro[peak_pos-start_day:-1],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="Ro = %{y:.4f}, <br> com %{x:.0f} dias de dados."))
fig3.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[Ro[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x}, com um Ro = %{y:.4f}."))
fig3.add_trace(go.Scatter(
                    mode="markers", x=[td[-1]], y=[Ro[-1]],
                    marker_symbol="triangle-right-dot", name="Valor atual",
                    marker_line_color="#a00037", marker_color="#ff5c8d",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="No dia %{x} da epidemia, <br> convergindo para Ro␣
↪= %{y:.4f}."))
fig3.update_layout(template='xgridoff',
                   legend_orientation="h", legend=dict(x=0.07, y=1.06),
                   xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                   yaxis_title='Par^ametro Ro',
                   title_text="Estimativa do par^ametro Ro -  " + country['data']['name
↪'])

fig3.show(renderer="png")
```



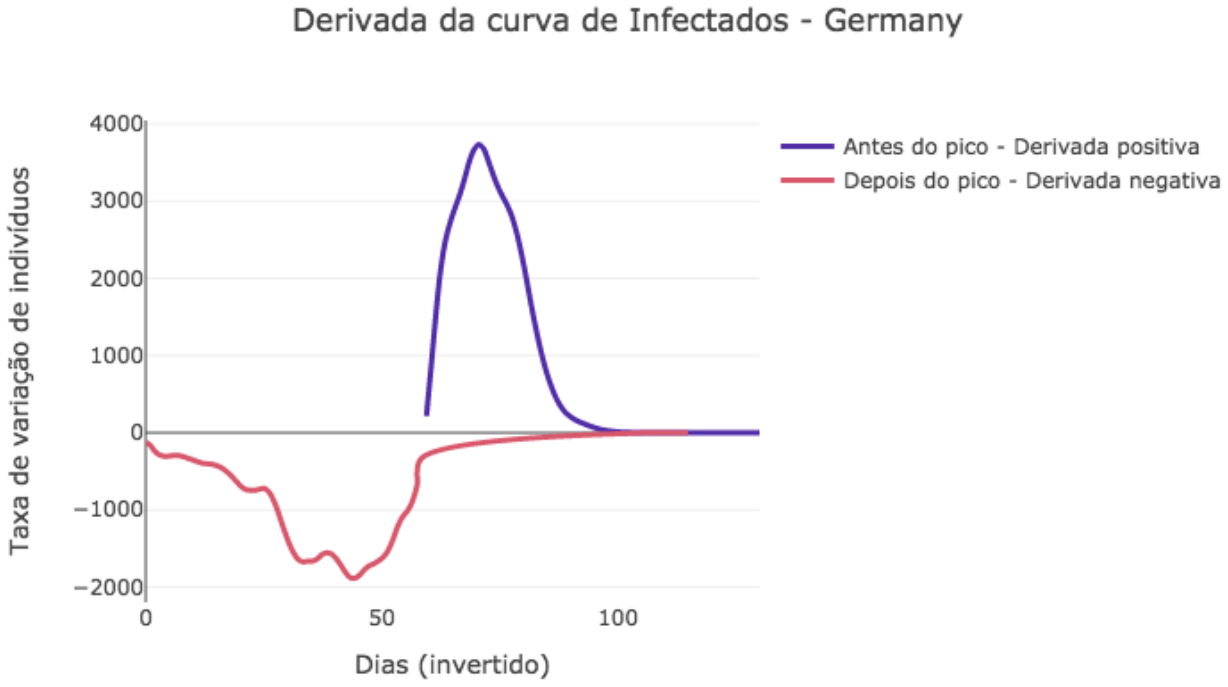
3.10.8 Análise de confiança

[22]:
```
from PyAstronomy import pyasl

dI = np.gradient(pyasl.smooth(I, 13, "hamming"))
t = np.linspace(0, len(dI), len(dI))

signal = np.array([di >= 0 for di in dI[::-1]])


import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(
                    x=t[signal],
                    y=dI[::-1][signal],
                    mode='lines',
                    name='Antes do pico - Derivada positiva',
                    line_shape='spline',
                    line = dict(color='#512da8', width=3)))
fig.add_trace(go.Scatter(
                    x=t[~signal],
                    y=dI[::-1][~signal],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico - Derivada negativa',
```

(continues on next page)

(continued from previous page)

```
                    line = dict(color='#d8576b', width=3)))

fig.update_layout(template='xgridoff',
                  xaxis_title='Dias (invertido)',
                  yaxis_title='Taxa de variação de indivíduos',
                  title_text="Derivada da curva de Infectados - " + country['data']['name'])
# 'ggplot2', 'seaborn', 'simple_white', 'plotly',
# 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
# 'ygridoff', 'gridon', 'none'

fig.show(renderer="png")
```



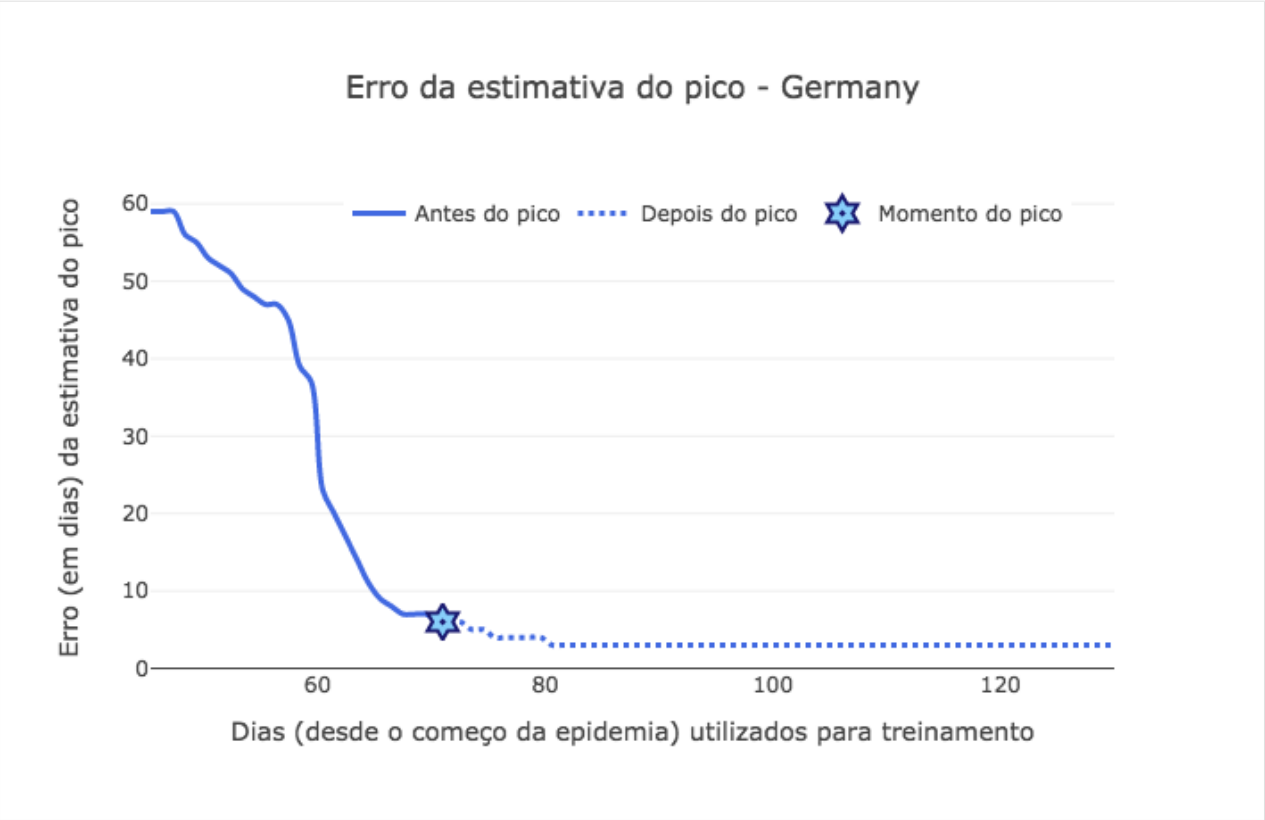
[23]:

```
peak_pos = len(Id) - np.argmax(signal)
print("O pico da epidemia acontece no dia", peak_pos)

final_day = int(td[-1])

estimated_peaks = []
for data in saved_prediction:
    dI = np.gradient(data[1][:final_day])
    signal_pred = np.array([di >= 0 for di in dI[::-1]])
    estimated_peaks.append(len(Id) - np.argmax(signal_pred))
estimated_peaks = np.array(estimated_peaks)
```

```
O pico da epidemia acontece no dia 71
```

[24]:

```
import plotly.graph_objects as go

peak_error = np.abs(estimated_peaks - peak_pos)

fig1 = go.Figure()
fig1.add_trace(go.Scatter(
                    x=td[start_day:peak_pos],
                    y=peak_error[:peak_pos-start_day],
                    mode='lines',
                    name='Antes do pico',
                    line_shape='spline',
                    line = dict(color='royalblue', width=3),
                    hovertemplate="Erro de %{y} dias, <br> com %{x:.0f} dias de dados.
↪"))
fig1.add_trace(go.Scatter(
                    x=td[peak_pos:],
                    y=peak_error[peak_pos-start_day:],
                    mode='lines',
                    line_shape='spline',
                    name='Depois do pico',
                    line = dict(color='royalblue', width=3, dash='dot'),
                    hovertemplate="Erro de %{y} dias, <br> com %{x:.0f} dias de dados.
↪"))
fig1.add_trace(go.Scatter(
                    mode="markers", x=[peak_pos], y=[peak_error[peak_pos-start_day]],
                    marker_symbol="hexagram-dot", name="Momento do pico",
                    marker_line_color="midnightblue", marker_color="lightskyblue",
                    marker_line_width=2, marker_size=15,
                    hovertemplate="Pico no dia %{x} depois do começo da epidemia."))

fig1.update_layout( template='xgridoff',
                  legend_orientation="h", legend=dict(x=0.20, y=1.0),
                  xaxis_title='Dias (desde o começo da epidemia) utilizados para␣
↪treinamento',
                  yaxis_title='Erro (em dias) da estimativa do pico',
                  title_text="Erro da estimativa do pico - " + country['data']['name
↪'])
# 'ggplot2', 'seaborn', 'simple_white', 'plotly',
# 'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
# 'ygridoff', 'gridon', 'none'

fig1.show(renderer="png")
```



3.11 epidemicModels

3.11.1 models package

Stochastic Search - Learning models

class models.stochastic_search.**SIR**(*pop=2000000, focus=['I', 'R'], algorithm='differential_evolution', simulation='discrete', stochastic_search=False, forced_search_pop=False, ode_full_output=False, verbose=True*)

Bases: object

This model concentrate all the developed algorithms to represent data driven SIR model. Using the Scipy default model structure.

cost_wrapper(**args*)

The method responsible for wrapping the cost function. This allows differential evolution algorithm to run with parallel processing.

Parameters ***args** (*tuple*) – cost function parameters

Returns the cost function outputs

Return type float

fit(*dataset, t, search_pop=True, Ro_bounds=None, pop_sens=[0.001, 0.0001], Ro_sens=[0.8, 15], D_sens=[5, 50], sigma_sens=None, mu_sens=[0.0001, 0.02], notified_sens=None, sample_ponder=None, optim_verbose=False, **kwargs*)

The method responsible for estimating a set of beta and r parameters for the provided data set. It assumes that in the data there is only one epidemic period.

Parameters

- **dataset** (*array*) – list with the respective arrays of Suceptible, Infected, Recovered and Deaths.
- **t** (*array*) – The time respective to each set of samples.
- **search_pop** (*bool*) – Flag to set the exposed population search, for better Suceptible extimation values. Default is `True`.
- **Ro_bounds** (*list*) – The bounds to build the constraints for `Ro = Beta / r`. With minimun and maximun values, respectivelly.
- **pop_sens** (*list*) – The sensibility (boudaries) for the proportion of `N` to be found by the pop parameters.
- **beta_sens** (*list*) – The beta parameter sensibility minimun and maximun boundaries, respectivelly. Default is `[100,100]`.
- **r_sens** (*list*) – The r parameter sensibility minimun and maximun boundaries, respectivelly. Default is `[100,1000]`.
- **sample_ponder** (*bool*) – The flag to set the pondering of the non informative recovered data.
- **optim_verbose** (*bool*) – If `True`, after fitting will show the optimization summary.
- ****kwargs** (*dict*) – The optimization search algorithms options.

fit_multiple(*Sd, Id, Bd, td, threshold_prop=1, cases_before=10, filt_estimate=False, filt_window=55, beta_sens=[100, 10], r_sens=[100, 10], out_type=0, **kwargs*)

The method responsible for estimating a set of beta and r parameters for each epidemy period existent in the provided dataset. It assumes that in the data there are several epidemic periods.

Parameters

- **Sd** (*array*) – Array with the suceptible data.
- **Id** (*array*) – Array with the infected data.
- **Bd** (*array*) – Array with the births data.
- **td** (*array*) – The time respective to each set of samples.
- **threshold_prop** (*float*) – The standard deviation proportion used as threshold for windowing. Default is `1.0`.
- **cases_before** (*int*) – The number of back samples to check for the initial window point. Default is `10`.
- **filt_estimate** (*bool*) – Flag to use filtered data to estimate the model parameters. Default is `False`.
- **filt_window** (*int*) – The window size used on the filtering technique, only if `filt_estimate=True`. Default is `55`.
- **beta_sens** (*list*) – The beta parameter sensibility minimun and maximun boundaries, respectivelly. Default is `[100,100]`.
- **r_sens** (*list*) – The r parameter sensibility minimun and maximun boundaries, respectivelly. Default is `[100,1000]`.
- **out_type** (*int*) – The output type, it can be `1` or `0`. Default is `0`.

Returns If the `out_type=0`, it returns a tuple with the estimated beta and r, estimated, with the year of each respective window. If *out_type=1* it returns the self.data of the model, a summary with all model information.

Return type tuple

monteCarlo_multiple(*Sd, Id, Bd, td, threshold_prop=1, cases_before=10, minimum_days=60, steps_indays=10, filt_estimate=False, filt_window=55, beta_sens=[1000, 10], r_sens=[1000, 10], out_type=0, **kwargs*)

The method responsible for estimating a set of beta and r parameters for each epidemy period existent in the provided dataset. It assumes that in the data there are several epidemic periods.

Parameters

- **Sd** (*array*) – Array with the suceptible data.
- **Id** (*array*) – Array with the infected data.
- **Bd** (*array*) – Array with the births data.
- **td** (*array*) – The time respective to each set of samples.
- **threshold_prop** (*float*) – The standard deviation proportion used as threshold for windowing. Default is `1.0`.
- **cases_before** (*int*) – The number of back samples to check for the initial window point. Default is `10`.
- **filt_estimate** (*bool*) – Flag to use filtered data to estimate the model parameters. Default is `False`.
- **filt_window** (*int*) – The window size used on the filtering technique, only if `filt_estimate=True`. Default is `55`.
- **beta_sens** (*list*) – The beta parameter sensibility minimun and maximun boundaries, respectivelly. Default is `[100,100]`.
- **r_sens** (*list*) – The r parameter sensibility minimun and maximun boundaries, respectivelly. Default is `[100,1000]`.
- **out_type** (*int*) – The output type, it can be `1` or `0`. Default is `0`.

Returns If the `out_type=0`, it returns a tuple with the estimated beta and r, estimated, with the year of each respective window. If *out_type=1* it returns the self.data of the model, a summary with all model information.

Return type tuple

predict(*initial*, *t*)

The function that uses the estimated parameters of the SIR model to predict the epidemy outputs (Suceptible, Infected and Recoverd) for the time samples provided, provided the initial conditions.

Parameters

- **initial** (*array*) – The initial values of the infected, suceptible and recovered data.
- **time** (*array*) – The time points to simulate the model.

Returns The values of the suceptible, infected and recovered, at time, respectivelly.

Return type tuple

result_summary(*out_plot=False, plot_size=[600, 400], save_results=False, folder_path='./', file_name='SIR_result_summary.png'*)

Method responsible for building a proper summary plot of the estimate process of the SIR model.

Parameters

- **out_plot** (*bool*) – Flag to output the bokeh.figure object.
- **plot_size** (*list*) – List with the plot size as *[width, height]*.
- **save_results** (*bool*) – Flag to save the results as a .png image.
- **folder_path** (*string*) – The path to the folder the user wants to save resulted image.
- **file_name** (*string*) – The name of the resulted image that will be saved.

Returns If *out_plot=True*, it returns a bokeh.figure object with the builded plots.

Return type bokeh.figure

simulate(*initial*, *time*, *theta*)

The function that simulate the differential SIR model, by computing the integration of the differential equations.

Parameters

- **initial** (*array*) – The initial values of the infected and suceptible data.
- **time** (*array*) – The time points to simulate the model.
- **theta** (*array*) – The Beta parameter, and r parameter, respectivelly.

Returns The values of the suceptible and infected, at time, respectivelly.

Return type tuple

models.stochastic_search.**findEpidemyBreaks**(*cases*, *threshold_prop=1.0*, *cases_before=10*)

The function responsible for determining the initial and final points of the epidemies windows.

Parameters

- **cases** (*array*) – The array with the cases values along time.
- **threshold_prop** (*float*) – The standard deviation proportion used as threshold for windowing. Default is *1.0*.
- **cases_before** (*int*) – The number of back samples to check for the initial window point. Default is *10*.

Returns With the list of window's starting points and window's final points, respectively.

Return type tuple

Cost Functions

models.cost_functions.**PrintException**(*e*)

models.cost_functions.**cost_NSIR**(*self*, *pars*, *dataset*, *initial*, *t*, *w*)

models.cost_functions.**cost_SEIR**(*self*, *pars*, *dataset*, *initial*, *t*, *w*)

The function to compute the error to guide the learning algorithm. It computes the quadratic error.

Parameters

- **pars** (*tuple*) – Tuple with Beta and r parameters, respectivelly.
- **dataset** (*list*) – The dataset with the respective S, I and R arrays.
- **initial** (*array*) – The initial values of suceptible and infected, respectivelly.
- **t** (*array*) – The time respective to each sample.

- **w** (*array*) – The weight respective to the suceptible and infected errors.

Returns The sum of the quadratic error, between simulated and real data.

Return type float

models.cost_functions.**cost_SIR**(*self*, *pars*, *dataset*, *initial*, *t*, *w*)
The function to compute the error to guide the learning algorithm. It computes the quadratic error.

Parameters

- **p** (*tuple*) – Tuple with Beta and r parameters, respectivelly.
- **S** (*array*) – The suceptible data values.
- **I** (*array*) – The infected data values.
- **initial** (*array*) – The initial values of suceptible and infected, respectivelly.
- **t** (*array*) – The time respective to each sample.
- **w** (*array*) – The weight respective to the suceptible and infected errors.

Returns The sum of the quadratic error, between simulated and real data.

Return type float

models.cost_functions.**cost_SIRD**(*self*, *pars*, *dataset*, *initial*, *t*, *w*)
The function to compute the error to guide the learning algorithm. It computes the quadratic error.

Parameters

- **p** (*tuple*) – Tuple with Beta and r parameters, respectivelly.
- **S** (*array*) – The suceptible data values.
- **I** (*array*) – The infected data values.
- **initial** (*array*) – The initial values of suceptible and infected, respectivelly.
- **t** (*array*) – The time respective to each sample.
- **w** (*array*) – The weight respective to the suceptible and infected errors.

Returns The sum of the quadratic error, between simulated and real data.

Return type float

models.cost_functions.**cost_dSIR**(*self*, *pars*, *dataset*, *initial*, *t*, *w*)
The function to compute the error to guide the learning algorithm. It computes the quadratic error.

Parameters

- **p** (*tuple*) – Tuple with Beta and r parameters, respectivelly.
- **S** (*array*) – The suceptible data values.
- **I** (*array*) – The infected data values.
- **initial** (*array*) – The initial values of suceptible and infected, respectivelly.
- **t** (*array*) – The time respective to each sample.
- **w** (*array*) – The weight respective to the suceptible and infected errors.

Returns The sum of the quadratic error, between simulated and real data.

Return type float

Differential Models

models.differential_models.**NSIR**(*self*, *y*, *t*, *beta*, *r*, *betan*, *alpha*, *rn*, **args*)

models.differential_models.**SEIR**(*self*, *y*, *t*, *Beta*, *r*, *sigma*)

The function that computes the diferential set of equations of the SEIR Epidemic Model.

Parameters

- **y** (*tuple*) – Tuple with the suceptible and infected data.
- **t** (*array*) – The time respective to each y set of samples.
- **Beta** (*float*) – The Beta parameter.
- **r** (*float*) – The r parameter.
- **sigma** (*float*) – The sigma parameter.

Returns The derivative of the suceptible and infected data.

Return type tuple

models.differential_models.**SIR**(*self*, *y*, *t*, *parameters*, **args*)

The function that computes the diferential set of equations of the SIR Epidemic Model.

Parameters

- **y** (*tuple*) – Tuple with the suceptible and infected data.
- **t** (*array*) – The time respective to each y set of samples.
- **Beta** (*float*) – The Beta parameter.
- **r** (*float*) – The r parameter.

Returns The derivative of the suceptible and infected data.

Return type tuple

models.differential_models.**SIRD**(*self*, *y*, *t*, *parameters*)

The function that computes the diferential set of equations of the SIRD Epidemic Model.

Parameters

- **y** (*tuple*) – Tuple with the suceptible and infected data.
- **t** (*array*) – The time respective to each y set of samples.
- **Beta** (*float*) – The Beta parameter.
- **r** (*float*) – The r parameter.
- **mi** (*float*) – The mi parameter.

Returns The derivative of the suceptible and infected data.

Return type tuple

Module contents

copyright 2010 Marcelo Lima

license BSD-3-Clause

3.12 Vanderlei Cunha Parro

Find me on LinkedIn Find me on GitHub

Send me a message Watch my channel!

3.13 Marcelo Mendes Lafetá Lima

Find me on LinkedIn Find me on GitHub Send me a message

3.14 Felipe Brandão Ippolito

Find me on GitHub Send me a message

3.15 Felipe Antonio Silva de Andrade

Find me on LinkedIn Find me on GitHub Send me a message

CHAPTER 4

Indices e tabelas

- genindex
- modindex
- search

Python Module Index

m

models, 112
models.cost_functions, 110
models.differential_models, 112
models.stochastic_search, 107

Index

C

cost_dSIR() (*in module models.cost_functions*), 111
cost_NSIR() (*in module models.cost_functions*), 110
cost_SEIR() (*in module models.cost_functions*), 110
cost_SIR() (*in module models.cost_functions*), 111
cost_SIRD() (*in module models.cost_functions*), 111
cost_wrapper() (*models.stochastic_search.SIR method*), 107

F

findEpidemyBreaks() (*in module models.stochastic_search*), 110
fit() (*models.stochastic_search.SIR method*), 107
fit_multiple() (*models.stochastic_search.SIR method*), 108

M

models (*module*), 112
models.cost_functions (*module*), 110
models.differential_models (*module*), 112
models.stochastic_search (*module*), 107
monteCarlo_multiple() (*models.stochastic_search.SIR method*), 109

N

NSIR() (*in module models.differential_models*), 112

P

predict() (*models.stochastic_search.SIR method*), 109
PrintException() (*in module models.cost_functions*), 110

R

result_summary() (*models.stochastic_search.SIR method*), 109

S

SEIR() (*in module models.differential_models*), 112
simulate() (*models.stochastic_search.SIR method*), 110
SIR (*class in models.stochastic_search*), 107
SIR() (*in module models.differential_models*), 112
SIRD() (*in module models.differential_models*), 112