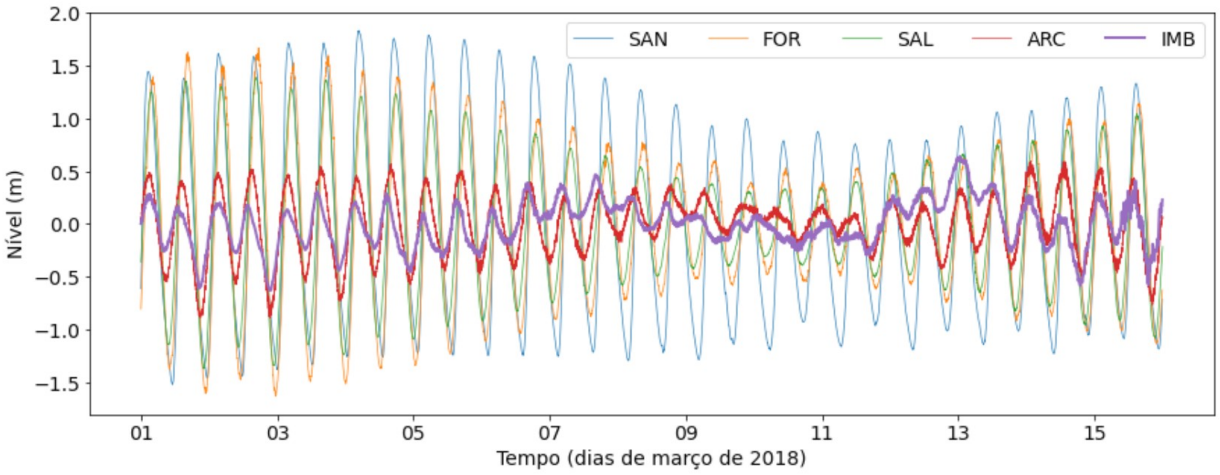
Analisando dados com Python

Fazendo um gráfico de marés observadas pelo RPMG/IBGE.

Parte 2

V2. 2020.01.22



Carlos A.F. Schettini

Laboratório de Oceanografia Costeira e Estuarina

LOCostE / FURG

Objetivos:

1 Carregar dados de vários arquivos com uma ‘function’

2 Preparar as variáveis

3 Visualizar graficamente

Parte 2 – Comparando dados maré de várias estações

Na primeira parte do tutorial de marés vimos como obter dados de nível da água da Rede Maregráfica Permanente para Geodésia (RMPG) do Instituto Brasileiro de Geografia e Estatística (IBGE), carregar no Juyter, manipular o conteúdo e gerar um gráfico. Neste tutorial vamos pegar vários dias de dados e de várias estações para visualizar.

Os dados no site to IBGE (https://www.ibge.gov.br/geociencias/informacoes-sobre-posicionamento-geodesico/rede-geodesica/10842-rmpg-rede-maregrafica-permanente-para-geodesia.html?=&t=downloads) não permite indicar uma pasta para salvar, e a estrutura de armazenamento consiste de diretórios sob ‘dados’ com pastas indicadas pelo mês e dia (ex: 0104 significa janeiro dia 4)(Figura 1). Entrando em uma destes diretórios os dados estão em arquivos ‘.zip’, com os três primeiros caracteres indicando a estação, depois dois números indicando o ano, outros dois para o mês e outros dois para o dia. Assim, na mesma pasta tem dados de vários anos para o dia 4 de janeiro. Meio confuso… mas enfim.

Clicando nos arquivos eles são automaticamente descarregados para a pasta de downloads do seu computador. Então, para pegar um mês de dados de cada estação, vai aí um certo trabalho e atenção! Vou pegar Setembro de 2020. Setembro porque é período equinoxial e as marés são extremas, e 2020 na esperança que a maior parte dos registros estejam bons.

< interlúdio investigando os site do IBGE >

Escrutinando melhor o site, na opção 'Dados Diários' encontrei uma descrição melhor de como estão organizados os dados. Acima foi 'intuição'. E há uma ferramenta que permite ver se os dados estão disponíveis ou não. Assim, consultando para setembro de 2020 vi que não há dados para Fortaleza nem Salvador. Então, tentando, achei que para março (também é equinocial, em caso de dúvida!) de 2018 há dados para Imbituba (Região Sul), Arraial do Cabo (Região Sudeste), Salvador (Região Nordeste, costa lestes), Fortaleza (Região Nordeste, costa norte), e Santana (Região Norte). Aqui os arquivos já parecem em uma tabela o que torna muito mais simples baixar os dados. <outro interlúdio> Ledo engano… não dá para baixar os arquivos por aqui. Ou seja, temos que ir pasta por pasta, encontrar os arquivos de cada estação e para o ano que queremos e baixar. (!).

Figura 1: Disponibilização de dados de nível da água no site do IBGE.

Com paciência e algum tempo depois (uns 20 minutos), baixei os dados das cinco estações para o período de 1 até 15 de março de 2018. O ideal seria pegar um mês, mas 15 dias resolve. Movi todos os arquivos para um diretório ‘Mare_IBGE’, sobre o meu diretório de trabalho. Agora, ao Jupyter.

Primeiramente precisamos descompactar os arquivos. Em vez de fazer um a um manualmente, podemos usar o Python para isso. No tutorial do ‘CTD’ vimos o pacote ‘os’ que fornece acesso ao sistema de diretórios e arquivos. E com o pacote ‘zipfile’ podemos acessar o

conteúdo de um arquivo ‘.zip’ e fazer a extração, e no embalo despachar o arquivo zipado (Código 1).

Analisa maré comparativa do IBGE/RMPG

2021 Jan 22

[2]:
```
import os
from zipfile import ZipFile
```

[12]:
```
path = r'c:\GUTO\Academia\Aulas_AnalisedeDados\Mare_IBGE\Dados_mare_IBGE\\'
mydir = os.listdir(path)
```

[13]:
```
for file in mydir:

    with ZipFile(path + file, 'r') as zip:

        zip.extractall(path)

        zip.close()

        os.remove(path + file)
```

Código 1: Descompactando e removendo arquivos.

Como bom observador tu deves ter percebido as células foram executadas diversas vezes, pois esta é a primeira vez que eu uso o pacote ‘zipfile’, então foram algumas tentativas até dar certo. Aqui se deve ter muita atenção: o conteúdo do diretório foi alterado, e se executar novamente vai dar erro. Antes todos os arquivos que etavam lá eram ‘.zip’, e depois de executar, todos os arquivos são ‘.txt’. A célula de código da extração dos zips e sua remoção não tem mais utilidade e pode ser deletada. Mas, para fins didádicos, e para quando no futuro precisar fazer isso de novo e lembrar como faz, vamos apenas deixá-la comentada.

Agora temos todos os arquivos em formato ‘.txt’ na pasta. No outro tutorial de marés vimos como processar um destes arquivos (Código 2). Agora temos 75 arquivos. Uma opção é

repetir a célula do processamento 75 vezes, uma para cada arquivo e ver um jeito de juntar os resultados. Ou converter esse procedimento em uma função (Código 3), como usamos com os dados de CTD. E, lembrando. Basta usar o '`def`' para definir a função, dar um nome que não te cause problemas, e no final incluir o '`return`' com o que deve ser devolvido ao programa. Cuidado com a tabulação! Uma vez função, tu podes testar para ver se está tudo bem! Carregue outros arquivos e faça outros gráficos. E uma função pode ser chamada dentro de outra função. Então, tente fazer uma função que gere o gráfico do nível da água bastando indicar o arquivo de entrada!

Uma outra coisa… o código (2) foi um pouco modificado. Testando com arquivos de outras estações eu descobri que o número de colunas pode variar. Alguns tem dois registros de nível, como em Imbituba, mas outros têm somente um. Para que o código fique flexível para aceitar as duas condições de entrada, foi adicionado um teste '`if`' que verifica o número de colunas. Se não há a oitava coluna, é inserido um 'not-a-number' '`np.nan`'. O 'nan' não existe no Python nativo, e é uma ferramenta do numpy, daí o 'np'.

[3]:

```
guarda_linhas = []

for linha in linhas:

        linha_quebrada = linha.split()

        ldata = linha_quebrada[0].split('/')
        lhora = linha_quebrada[1].split(':')
        var1 = linha_quebrada[2].replace(',', '.')

        dia = int(ldata[0])
        mes = int(ldata[1])
        ano = int(ldata[2])
        hora = int(lhora[0])
        minuto = int(lhora[1])
        segundo = int(lhora[2])
        var1 = float(var1)

        if len(linha_quebrada) == 8:
            var2 = linha_quebrada[3].replace(',', '.')
            var2 = float(var2)
        else:
            var2 = np.nan

        nova_linha = [ano, mes, dia, hora, minuto, segundo, var1, var2]

        guarda_linhas.append(nova_linha)

```

Código 2: Processamento de um arquivo de maré do IBGE para 7 ou 8 colunas de dados.

[137]:

```python
def ajeita_dados(linhas):
    guarda_linhas = []

    for linha in linhas:

        linha_quebrada = linha.split()

        ldata = linha_quebrada[0].split('/')
        lhora = linha_quebrada[1].split(':')
        var1 = linha_quebrada[2].replace(',', '.')

        dia = int(ldata[0])
        mes = int(ldata[1])
        ano = int(ldata[2])
        hora = int(lhora[0])
        minuto = int(lhora[1])
        segundo = int(lhora[2])
        var1 = float(var1)

        if len(linha_quebrada) == 8:
            var2 = linha_quebrada[3].replace(',', '.')
            var2 = float(var2)
        else:
            var2 = np.nan

        nova_linha = [ano, mes, dia, hora, minuto, segundo, var1, var2]
        guarda_linhas.append(nova_linha)

    return guarda_linhas
```

Código 3: função para processar dados de maré do IBGE.

Uma parte do nosso problema é que temos multiplos arquivos para carregar, processar e gerar gráficos. Nós sabemos que os três primeiros caracteres dos nomes dos arquivos indicam a estação, e só podem ser as strings ‘IMB’, ‘ARC’, ‘SAL’, ‘FOR’ e ‘SAN’ e mais nenhuma outra. Então podemos usar este regra para separar os dados e juntar todos os dados de uma mesma estação me uma lista, resultando ao final cinco listas.

Sempre há muitos caminhos que podem ser utilizados, e qual utilizar vai da habilidade do analista e do custo-benefício da implementação. A solução mais explícita e fazer uma cadeia de testes. Se as três primeiras letras do arquivo começam com ‘IMB’, então eu guardo aqui, senão

se começam com ‘ARC’, eu guardo ali, e assim por diante. E para guardar, tem que ter uma variável para cada estação para isso (Codigo 4).

[154]:
```
IMB = []
ARC = []
SAL = []
FOR = []
SAN = []

for file in mydir:

    with open(path + file) as f:

        conteudo = f.readlines()
        dados = ajeita_dados(conteudo)

    if file[0:3] == 'IMB':

        IMB.extend(dados)

    elif file[0:3] == 'ARC':

        ARC.extend(dados)

    elif file[0:3] == 'SAL':

        SAL.extend(dados)

    elif file[0:3] == 'FOR':

        FOR.extend(dados)

    elif file[0:3] == 'SAN':

        SAN.extend(dados)

print(len(IMB), len(ARC), len(SAL), len(FOR), len(SAN))
```

```
4320 21600 7776 4320 4317
```

Código 4: Carregando, processando e separando os dados, e apresentando o tamanho.

Aparentemente está funcionado (só depois de ver o gráfico para ter certeza). Este tipo de abordagem funciona, mas não é a ideal. Se houvesse 15 estações, teriam que haver 15 testes, e mais chances de erros e deterioração da leitura do código, pelo tamanho. Aqui apresento para vocês um outro objeto da fauna Pythoniana (também de outras linguagens) que são os dicionários. Confesso que eu não sou muito familiarizado com dicionários, mas de vez em quando eles realmente ajudam. Um dicionário é uma maneira de atrelar diversos conteúdos em

diversas variáveis (keys). Veja alguns tutoriais sobre isso. No nosso caso, temos que inicialmente criar as chaves (keys) às quais serão associados conteúdos. Uma lista de strings que serão tanto a chave (nome da variável) quanto parâmetro para testar a qual estação será adicionado os dados de cada arquivo. Para criar um dicionário basta usar chaves {} como '`dicionario = {}`'. E, como usaremos ele dentro de um loop, devemos informar quais das chaves esperadas (linhas 4 e 5 do codigo 5). Depois o que muda é que teremos um único teste ao invés de cinco, mas temos um loop interno adicional para mudar o parâmetro do teste e a chave onde será armazenado os dados. Dizemos que este código é mais 'elegante' do que o anterior. O outro é 'força bruta'. Se tivermos 15 estações, basta incluir as strings adicionais em '`estacoes`'. O lado negativo deste código é a sua leitura que requer um maior grau de abstração.

Agora temos um dicionário 'base_dados' que têm cinco chaves cujos nomes são os acrônimos das estações em letras maiúsculas. A cada chave está associada uma lista que contém o conteúdo de todos os arquivos cujas primeiras três letras são iguais ao nome da chave. E, cada elemento de cada uma destas listas é uma lista em si com 8 elementos. Não vemos isso, mas temos que ter essa abstração. É isso ou planilha eletrônica! Podemos investigar as propriedades do dicionário tal como faríamos com uma lista, mas sendo um objeto dicionário, há métodos adicionais que podem ser utilizados (Código 6).

[155]:

```
estacoes = ['IMB', 'ARC', 'SAL', 'FOR', 'SAN']

base_dados = {}
for k in estacoes:
    base_dados[k]=[]

for file in mydir:

     with open(path + file) as f:

        conteudo = f.readlines()
        dados = ajeita_dados(conteudo)

        for estacao in estacoes:

            if file[0:3] == estacao:

                base_dados[estacao].extend(dados)

for est in estacoes:
    print(len(base_dados[est]))
```

```
4320
21600
7776
4320
4317
```

Código 5: O mesmo que o Código 4, porém usando dicionário.

[193]:

```
print(type(base_dados))
print(len(base_dados))
print(base_dados.keys())
for est in estacoes:
    print('Tamanho da chave', est, '=',  len(base_dados[est]))
```

```
<class 'dict'>
5
dict_keys(['IMB', 'ARC', 'SAL', 'FOR', 'SAN'])
Tamanho da chave IMB = 4320
Tamanho da chave ARC = 21600
Tamanho da chave SAL = 7776
Tamanho da chave FOR = 4320
Tamanho da chave SAN = 4317
```

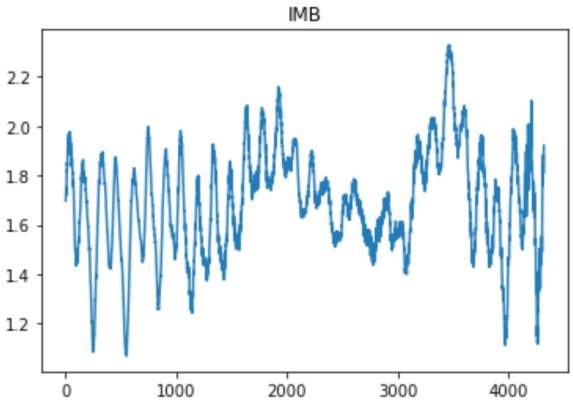
Código 6: Propriedades do dicionário 'base_dados'.

Agora podemos dar uma primeira espiadela nos dados que carregamos. Como os dados estão ordenados em colunas, a maneira mais simples é converter o conteúdo das chaves do dicionário em arranjos do numpy. Ou seja, a lista de listas será transformada em um arranjo 2D o

qual eu posso fatiar. Sendo um arranjo, podemos usar o método ‘shape()’ do numpy para indicar quantas linhas e quantas colunas há (contando a partir do 1!)(Código 7).

[201]:
```
1 est = 0
2 d = np.array(base_dados[estacoes[est]])
3 print(d.shape)
4
5 plt.plot(d[:,6])
6 plt.title(estacoes[est])
7 plt.show()
```

```
(4320, 8)
```



Código 7: Dando uma primeira olhada nos dados.

Essa primeira olhada nos dados é importante para ver se os dados estão com cara de dados. Sabemos que são dados de registros do nível do mar para observações de marés, então temos a expectativa de ver oscilações de maré com cara de maré. E são 15 dias, o que necessariamente conterá umas 30 oscilações semi-diurnas e um ciclo de sizígia e quadratura. Se você nunca viu um registro de maré, não terá referência para dizer se isso tem cara de maré. Mas aí tem a internet pra ajudar! Pesquisando imagens de ‘tidal records’ vai ver alguns.

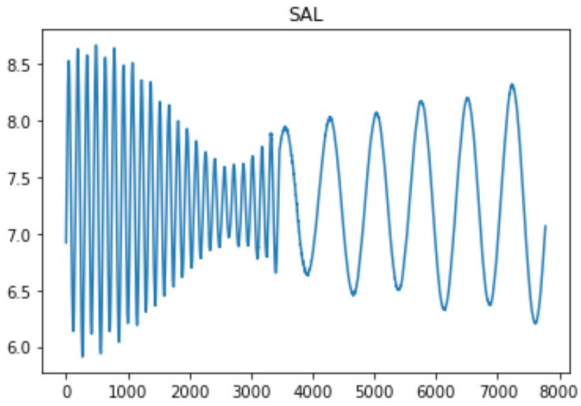
O registro de Imbituba tem cara de dados de maré. Atenção para o eixo x que no caso representa a número da linha da observação e não o tempo. Os dados de Arraial do Cabo, Fortaleza e Santana também parecem estar em ordem, a despeito das diferentes quantidades de

dados, o que indica diferentes taxas de aquisição. Mas os dados de Salvador estão destoando dos demais (Código 8). Em ~3500 há uma clara mudança do padrão da onda. Mas isso não é motivo para desespero! (se sua tese dependesse disto). Isto indica apenas uma mudança da taxa de aquisição de dados, que aumentou. Quando indexarmos os dados ao tempo, veremos se realmente há 15 dias completos.

[203]:

```
est = 2
d = np.array(base_dados[estacoes[est]])
print(d.shape)

plt.plot(d[:,6])
plt.title(estacoes[est])
plt.show()
```

```
(7776, 8)
```



Código 8: Dados da estação maregráfica de Salvador.

Já vimos antes como criar um objeto datetime. Fazemos (ou copiamos) a função que cria o objeto datetime a partir dos nossos dados (Código 10). Refazendo o gráfico com os dados da estação Salvador, mas agora com os dados indexados pelo tempo, vemos que os 15 dias de dados estão lá (sua tese está salva!)(Código 10). E não se preocupe com a borradeira das legendas do eixo x. Arrumaremos depois.

[207]:

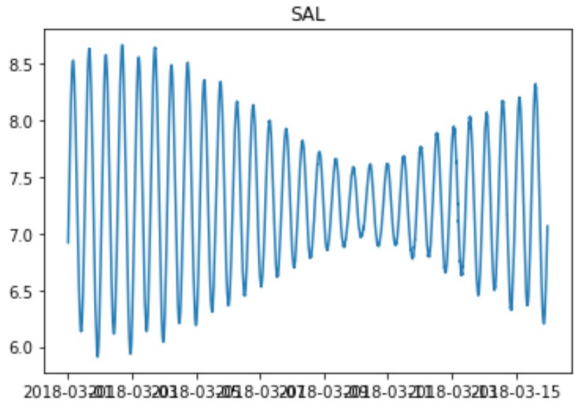
```
def monta_datetime(d):

    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
```

Código 9: Função para criar objeto datetime.

[208]:

```
1 est = 2
2 d = np.array(base_dados[estacoes[est]])
3
4 t = monta_datetime(d)
5
6 plt.plot(t, d[:,6])
7 plt.title(estacoes[est])
8 plt.show()
```

```
(7776, 8)
```

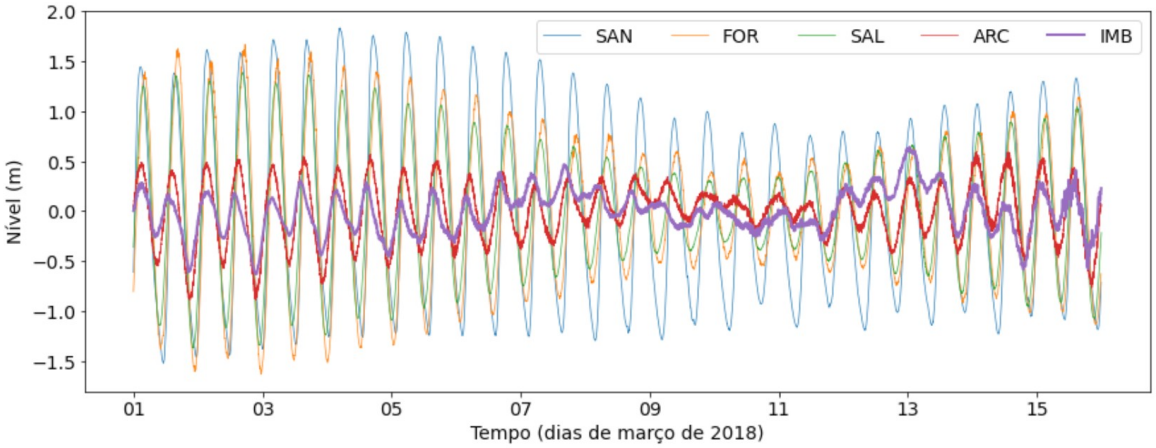


Código 10: Dados da estação Salvador indexados pelo tempo.

Agora o grand finale é fazer o gráfico com as cinco estações (Código 11). A parte do código de geração do gráfico, a inversão da ordem das estações é para realçar Imbituba e Arraial do Cabo, ficando na frente das outras. Já disse antes que saber fazer gráficos elegantes e informativos é muito importante. E a elaboração de gráficos no Python é quase que uma programação a parte! O matplotlib é o pacote mais genérico para fazer gráficos, mas há outros (https://www.tutorialdocs.com/article/best-python-graphics-libraries.html) Código : Código para geração do gráfico final.

[242]:

```
estacoes = ['SAN', 'FOR', 'SAL', 'ARC', 'IMB']

fig = plt.figure(figsize=(14,5))
ax = fig.add_axes([.1, .1, .8, .8])

for est in range(len(estacoes)):

    d = np.array(base_dados[estacoes[est]])

    t = monta_datetime(d)

    lw = .7
    if estacoes[est] == 'IMB':
        lw = 2

    ax.plot(t, d[:,6] - np.mean(d[:,6]), linewidth=lw, label=estacoes[est])

formato = mdates.DateFormatter('%d')
ax.xaxis.set_major_formatter(formato)

ax.set_ylabel('Nível (m)')
ax.set_xlabel('Tempo (dias de março de 2018)')

plt.rcParams.update({'font.size': 14})
plt.legend(ncol=5)
plt.show()
```



Código 11: Geração da figura final.

Anexos:

Notebook completo depurado.

Atenção à parte de leitura dos arquivos zipados, que foi comentada.

Analisa maré comparativa do IBGE/RMPG

2021 Jan 22

[1]:
```
import os
# from zipfile import ZipFile
import datetime
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

[2]:
```
path = r'c:\GUTO\Academia\Aulas_AnalisedeDados\Mare_IBGE\Dados_mare_IBGE\\'
mydir = os.listdir(path)
```

[3]:
```
## descompacta e remove os arquivos baixados da internet!
# for file in mydir:
#     with ZipFile(path + file, 'r') as zip:
#         zip.extractall(path)
#         zip.close()
#         os.remove(path + file)
```

[4]:

```python
def ajeita_dados(linhas):
    guarda_linhas = []

    for linha in linhas:

        linha_quebrada = linha.split()

        ldata = linha_quebrada[0].split('/')
        lhora = linha_quebrada[1].split(':')
        var1 = linha_quebrada[2].replace(',', '.')

        dia = int(ldata[0])
        mes = int(ldata[1])
        ano = int(ldata[2])
        hora = int(lhora[0])
        minuto = int(lhora[1])
        segundo = int(lhora[2])
        var1 = float(var1)

        if len(linha_quebrada) == 8:
            var2 = linha_quebrada[3].replace(',', '.')
            var2 = float(var2)
        else:
            var2 = np.nan

        nova_linha = [ano, mes, dia, hora, minuto, segundo, var1, var2]
        guarda_linhas.append(nova_linha)

    return guarda_linhas
```

[5]:

```
IMB = []
ARC = []
SAL = []
FOR = []
SAN = []

for file in mydir:

    with open(path + file) as f:

        conteudo = f.readlines()
        dados = ajeita_dados(conteudo)

    if file[0:3] == 'IMB':

        IMB.extend(dados)

    elif file[0:3] == 'ARC':

        ARC.extend(dados)

    elif file[0:3] == 'SAL':

        SAL.extend(dados)

    elif file[0:3] == 'FOR':

        FOR.extend(dados)

    elif file[0:3] == 'SAN':

        SAN.extend(dados)

print(len(IMB), len(ARC), len(SAL), len(FOR), len(SAN))
```

```
4320 21600 7776 4320 4317
```

[6]:

```
estacoes = ['IMB', 'ARC', 'SAL', 'FOR', 'SAN']

base_dados = {}
for k in estacoes:
    base_dados[k]=[]

for file in mydir:

     with open(path + file) as f:

        conteudo = f.readlines()
        dados = ajeita_dados(conteudo)

        for estacao in estacoes:

            if file[0:3] == estacao:

                base_dados[estacao].extend(dados)

for est in estacoes:
    print(len(base_dados[est]))
```

```
4320
21600
7776
4320
4317
```

[7]:

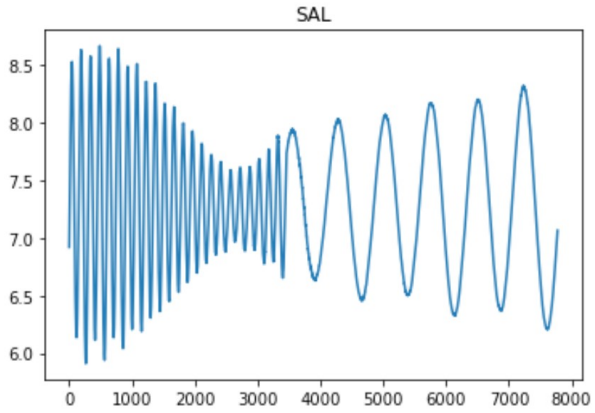
```
print(type(base_dados))
print(len(base_dados))
print(base_dados.keys())
for est in estacoes:
    print('Tamanho da chave', est, '=',  len(base_dados[est]))
```

```
<class 'dict'>
5
dict_keys(['IMB', 'ARC', 'SAL', 'FOR', 'SAN'])
Tamanho da chave IMB = 4320
Tamanho da chave ARC = 21600
Tamanho da chave SAL = 7776
Tamanho da chave FOR = 4320
Tamanho da chave SAN = 4317
```

[8]:

```
est = 2
d = np.array(base_dados[estacoes[est]])
print(d.shape)

plt.plot(d[:,6])
plt.title(estacoes[est])
plt.show()
```

```
(7776, 8)
```



[9]:

```
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
```

[10]:

```
estacoes = ['SAN', 'FOR', 'SAL', 'ARC', 'IMB']

fig = plt.figure(figsize=(14,5))
ax = fig.add_axes([.1, .1, .8, .8])

for est in range(len(estacoes)):

    d = np.array(base_dados[estacoes[est]])

    t = monta_datetime(d)

    lw = .7
    if estacoes[est] == 'IMB':
        lw = 2

    ax.plot(t, d[:,6] - np.mean(d[:,6]), linewidth=lw, label=estacoes[est])

formato = mdates.DateFormatter('%d')
ax.xaxis.set_major_formatter(formato)

ax.set_ylabel('Nível (m)')
ax.set_xlabel('Tempo (dias de março de 2018)')

plt.rcParams.update({'font.size': 14})
plt.legend(ncol=5)
plt.show()
```