Maré IBGE - parte 3

Análise do nível do mar

In [1]:
```
# ao rodar todo o notebook, garante que tudo está sendo removido da memória!
# evita problemas de conflito de variáveis com mesmo nome, etc.
%reset
```

In [2]:
```
import pickle
import numpy as np
import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pandas as pd
from utide import solve, reconstruct
import scipy.signal as signal
```

Carregando dados pré-processados no notebook 'Mare_IBGE_2.ipynb'

In [3]:
```
with open('Ibge_base_dados.pik', 'rb') as handle:

    base_dados = pickle.load(handle)
```

In [4]:
```
# investiga o conteúdo do dicionário
base_dados.keys()
```

Out[4]:
```
dict_keys(['IMB', 'ARC', 'SAL', 'FOR', 'SAN'])
```

In [5]:
```
def monta_datetime(d):

    tempo = []
    for i in range(len(d)):
        ano = int(d[i,0])
        mes = int(d[i,1])
        dia = int(d[i,2])
        hora = int(d[i,3])
        minuto = int(d[i,4])
        segundo = int(d[i,5])

        c_tempo = datetime.datetime(ano, mes, dia, hora, minuto, segundo)

        tempo.append(c_tempo)

    return tempo
```

Cria vetor de tempo regularmente espação para uniformização dos dados

In [6]:
```
tempo_inicio = datetime.datetime(2018, 3, 1, 0, 0, 0)

tempo_fim = datetime.datetime(2018, 3, 15, 23, 55, 0)

intervalo = datetime.timedelta(minutes = 5)
```

```
tempo_ref = np.arange(tempo_inicio, tempo_fim + intervalo, intervalo)

tempo_ref_n = mdates.date2num(tempo_ref)
```

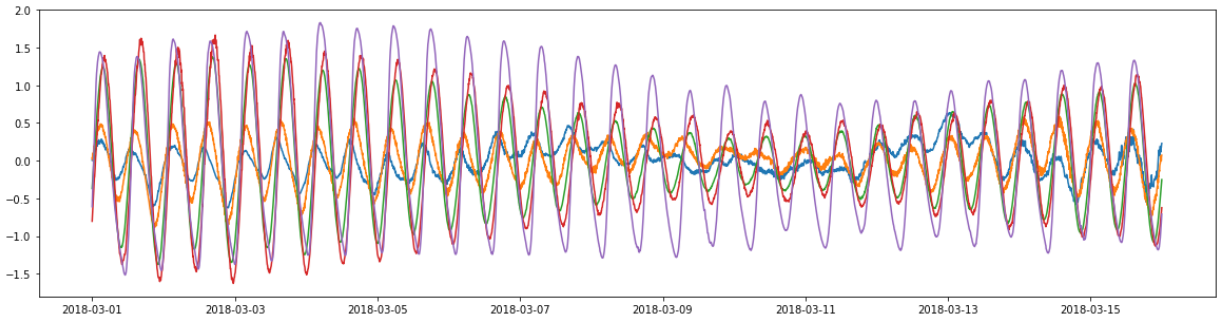
Re-amostra os dados de cada estação para o vetor criado acima!

In [7]:

```
estacoes = list(base_dados.keys())

g_nivel_i = []

for i in range(5):

    dados = base_dados[estacoes[i]]

    dados = np.array(dados)

    tempo = monta_datetime(dados)

    nivel = dados[:,6]

    tempo_n = mdates.date2num(tempo)

    nivel_i = np.interp(tempo_ref_n, tempo_n, nivel)

    nivel_i = nivel_i - np.mean(nivel_i)

    g_nivel_i = g_nivel_i + nivel_i.tolist()

nivel_i = np.reshape(g_nivel_i, (5,-1)).T
```

In [8]:

```
plt.figure(figsize=(20,5))
plt.plot(tempo_ref, nivel_i)
plt.show()
```



Define as latitudes das estações, a partir do site do IBGE

As latitutdes serão usadas na análise harmônica!

In [9]:

```
lat_imb = -(28 + 13/60 + 52.3/3600)
lat_arr = -(22 + 58/60 + 21/3600)
lat_sal = -(12 + 58/60 + 26.29/3600)
lat_for = -(3 + 42/60 + 52.55/3600)
lat_san = -(0 + 3/60 + 41/3600)

latitudes = [lat_imb, lat_arr, lat_sal, lat_for, lat_san]
```

Faz a análise harmônica

In [10]:

```
coefs = []
g_sinal_h = []
```

```
for i in range(5):

    nivel = nivel_i[:,i]

    coef = solve(tempo_ref_n + 3/24, nivel,
                 lat = latitudes[0],
                 method = 'ols',
                 conf_int = 'MC')

    coefs.append(coef)

    sinal_h = reconstruct(tempo_ref_n + 3/24, coef)

    g_sinal_h.append(sinal_h)
```

```
solve: matrix prep ... solution ... done.
prep/calcs ... done.
solve: matrix prep ... solution ... done.
prep/calcs ... done.
solve: matrix prep ... solution ... done.
prep/calcs ... done.
solve: matrix prep ... solution ... done.
prep/calcs ... done.
solve: matrix prep ... solution ... done.
prep/calcs ... done.
```

In [11]:

```
print(coef.keys())
```

```
dict_keys(['name', 'aux', 'nR', 'nNR', 'nI', 'weights', 'A', 'g', 'mean', 'slope',
'g_ci', 'A_ci', 'diagn', 'PE', 'SNR'])
```

Bonus: gera uma tabela com os principais constituintes

In [12]:

```
name_order = ['MSF', 'O1', 'K1', 'M2', 'S2', 'M4']
j = np.zeros((1,len(name_order)))

for i in range(5):
    name = coefs[i]['name']
    a = coefs[i]['A']
    g = coefs[i]['g']

    g_a = []
    g_g = []
    for no in name_order:
        for i2, na in enumerate(name):
            if na == no:
                g_a.append(np.round(a[i2],2))
                g_g.append(np.round(g[i2],0))

    j_ag = np.vstack((g_a, g_g))
    j = np.vstack((j, j_ag))

j = np.delete(j, 0, 0)
```

In [13]:

```
name_col = list(base_dados.keys())

new_name = []
for i in range(len(name_col)*2):

    if i%2 == 0:
        new_name.append(name_col[i//2] + '(A)')
    else:
```

```
        new_name.append(name_col[i//2] + '(g)')

df = pd.DataFrame(j.T, index= name_order, columns = new_name)
df
```

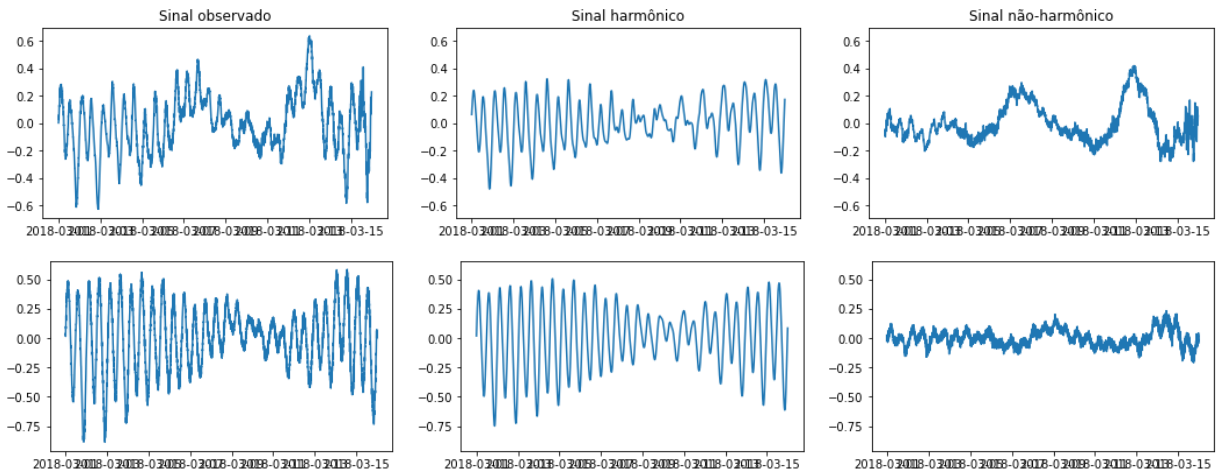
Out[13]:

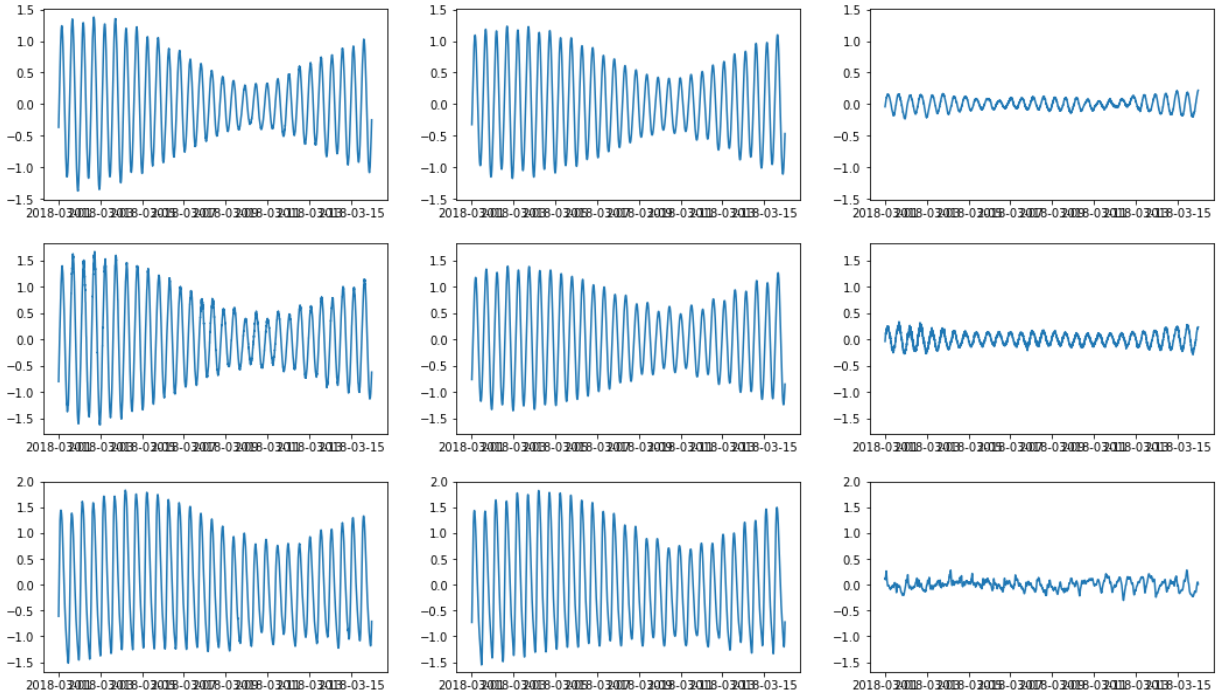
	IMB(A)	IMB(g)	ARC(A)	ARC(g)	SAL(A)	SAL(g)	FOR(A)	FOR(g)	SAN(A)	SAN(g)
MSF	0.05	69.0	0.04	37.0	0.03	276.0	0.03	311.0	0.26	306.0
O1	0.08	230.0	0.08	244.0	0.04	277.0	0.05	338.0	0.05	144.0
K1	0.04	196.0	0.04	212.0	0.03	288.0	0.04	285.0	0.04	59.0
M2	0.14	264.0	0.33	281.0	0.80	317.0	0.96	337.0	1.22	322.0
S2	0.13	168.0	0.20	190.0	0.37	231.0	0.38	256.0	0.29	253.0
M4	0.04	56.0	0.03	73.0	0.02	315.0	0.01	44.0	0.27	213.0

Visualiza a decomposição do sinal original, harmônico e não harmônico

In [14]:

```
g_nivel_nao_h = []

for i in range(5):

    nivel_h = g_sinal_h[i]['h']

    nivel_nao_h = nivel_i[:,i] - nivel_h

    g_nivel_nao_h.append(nivel_nao_h)

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 3))
    ax1.plot(tempo_ref, nivel_i[:,i])

    ylims = ax1.get_ylim()

    ax2.plot(tempo_ref, nivel_h)
    ax2.set_ylim(ylims)

    ax3.plot(tempo_ref, nivel_nao_h)
    ax3.set_ylim(ylims)

    if i == 0:
        ax1.set_title('Sinal observado')
        ax2.set_title('Sinal harmônico')
        ax3.set_title('Sinal não-harmônico')
```





Aplica filtro Butterworth para decompor as bandas sub-mareal, mareal e supra-mareal

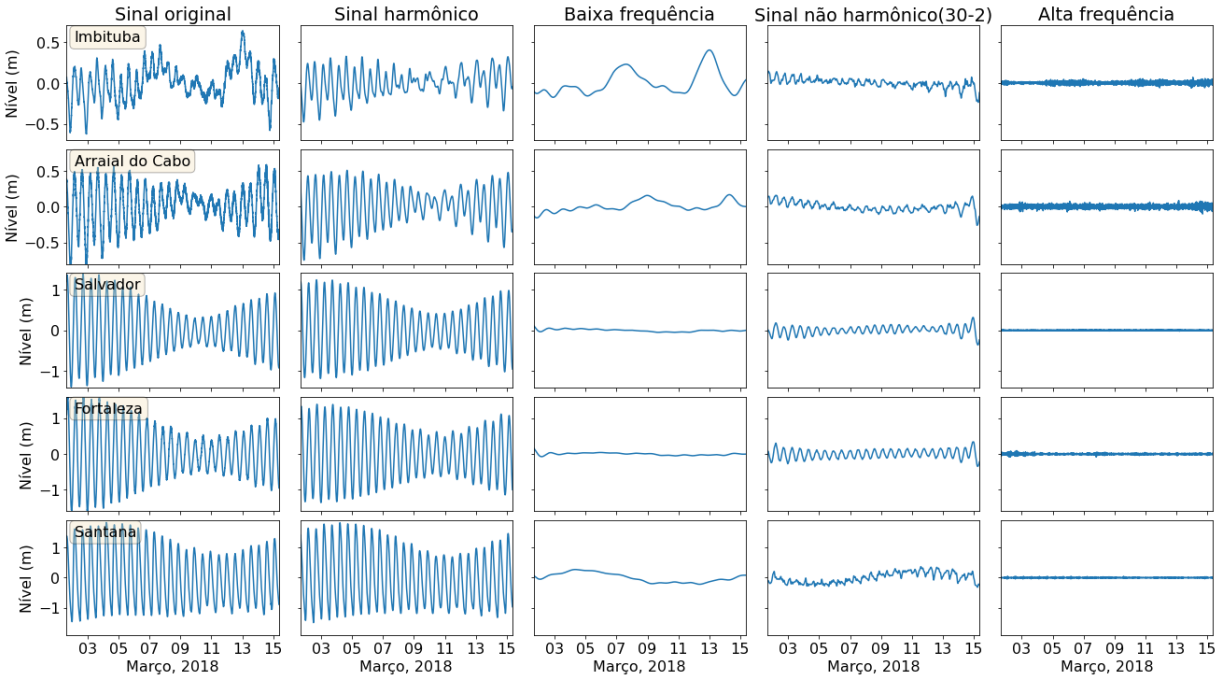
In [15]:

```
frequencia_amostral = 12 # 12 por hora, ou a cada 5 minutos!
ordem_filtro = 5
frequencia_corte_baixa = 1/30    # = periodo de 30 horas
frequencia_corte_alta = 1/2      # = período de 2 horas

# delineamento filtro passa baixa
B_30, A_30 = signal.butter(ordem_filtro, frequencia_corte_baixa,
                           fs = frequencia_amostral,btype  = 'lowpass')

# delineamento do filtro passa alta
B_2, A_2 = signal.butter(ordem_filtro, frequencia_corte_alta,
                           fs = frequencia_amostral, btype  = 'highpass')

# delineamento do filtro passa banda
B_30_2, A_30_2 = signal.butter(ordem_filtro,
                           [frequencia_corte_baixa, frequencia_corte_alta],
                           fs = frequencia_amostral, btype  = 'band')

def filtra_series(serie):
    serie_f_30 = signal.filtfilt(B_30, A_30, serie)
    serie_f_30_2 = signal.filtfilt(B_30_2, A_30_2, serie)
    serie_f_2 = signal.filtfilt(B_2, A_2, serie)
    serie_f = [serie_f_30, serie_f_30_2, serie_f_2]
    return serie_f

g_niveis = []
for i in range(5):
    org_niveis = []
    nivel = nivel_i[:,i]                                    # nivel interpolado, sina
    nivel_f = filtra_series(nivel)                          # nivel filtrado nas band
    nivel_h = g_sinal_h[i]['h']                             # recupera nível harmônic
    nivel_nh = np.array(nivel_f[1]) - np.array(nivel_h)     # calcula o nível não har

    org_niveis.append(nivel)      # organiza o ordem das séries na lista de saída!
    org_niveis.append(nivel_h)
    org_niveis.append(nivel_f[0])
    org_niveis.append(nivel_nh)
    org_niveis.append(nivel_f[2])
```

```
    g_niveis.append(org_niveis)
```

In [16]:

```
tempo_ref = tempo_ref.tolist()
plt.rcParams.update({'font.size': 16})
fig, axs = plt.subplots(5,5, figsize=(22,12))
fig.subplots_adjust(hspace = .08, wspace=.1)

estacoes = ['Imbituba', 'Arraial do Cabo', 'Salvador', 'Fortaleza', 'Santana']
xlims = [tempo_ref[0] + datetime.timedelta(hours=15), tempo_ref[-1] - datetime.timed
ylims = [[-.7, .7], [-.8, .8], [-1.4, 1.4], [-1.6, 1.6], [-1.9, 1.9]]
titles = ['Sinal original', 'Sinal harmônico', 'Baixa frequência', 'Sinal não harmôn

axs = axs.ravel()

props = dict(boxstyle='round', facecolor='wheat', alpha=0.3)
ax = 0
for li in range(5):

    for co in range(5):

        p_niveis = g_niveis[li]

        p_niveis2 = p_niveis[co]

        axs[ax].plot(tempo_ref, p_niveis2)
        axs[ax].set_xlim(xlims)
        axs[ax].set_ylim(ylims[li])

        if li < 4:
            axs[ax].set_xticklabels('')
        if co > 0:
            axs[ax].set_yticklabels('')
        if co == 0:
            axs[ax].set_ylabel('Nível (m)')
            axs[ax].text(xlims[0] + datetime.timedelta(hours = 12), ylims[li][1] - y
        if li == 4:
            axs[ax].xaxis.set_major_formatter(mdates.DateFormatter('%d'))
            axs[ax].set_xlabel('Março, 2018')
        if li == 0:
            axs[ax].set_title(titles[co])

        ax += 1
```



Elimina o início e fim das séries e calcula a variância de cada componentes

In [17]:

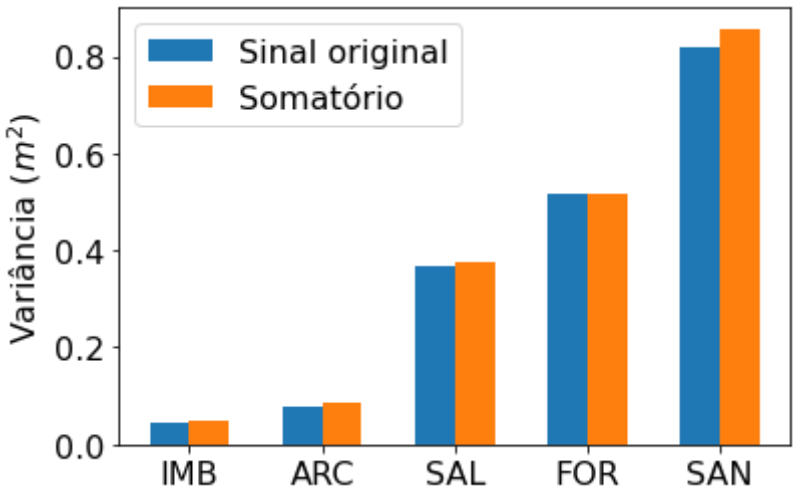
```
tempo_ref = np.array(tempo_ref)
td_15_hs = datetime.timedelta(hours = 15)

tempo_ref_2 = tempo_ref[(tempo_ref > tempo_ref[0] + td_15_hs) & (tempo_ref < tempo_r

g_var = np.zeros((5, 5))
g_var[:] = np.nan

g_niveis_2 = []
for li in range(5):

    lista_temporaria = []
    var = []

    for co in range(5):
        pega_lista = g_niveis[li][co]

        pega_lista = pega_lista[(tempo_ref > tempo_ref[0] + td_15_hs) & (tempo_ref <

        calc_var = np.var(pega_lista)

        g_var[li, co] = calc_var

        lista_temporaria.append(pega_lista)


    g_niveis_2.append(lista_temporaria)
```

In [18]:

```
sinal_o = g_var[:,0]
sinal_d = np.sum(g_var[:,1:], axis=1)

labs = list(base_dados.keys())
labs.insert(0, '')
xlabs = np.arange(0, 5)

fig, ax = plt.subplots()
ax = plt.gca()
```

```
ax.bar(xlabs - .15, sinal_o, width=.3, label='Sinal original')
ax.bar(xlabs + .15, sinal_d, width=.3, label='Somatório ')
ax.set_xticklabels(labs)
ax.set_ylabel('Variância ($m^2$)')

plt.legend()
plt.show()
```

```
<ipython-input-18-66eebe275904>:13: UserWarning: FixedFormatter should only be used
together with FixedLocator
  ax.set_xticklabels(labs)
```



In [19]:
```
sinal_d = np.atleast_2d(sinal_d).T

var_percentual = g_var[:,1:]/sinal_d*100
```

In [20]:
```
np.set_printoptions(suppress=True)

print(np.round(var_percentual, 2))
```

```
[[51.18 42.33  6.08  0.41]
 [88.95  5.79  4.78  0.48]
 [97.23  0.23  2.53  0.01]
 [97.01  0.19  2.76  0.04]
 [93.95  2.69  3.36  0.01]]
```

In [21]:
```
df = pd.DataFrame(np.round(var_percentual, 2), index=labs[1:],
                  columns = titles[1:])
df.style.set_properties(**{'text-align': 'center'})
df
```

Out[21]:

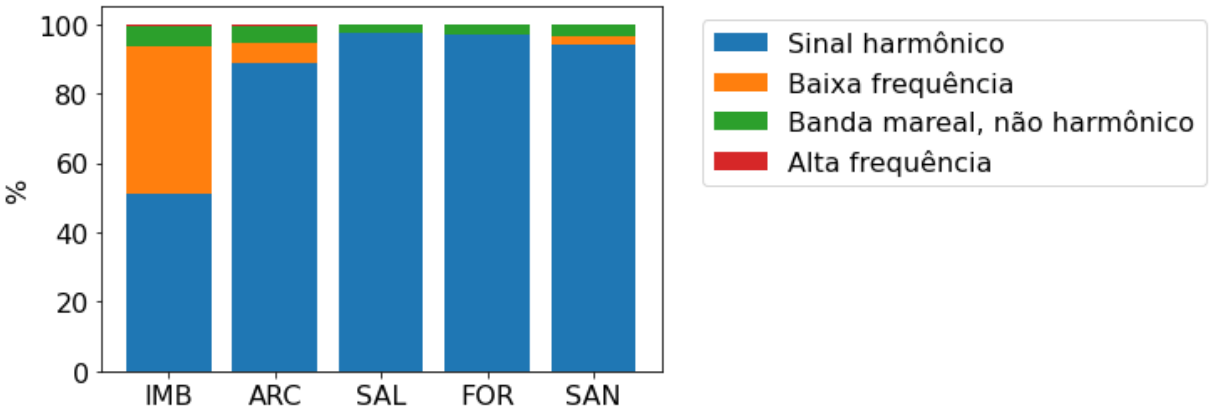
	Sinal harmônico	Baixa frequência	Sinal não harmônico(30-2)	Alta frequência
IMB	51.18	42.33	6.08	0.41
ARC	88.95	5.79	4.78	0.48
SAL	97.23	0.23	2.53	0.01
FOR	97.01	0.19	2.76	0.04
SAN	93.95	2.69	3.36	0.01

In [22]:
```
fig, ax = plt.subplots()
```

```
x= np.arange(0,5)

ax.bar(x, var_percentual[:,0], label = 'Sinal harmônico')
ax.bar(x, var_percentual[:,1], bottom=var_percentual[:,0], label = 'Baixa frequência
ax.bar(x, var_percentual[:,2], bottom=var_percentual[:,0] + var_percentual[:,1], lab
ax.bar(x, var_percentual[:,3], bottom=var_percentual[:,0] + var_percentual[:,1] + va

ax.legend(bbox_to_anchor=(1.05, 1))
ax.set_xticklabels(labs)
ax.set_ylabel('%')
```

```
<ipython-input-22-77ff44433c64>:11: UserWarning: FixedFormatter should only be used
together with FixedLocator
  ax.set_xticklabels(labs)
```

Out[22]: Text(0, 0.5, '%')



Armazena os resultados

In [23]:

```
dados_ibge_processados = [tempo_ref_2, g_niveis_2]

with open('Ibge_base_dados_nivel2.pik', 'wb') as file:

    pickle.dump(dados_ibge_processados, file)
```

to be continued...