SEIRD Data Fitting Model from Henri Froesi for Brazilian States Covid19 - Evolution and Prediction

References:

[1] Modelling Beyond the Basic SIR Model - Henri Froesi - Towards Data Science

https://towardsdatascience.com/infectious-disease-modelling-beyond-the-basic-sir-model-216369c584c4 (https://towardsdatascience.com/infectious-disease-modelling-beyond-the-basic-sir-model-216369c584c4)

[2] Infectious Disease Modelling. Fitting Your Model to Corona Virus Data - Henri Froesi-Towards Data Science

https://towardsdatascience.com/infectious-disease-modelling-fit-your-model-to-coronavirus-data-2568e672dbc7 (https://towardsdatascience.com/infectious-disease-modelling-fit-your-model-to-coronavirus-data-2568e672dbc7)

Adapted by Ivan Carlos P. da Cruz (GitHub @icpcruz)

Revision and contributions by Eduardo Correa Araújo

Model summary ref. [1]

Imports

In [1]:

```
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline

from scipy.integrate import odeint

# !pip install lmfit - DONE !
import lmfit
from lmfit import Model # classe lmfit para a criação do modelo de fitting

import numdifftools # permite que lmfit calcule a matriz de correlações

import datetime

import warnings
warnings.filterwarnings('ignore')
```

Importação de dados de óbitos diários por estado brasileiro (NE)

Diretório sincronizado com o GitHub dos dados (arquivos csv):

In [2]:

```
path = "C:/Users/Ivan/Documents/GitHub/googleData/data/"
```

Dicionário de nomes dos arquivos de óbitos diários por estado do NE

In [3]:

```
pref = "ON"
suf  = "_An.csv"
Data_files_names = {
    "Alagoas"    : pref+"AiL"+suf ,
    "Bahia"      : pref+"BiA"+suf ,
    "Ceará"      : pref+"CiE"+suf ,
    "Maranhão"   : pref+"MiA"+suf ,
    "Paraíba"    : pref+"PiB"+suf ,
    "Pernambuco" : pref+"PiE"+suf ,
    "Piauí"      : pref+"PiI"+suf ,
    "RioGrdoN"   : pref+"RiN"+suf ,
    "Sergipe"    : pref+"SiE"+suf
}
```

Leitura dos arquivos (atualizados diariamente)

In [4]:

```
data_ON_AL = pd.read_csv(path+Data_files_names["Alagoas"], sep =",")
data_ON_BA = pd.read_csv(path+Data_files_names["Bahia"], sep =",")
data_ON_CE = pd.read_csv(path+Data_files_names["Ceará"], sep =",")
data_ON_MA = pd.read_csv(path+Data_files_names["Maranhão"], sep =",")
data_ON_PB = pd.read_csv(path+Data_files_names["Paraíba"], sep =",")
data_ON_PE = pd.read_csv(path+Data_files_names["Pernambuco"], sep =",")
data_ON_PI = pd.read_csv(path+Data_files_names["Piauí"], sep =",")
data_ON_RN = pd.read_csv(path+Data_files_names["RioGrdoN"], sep =",")
data_ON_SE = pd.read_csv(path+Data_files_names["Sergipe"], sep =",")
```

"Parsing" das datas

In [5]:

```
format_str = "%Y-%m-%d"
data_ON_AL['Data'] = data_ON_AL['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_BA['Data'] = data_ON_BA['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_CE['Data'] = data_ON_CE['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_MA['Data'] = data_ON_MA['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_PB['Data'] = data_ON_PB['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_PE['Data'] = data_ON_PE['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_PI['Data'] = data_ON_PI['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_RN['Data'] = data_ON_RN['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
data_ON_SE['Data'] = data_ON_SE['Data'].apply(lambda x:datetime.datetime.strptime(x,for
mat_str))
```

In [6]:

```
data_ON_CE.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85 entries, 0 to 84
Data columns (total 2 columns):
Data        85 non-null datetime64[ns]
ONCiE_An    85 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 1.4 KB
```

In [7]:

```
data_ON_CE.tail()
```

Out[7]:

	Data	ONCiE_An
80	2020-06-05	77.0
81	2020-06-06	75.0
82	2020-06-07	17.0
83	2020-06-08	138.0
84	2020-06-09	189.0

In [8]:

```
data_ON_CE.describe()
```

Out[8]:

	ONCiE_An
count	85.000000
mean	50.694118
std	60.005560
min	0.000000
25%	4.000000
50%	25.000000
75%	75.000000
max	261.000000

Definindo as colunas 'Data' como índices

In [9]:

```
data_ON_AL.set_index('Data')
data_ON_BA.set_index('Data')
data_ON_CE.set_index('Data')
data_ON_MA.set_index('Data')
data_ON_PB.set_index('Data')
data_ON_PE.set_index('Data')
data_ON_PI.set_index('Data')
data_ON_RN.set_index('Data')
data_ON_SE.set_index('Data')
data_ON_CE.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85 entries, 0 to 84
Data columns (total 2 columns):
Data        85 non-null datetime64[ns]
ONCiE_An    85 non-null float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 1.4 KB
```

Parâmetros Epidemiológicos Assumidos para a Covid-19 (como universais) p/o SEIRD

In [10]:

```
#
# VALORES MÉDIOS ASSUMIDOS (ref. https://theconversation.com/how-long-are-you-infectious-when-you-have-coronavirus-135295):
#
num_of_incubation     = 5      # número de dias de incubação da doença (1/delta).
num_of_infecting_days = 9      # número de dias que um infectado (ativo) infecta outros (= D = 1/gama)
num_to_dead           = 17     # número de dias para evoluir de infectado para óbito (1/rho)
subnotif_index        = 12     # número de vezes que os casos reais de infectados é maior que os notificados
apparent_letality     = 0.065  # taxa aparente de letalidade (% dos notificados que vão a óbito)
#
Epi_days              = 200    # duração arbitrada da pandemia
#
# FREQUÊNCIAS (POR DIA) CORRESPONDENTES
#
alpha = apparent_letality/subnotif_index
delta = 1/num_of_incubation
gamma = 1/num_of_infecting_days
rho   = 1/num_to_dead
#
print('\n','*** PARÂMETROS EPIDEMIOLÓGICOS ADOTADOS NO MODELO SEIRD:','\n')
print('    Freq de letalidade   (alpha) = {:.3f}'.format(alpha))
print('    Freq de incubação    (delta) = {:.3f}'.format(delta))
print('    Freq de infetividade (gamma) = {:.3f}'.format(gamma))
print('    Freq para óbito      (rho)   = {:.3f}'.format(rho))
```

```
 *** PARÂMETROS EPIDEMIOLÓGICOS ADOTADOS NO MODELO SEIRD:

    Freq de letalidade   (alpha) = 0.005
    Freq de incubação    (delta) = 0.200
    Freq de infetividade (gamma) = 0.111
    Freq para óbito      (rho)   = 0.059
```

Dados da população por estado

In [11]:

```
N_BR ={"Brasil": 210147125,
             "Nordeste":
                 {'MA':7075181,
                  'PI':3273227,
                  'CE':9132078,
                  'RN':3506853,
                  'PB':4018127,
                  'PE':9557071,
                  'AL':3337357,
                  'SE':2298696,
                  'BA':14873064},
             "Norte":
                 {'RO':1777225,
                  'AC':881935,
                  'AM':4144597,
                  'RR': 605761,
                  'PA':8602865,
                  'AP':845731,
                  'TO':1572866},
             "Sudeste":
                 {'MG':21168791,
                  'ES':4018650,
                  'RJ':17264943,
                  'SP':45919049},
             "Sul":
                 {'PR':11433957,
                  'SC':7164788,
                  'RS':11377239},
             "Centro-Oeste":
                 {'MS':2778986,
                  'MT':3484466,
                  'GO':7018354,
                  'DF':3015268} }
```

Modelo com R_0 (e "beta") dependentes do tempo

Função sistema de EDOs do modelo SEIRD que é recursivamente chamada pelo solver das ODEs

In [12]:

```
def deriv(y, t, N, beta, gamma, delta, alpha, rho):
    #
    S, E, I, R, D = y
    #
    dSdt = -beta(t) * S * I / N
    dEdt = beta(t) * S * I / N - delta * E
    dIdt = delta * E - (1 - alpha) * gamma * I - alpha * rho * I
    dRdt = (1 - alpha) * gamma * I
    dDdt = alpha * rho * I
    #
    return dSdt, dEdt, dIdt, dRdt, dDdt
```

R_0(t) como uma função logística (com 4 parâmetros a serem ajustados pelo algorítimo de fitting/otimizacão)

In [13]:

```
def logistic_R_0(t, R_0_start, k, x0, R_0_end):
    return (R_0_start-R_0_end) / (1 + np.exp(-k*(-t+x0))) + R_0_end
```

Os parâmetros que serão ajustados no processo de otimização (fitting), são:

- **R_0_start** : qual era o valor da taxa de infetividade (R_0) no início da pandemia
- **R_0_end** : qual será o valor da taxa de infetividade (R_0) ao fim da pandemia
- **k** : parâmetro que fornece a taxa de decaímento de R_0
- **x0** : posição "mediana" da curva logística => para lim(k -> infinito) => função degrau, x_0 é a posição do degrau

Valores inicias arbitrados dos parâmetros para o processo de otimização dados como tuplas (esperado, mínimo, máximo)

In [14]:

```
params_init_min_max = {"R_0_start": (3.0, 2.0, 6.0),
                       "k": (0.5, 0.01, 10.0),
                       "x0": (30, 0, 250),
                       "R_0_end": (0.9, 0.1, 3.5)}
```

Ajuste das curvas R_0(t) e SEIRD a partir dos dados de mortalidade diária

Definição da função solver do sistema de EDOs do modelo SEIRD que é recursivamente chamada pelo algoritmo de otimização (fitting aos dados)

In [15]:

```
def SolverEDOs(days, N, R_0_start, k, x0, R_0_end):
    #
    # definição da função beta(t) = R_0(t)*gama
    #
    def beta(t):
        return logistic_R_0(t, R_0_start, k, x0, R_0_end) * gamma
    #
    # Valores iniciais das variáveid do SEIRD: N-1 (população), 1 exposto, 0 infectado,
recuperados ou mortos
    #
    y0 = N-1.0, 1.0, 0.0, 0.0, 0.0
    #
    # geração da grid temporal a partir da quantidade de dias informados
    #
    t = np.arange(days)
    #print(t)
    #
    # Integração do sistema de EDOs - o sistema é passado pela função 'deriv' na chamad
a do solver
    #
    ret = odeint(deriv, y0, t, args=(N, beta, gamma, delta, alpha, rho))
    #
    # Salvando os resultados nas variáveis do sistema
    #
    S, E, I, R, D = ret.T
    #
    # Cálculo de R_0(t)
    #
    R_0_over_time = [beta(i)/gamma for i in range(len(t))]
    #
    return t, S, E, I, R, D, R_0_over_time
```

Estado teste: Maranhão

In [16]:

```
N = N_BR['Nordeste']['MA']                  # <-- população do estado
y_data = data_ON_MA['ONMiA_An'].to_numpy() # <-- de pandas series para numpy array - nd
array
x_data = np.arange(len(y_data))             # <-- do dia 1 ao final da série (Python ini
cia em 'zero', daí o +1)
days   = len(x_data)                        # <-- duração, até "hoje", da pandemia no es
tado, para o fitting dos parâmetros
```

Definição da função de fitting que chama o solver das EDOs

In [17]:

```
def fitter(x, R_0_start, k, x0, R_0_end):
    ret = SolverEDOs(days, N, R_0_start, k, x0, R_0_end) # <-- chama a função solver da
s EDOs (que chama odeint para integrar)
    deaths_fitted = ret[5]                               # <-- óbitos: 6o elem. da tupl
a (t, S, E, I, R, D, R_0)
    return deaths_fitted[x]                              # <-- retorna vetor com as fat
alidades no tempo
```

Criando o modelo de fitting com o pacote 'lmfit'

In [18]:

```
# Cria o modelo de fitting passando a função fitter como parâmetro para lmfit
#
mod = lmfit.Model(fitter)
#
# Criação do vetor de parâmetros para o fitting, incluindo 3 estimativas iniciais (inic
ial, min, max) p/cada parâmetro
#
for kwarg, (init, mini, maxi) in params_init_min_max.items(): # <-- usa as estimativas
 fornecidas acima
    mod.set_param_hint(str(kwarg), value=init, min=mini, max=maxi, vary=True)
#
# Criação do vetor de parâmetros para o otimizador
#
params = mod.make_params()
```

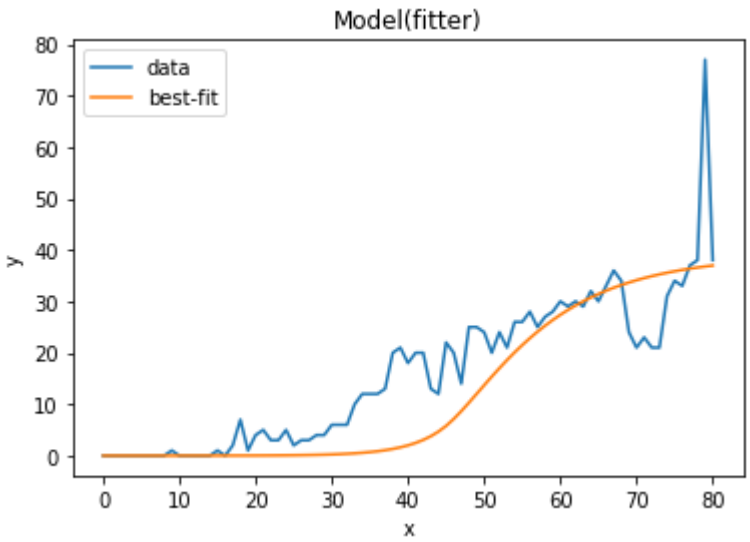
Realizando o fitting com os dados de óbitos diários (não suavizados)

In [19]:

```
result = mod.fit(y_data, params, method="least_squares", x=x_data)
```

In [20]:

```
result.plot_fit(datafmt="-");
```



In [151]:

```
result
```

Out[151]:

Model

Model(fitter)

Fit Statistics

fitting method	least_squares
# function evals	18
# data points	81
# variables	4
chi-square	6204.49746
reduced chi-square	80.5778891
Akaike info crit.	359.425025
Bayesian info crit.	369.002821

Variables

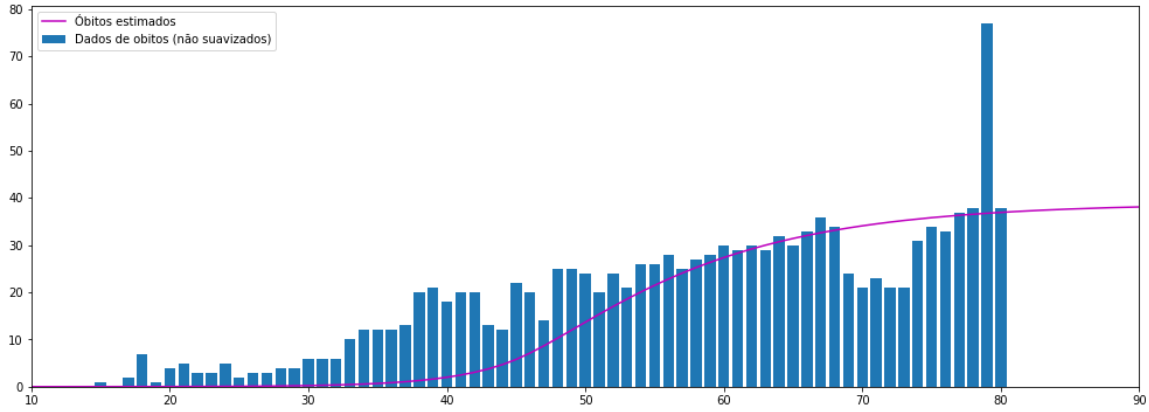
name	value	standard error	relative error	initial value	min	max	vary
R_0_start	6.00000000	0.75073533	(12.51%)	3.0	2.00000000	6.00000000	True
k	10.0000000	7764.41337	(77644.13%)	0.5	0.01000000	10.0000000	True
x0	45.1149149	8.01967269	(17.78%)	30	0.00000000	250.000000	True
R_0_end	0.10000000	0.71470405	(714.70%)	0.9	0.10000000	3.50000000	True

Correlations (unreported correlations are < 0.100)

R_0_start	x0	-0.9875
k	x0	0.8916
R_0_start	k	-0.8544
k	R_0_end	0.6718
R_0_start	R_0_end	-0.3216
x0	R_0_end	0.3192

In [98]:

```
plt.figure(figsize = [17, 6])
plt.xlim(10,90)
#plt. ylim(0,40)
plt.plot(res_SEIRD[0],res_SEIRD[5],          label = 'Óbitos estimados', color='m')
plt.bar(x_data, y_data,                      label = 'Dados de obitos (não suavizado
s)')
plt.legend()
plt.show()
```



Parâmetros otimizados do modelo com dados diários (não suavizados)

In [21]:

```
parametros = []
paramsfit = result.best_values
parametros.append(paramsfit['R_0_start'])
parametros.append(paramsfit['k'])
parametros.append(paramsfit['x0'])
parametros.append(paramsfit['R_0_end'])

#
print('\n','*** PARÂMETROS OTIMIZADOS DA CURVA LOGÍSTICA DE R_0 PELO FITTING COM OS DAD
OS DE ÓBITOS:','\n')
print('    R_0_start = {:.3f}'.format(paramsfit['R_0_start']))
print('    k         = {:.3f}'.format(paramsfit['k']))
print('    x_0       = {:.0f}'.format(paramsfit['x0']))
print('    R_0_end   = {:.3f}'.format(paramsfit['R_0_end']))
```

```
 *** PARÂMETROS OTIMIZADOS DA CURVA LOGÍSTICA DE R_0 PELO FITTING COM OS D
ADOS DE ÓBITOS:

    R_0_start = 6.000
    k         = 10.000
    x_0       = 45
    R_0_end   = 0.100
```

Realizando o fitting com os dados de óbitos diários suavizados (por média móvel)

In [94]:

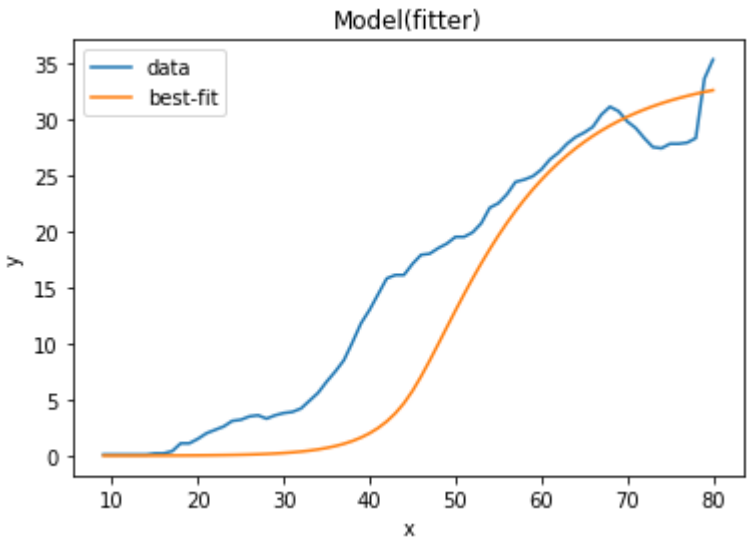
```
window_size = 10 #  <-- tamanho da janela móvel para a suavização

# First, we need to convert y_data Numpy ndarray to a Pandas Series

y_series = pd.Series(y_data)

# Tail-rolling average transform

y_rolling_S = y_series.rolling(window=window_size).mean().dropna()

# Now getting back to a ndarray

y_rolling = y_rolling_S.to_numpy()
x_rolling = x_data[window_size-1:] # <-- remove os 'n-1' pontos do início da série por
 conta da média móvel sobre 'n' pontos
```

In [52]:

```
result_roll = mod.fit(y_rolling, params, method="least_squares", x=x_rolling)
```

In [54]:

```
result_roll.plot_fit(datafmt="-");
```



Parâmetros otimizados do modelo com dados diários (originais, não suavizados)

In [55]:

```
params_roll = []
paramsfit_roll = result_roll.best_values
params_roll.append(paramsfit_roll['R_0_start'])
params_roll.append(paramsfit_roll['k'])
params_roll.append(paramsfit_roll['x0'])
params_roll.append(paramsfit_roll['R_0_end'])

#
print('\n','*** PARÂMETROS OTIMIZADOS DA CURVA LOGÍSTICA DE R_0 PELO FITTING COM OS DAD
OS SUAVIZADOS DOS ÓBITOS:','\n')
print('    R_0_start = {:.3f}'.format(paramsfit_roll['R_0_start']))
print('    k         = {:.3f}'.format(paramsfit_roll['k']))
print('    x_0       = {:.0f}'.format(paramsfit_roll['x0']))
print('    R_0_end   = {:.3f}'.format(paramsfit_roll['R_0_end']))
```

```
 *** PARÂMETROS OTIMIZADOS DA CURVA LOGÍSTICA DE R_0 PELO FITTING COM OS D
ADOS SUAVIZADOS DOS ÓBITOS:

    R_0_start = 6.000
    k         = 10.000
    x_0       = 45
    R_0_end   = 0.100
```

In [147]:

```
result_roll
```

Out[147]:

Model

Model(fitter)

Fit Statistics

fitting method	least_squares
# function evals	17
# data points	72
# variables	4
chi-square	1919.59648
reduced chi-square	28.2293600
Akaike info crit.	244.390699
Bayesian info crit.	253.497364

Variables

name	value	standard error	relative error	initial value	min	max	vary
R_0_start	6.00000000	0.51869023	(8.64%)	3.0	2.00000000	6.00000000	True
k	10.0000000	5255.58346	(52555.83%)	0.5	0.01000000	10.0000000	True
x0	44.5049704	5.46622767	(12.28%)	30	0.00000000	250.000000	True
R_0_end	0.10000000	0.46775650	(467.76%)	0.9	0.10000000	3.50000000	True

Correlations (unreported correlations are < 0.100)

R_0_start	x0	-0.9878
k	x0	0.8961
R_0_start	k	-0.8586
k	R_0_end	0.6718
x0	R_0_end	0.3283
R_0_start	R_0_end	-0.3279

Relatórios de acurácia e intervalo de confiança do fitting dos dados suavizados

Relatório de métricas de erros e matriz de coeficientes de correlação

In [68]:

```
from io import StringIO
```

In [73]:

```
fit_report_str = StringIO(result_roll.fit_report())

df_fit_report = pd.read_csv(fit_report_str, sep =";")
df_fit_report
```

Out[73]:

	[[Model]]
0	Model(fitter)
1	[[Fit Statistics]]
2	# fitting method = least_squares
3	# function evals = 17
4	# data points = 72
5	# variables = 4
6	chi-square = 1919.59648
7	reduced chi-square = 28.2293600
8	Akaike info crit = 244.390699
9	Bayesian info crit = 253.497364
10	[[Variables]]
11	R_0_start: 6.00000000 +/- 0.51869023 (8.6...
12	k: 10.0000000 +/- 5255.58346 (525...
13	x0: 44.5049704 +/- 5.46622767 (12....
14	R_0_end: 0.10000000 +/- 0.46775650 (467...
15	[[Correlations]] (unreported correlations are ...
16	C(R_0_start, x0) = -0.988
17	C(k, x0) = 0.896
18	C(R_0_start, k) = -0.859
19	C(k, R_0_end) = 0.672
20	C(x0, R_0_end) = 0.328
21	C(R_0_start, R_0_end) = -0.328

In [74]:

```
## result_roll.conf_interval() <-- *** DEMORA PACAS !!!
```

Out[74]:

```
OrderedDict([('R_0_start',
              [(0.9973002039367398, 5.897959978791761),
               (0.9544997361036416, 5.955459255620598),
               (0.6826894921370859, 5.986748532097236),
               (0.0, 5.999999999999999),
               (0.6826894921370859, inf),
               (0.9544997361036416, inf),
               (0.9973002039367398, inf)]),
             ('k',
              [(0.9973002039367398, 0.5150533268888328),
               (0.9544997361036416, 0.8100106315330966),
               (0.6826894921370859, 1.6035312303418328),
               (0.0, 9.999999999999934),
               (0.6826894921370859, inf),
               (0.9544997361036416, inf),
               (0.9973002039367398, inf)]),
             ('x0',
              [(0.9973002039367398, 43.68066689134777),
               (0.9544997361036416, 44.13774961199189),
               (0.6826894921370859, 44.31085940948412),
               (0.0, 44.504970418607314),
               (0.6826894921370859, 44.68569532774454),
               (0.9544997361036416, 44.865219293173986),
               (0.9973002039367398, 45.084043525411545)]),
             ('R_0_end',
              [(0.9973002039367398, -inf),
               (0.9544997361036416, -inf),
               (0.6826894921370859, -inf),
               (0.0, 0.10000000000000127),
               (0.6826894921370859, 0.13776573755230834),
               (0.9544997361036416, inf),
               (0.9973002039367398, inf)])])
```

Intervalos de confiança

In [75]:

```
result_roll.ci_report()
```

Out[75]:

```
'              99.73%    95.45%    68.27%    _BEST_    68.27%    95.45%
99.73%\n R_0_start:  -0.10204  -0.04454  -0.01325   6.00000      +inf
+inf      +inf\n k        :  -9.48495  -9.18999  -8.39647  10.00000      +
inf      +inf      +inf\n x0       :  -0.82430  -0.36722  -0.19411  44.504
97  +0.18072  +0.36025  +0.57907\n R_0_end  :      -inf      -inf      -in
f   0.10000  +0.03777      +inf      +inf'
```

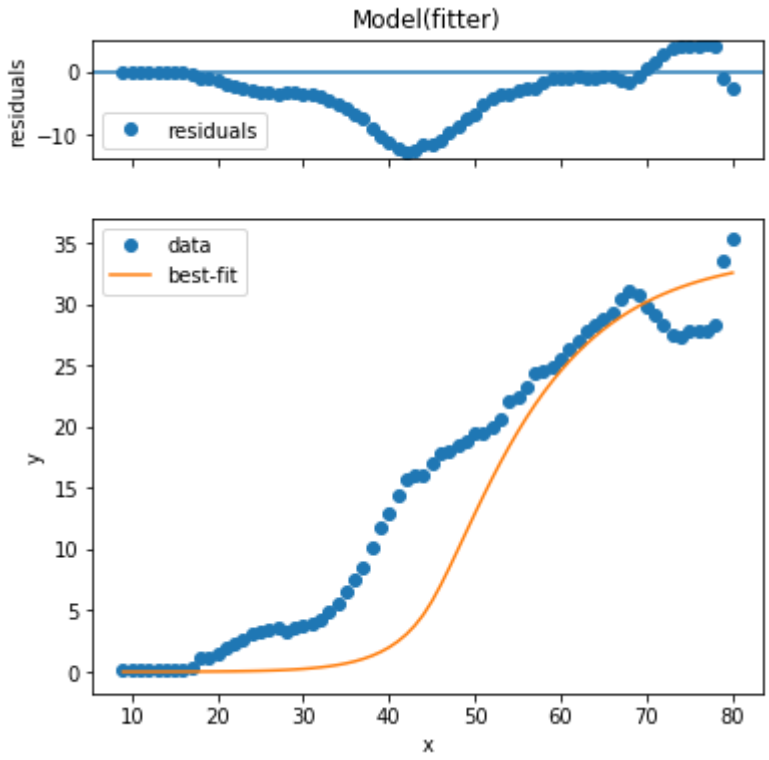
Plotagem dos resíduos

In [104]:

```
result_roll.plot() # _residuals
```

Out[104]:

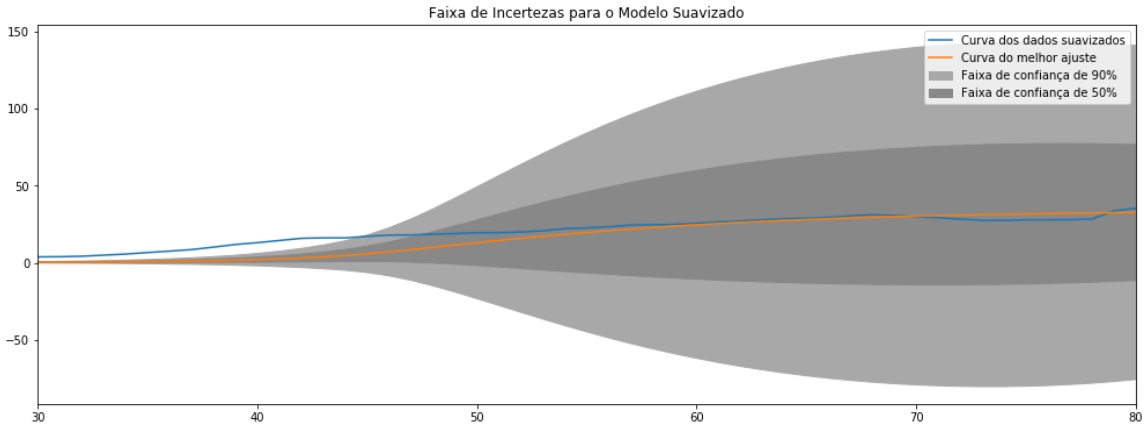
```
(<Figure size 432x432 with 2 Axes>, GridSpec(2, 1, height_ratios=[1, 4]))
```



Avaliação das incertezas - 50% e 90% de intervalos deconfiança

In [137]:

```
result_roll_50 = result_roll.eval_uncertainty(x=x_rolling, sigma = 0.50)

result_roll_90 = result_roll.eval_uncertainty(x=x_rolling, sigma = 0.90)

plt.figure(figsize = [17, 6])
plt.xlim(30,80)
plt.title('Faixa de Incertezas para o Modelo Suavizado')
plt.plot(x_rolling, y_rolling, label='Curva dos dados suavizados')
plt.plot(x_rolling, result_roll.best_fit, label= 'Curva do melhor ajuste')
plt.fill_between(x_rolling, result_roll.best_fit-result_roll_90,
                 result_roll.best_fit+result_roll_90, color='#A8A8A8',label = 'Faixa de
confiança de 90%')
plt.fill_between(x_rolling, result_roll.best_fit-result_roll_50,
                 result_roll.best_fit+result_roll_50, color='#888888', label = 'Faixa d
e confiança de 50%')
plt.legend()
plt.show()
```



Chamada do SOLVER com os parâmetros otimizados da curva logística para R_0(t) para obter as var SEIRD

In [22]:

```
#
# Chamada do SOLVER com os parâmetros otimizados da curva logística para R_0(t)
#
Epi_days = 250 # <-- tempo estimado de duração da pandemia
#
res_SEIRD = SolverEDOs(Epi_days, N,*parametros)
```

In [56]:

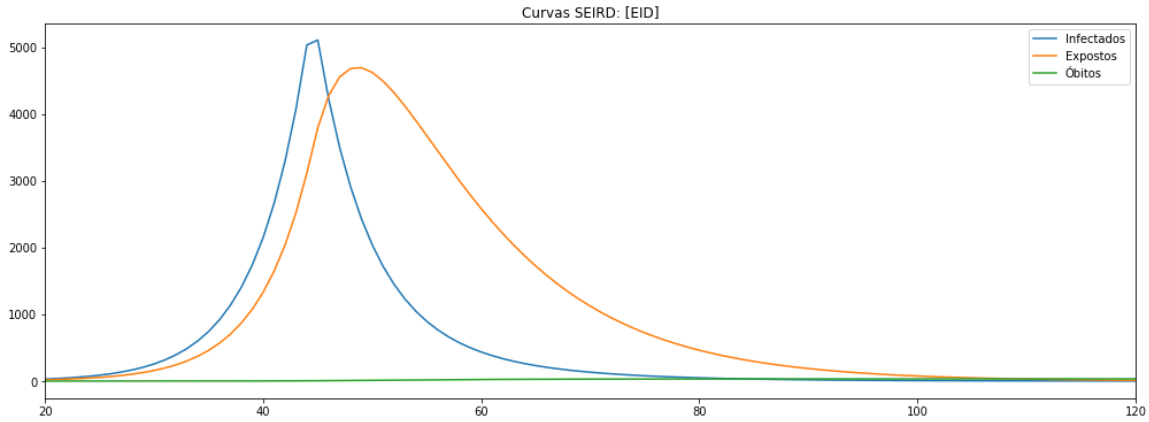
```
res_SEIRD_roll = SolverEDOs(Epi_days, N,*params_roll)
```

Plotagem das curvas SEIRD para o modelo com dados suavizados

Curvas IED (S e R fora de escala nessa fase da pandemia)

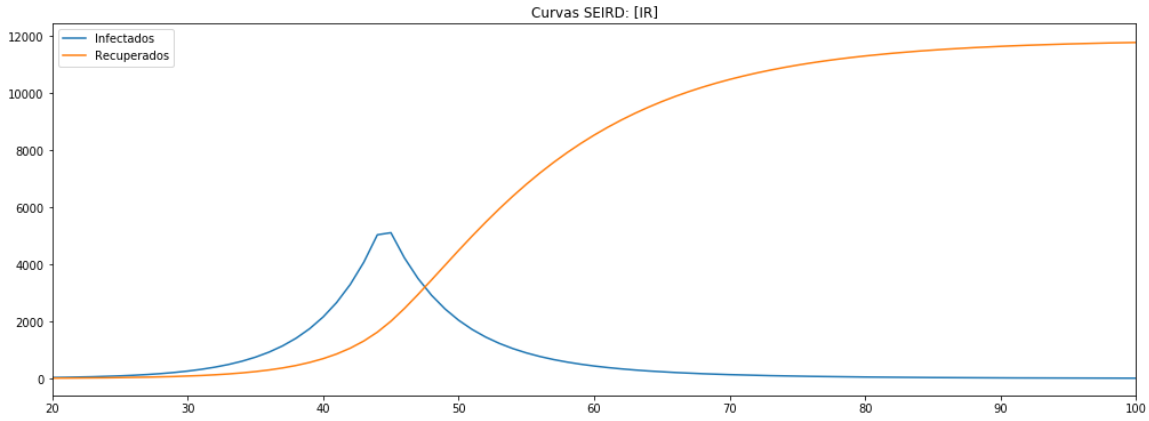
In [101]:

```
plt.figure(figsize = [17, 6])
plt.title('Curvas SEIRD: [EID]')
plt.xlim(20,120)
#plt.plot(res_SEIRD[0],res_SEIRD[1], label = 'Suscetíveis')
plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[2], label = 'Infectados')
plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[3], label = 'Expostos')
#plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[4], label = 'Recuperados')
plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[5], label = 'Óbitos')
plt.legend()
plt.show()
```



In [100]:

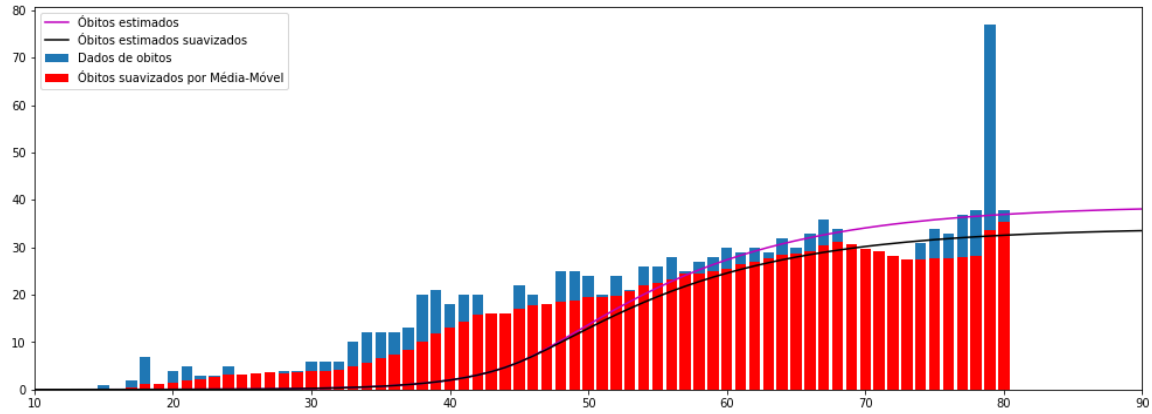
```
plt.figure(figsize = [17, 6])
plt.title('Curvas SEIRD: [IR]')
plt.xlim(20,100)
#plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[1], label = 'Suscetíveis')
plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[2], label = 'Infectados')
#plt.plot(dados[0],dados[3], label = 'Expostos')
plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[4], label = 'Recuperados')
plt.legend()
plt.show()
```



Comparação do ajuste das curvas estimadas de óbitos aos dados originais e suavizados (média móvel de 10 dias)

In [99]:

```
plt.figure(figsize = [17, 6])
plt.xlim(10,90)
#plt. ylim(0,40)
plt.plot(res_SEIRD[0],res_SEIRD[5],           label = 'Óbitos estimados', color='m')
plt.bar(x_data, y_data,                       label = 'Dados de obitos')
plt.plot(res_SEIRD_roll[0],res_SEIRD_roll[5], label = 'Óbitos estimados suavizados', co
lor='black')
plt.bar(x_rolling, y_rolling,                 label = 'Óbitos suavizados por Média-Móve
l', color='red')
plt.legend()
plt.show()
```



Forecast para 14 dias (com os dados suavizados)

In [149]:

```
period = 14 #<-- days forecast
#
last_day_in_data = x_rolling[len(x_rolling) - 1]
last_day_forecast  = last_day_in_data + period
```

In [178]:

```
Day = pd.Series(res_SEIRD_roll[0])
S   = pd.Series(res_SEIRD_roll[1])
E   = pd.Series(res_SEIRD_roll[2])
I   = pd.Series(res_SEIRD_roll[3])
R   = pd.Series(res_SEIRD_roll[4])
D   = pd.Series(res_SEIRD_roll[5])
```

In [187]:

```
df = pd.DataFrame()
df['Dia da Epidemia'] = Day
df['Suscetíveis']     = S
df['Expostos']        = E
df['Infectados']      = I
df['Recuperados']     = R
df['Óbitos']          = D
#
df.set_index(['Dia da Epidemia'])
#
pd.options.display.float_format = '{:,.0f}'.format
df[last_day_in_data : last_day_forecast]
```

Out[187]:

	Dia da Epidemia	Suscetíveis	Expostos	Infectados	Recuperados	Óbitos
80	80	7,063,341	49	462	11,297	33
81	81	7,063,336	45	422	11,345	33
82	82	7,063,331	41	386	11,390	33
83	83	7,063,327	37	353	11,431	33
84	84	7,063,324	34	323	11,468	33
85	85	7,063,320	31	295	11,502	33
86	86	7,063,317	28	269	11,533	33
87	87	7,063,314	25	246	11,562	33
88	88	7,063,312	23	225	11,588	33
89	89	7,063,309	21	206	11,612	33
90	90	7,063,307	19	188	11,633	34
91	91	7,063,305	18	172	11,653	34
92	92	7,063,303	16	157	11,671	34
93	93	7,063,302	15	143	11,688	34

In [183]:

```
df.describe()
```

Out[183]:

	Dia da Epidemia	Suscetíveis	Expostos	Infectados	Recuperados	Óbitos
count	250.000000	2.500000e+02	250.000000	250.000000	250.000000	250.000000
mean	124.500000	7.065276e+06	237.702052	429.388068	9211.003585	26.557762
std	72.312977	4.225226e+03	762.322199	1036.104065	4548.498877	13.114526
min	0.000000	7.063284e+06	0.000011	0.000000	0.000000	0.000000
25%	62.250000	7.063284e+06	0.002990	0.026943	9109.366176	26.264714
50%	124.500000	7.063285e+06	0.835154	2.866172	11852.725292	34.174545
75%	186.750000	7.063560e+06	36.042408	153.351185	11862.794787	34.203578
max	249.000000	7.075180e+06	5104.507170	4691.166852	11862.830655	34.203681

Curva variação de R_0(t)

In [57]:

```
plt.figure(figsize = [10, 4])
plt.xlim(30,60)
plt.title('R_0')
plt.plot(res_SEIRD[0],res_SEIRD[6], label = 'R_0')
```

Out[57]:

```
[<matplotlib.lines.Line2D at 0x21851f6ecc0>]
```