Aula 9 - matplotlib e pyplot

Karl Jan Clinckspoor

2 de julho de 2018

Sumário

1 Introdução

matplotlib é um enorme pacote para a criação de gráficos em Python. Sobre o matplotlib, foram criadas algumas funções para facilitar seu uso, e um desses pacotes é o pyplot, interno ao matplotlib. A sintaxe do pyplot é semelhante à do Matlab, propositalmente. É uma tarefa impossível tentar cobrir tudo o que o matplotlib pode oferecer. Este vídeo possui bastante informação sobre o pyplot, e vale a pena dar uma olhada.

O site oficial do matplotlib é matplotlib.org, onde há informações sobre muitas das funções e opções dentro do matplotlib. Também veja a galeria, que mostra vários gráficos de exemplo, e o código utilizado para criá-los, caso você tenha algum desejo específico e não consiga criar algo semelhante. Esta página contém um resumo sobre o que o *pyplot* oferece.

1.1 Importação

Geralmente, abreviamos o pyplot como *plt*.

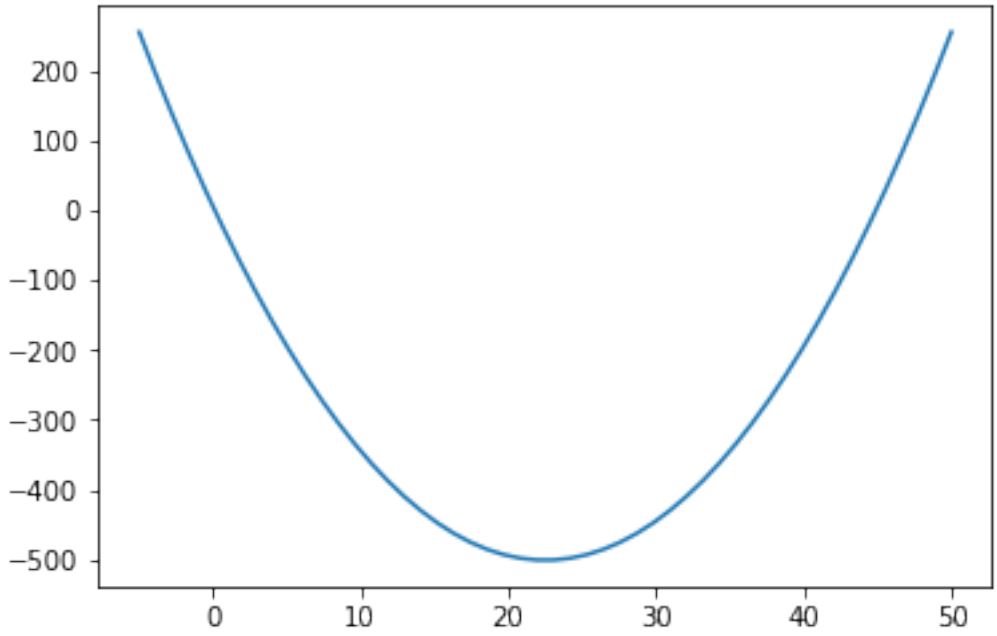
```
In [1]: import matplotlib.pyplot as plt
```

2 Primeiro plot - básicos

Para gerar um gráfico, devemos possuir valores para x e y. Vamos utilizar numpy para gerar alguns vetores de dados. Depois, vamos utilizar a função *plot* para criar um gráfico. Veja mais detalhes sobre essa função aqui.

```
In [2]: import numpy as np

        x = np.linspace(-5, 50)
        y = x ** 2 - 45 * x + 5

        plt.plot(x, y)
```

```
Out[2]: [<matplotlib.lines.Line2D at 0x7f8222ceb828>]
```



Veja que as figuras já apareceram no Jupyter Notebook. Isso facilita bastante a organização dos dados. Caso isso não ocorra, execute o seguinte código numa cela de Python:

```
%matplotlib inline
```

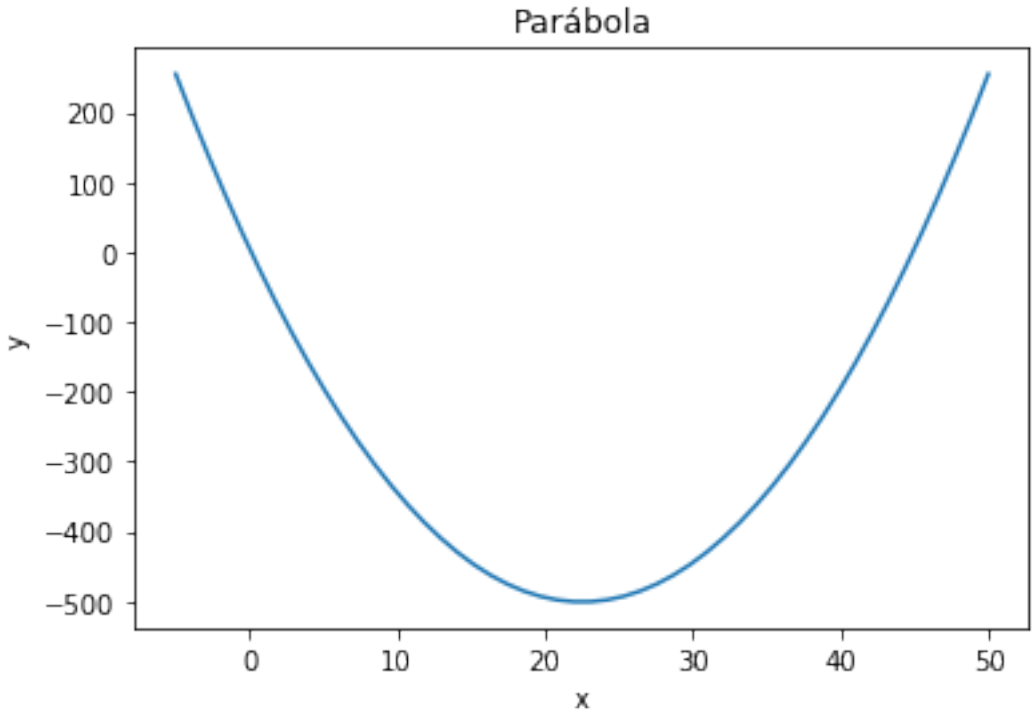
Você já viu isso anteriormente, com o comando %load. Esses são chamados de *magic commands*, e sempre começam com %. Uma lista dos magic commands disponíveis está aqui.

Nota importante: em Jupyter Notebooks, quando a execução de uma cela terminou e um gráfico foi criado, ele automaticamente mostra, caso o estilo seja inline. Caso não seja, como 'nbagg' (uma maneira com um pouco mais de interatividade), é necessário utilizar o comando plt.show() no final, para de fato fazer o gráfico aparecer.

2.1 Colocando nomes nos eixos

Vamos agora colocar nomes para os eixos x, y e título para o gráfico.

```
In [3]: plt.plot(x, y)

        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Parábola')

Out[3]: Text(0.5,1,'Parábola')
```

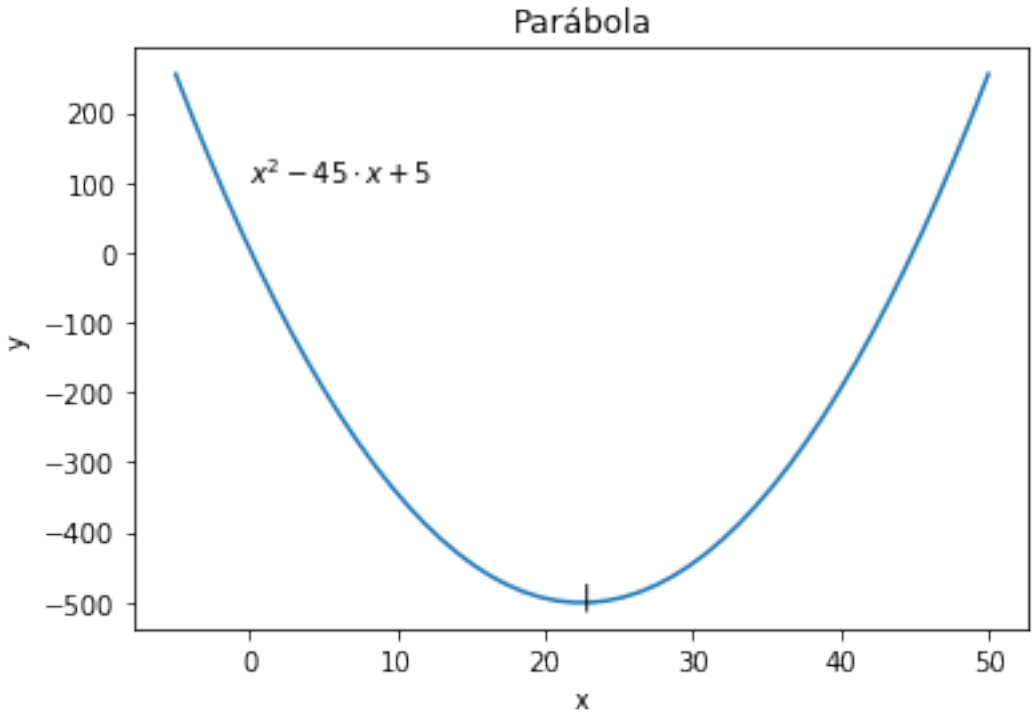


2.2 Colocando texto

Vamos agora localizar o mínimo dessa parábola, colocar uma marca nesse ponto, e colocar a equação da parábola num local da figura.

```
In [4]: plt.plot(x, y)
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Parábola')

        ymin = y.min()
        # Como x e y tem o mesmo comprimento, o valor da posição do mínimo em y
        # é o mesmo da posição de mínimo em x
        xmin = x[y.argmin()]

        plt.text(xmin, ymin, '|', horizontalalignment='right')
        plt.text(0, 100, r'$x^2 - 45 \cdot x + 5$')
        # Escolhidos a olho
        # Sintaxe do string é de matemática do LaTeX, pois está entre $$
```

```
Out[4]: Text(0,100,'$x^2 - 45 \\cdot x + 5$')
```



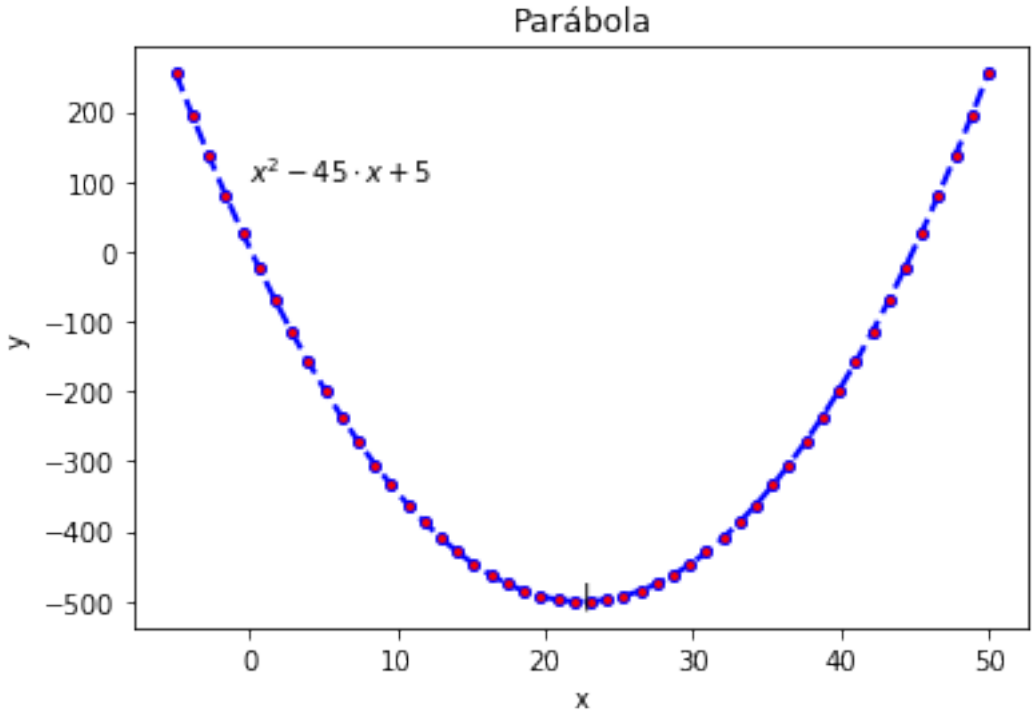
2.3 Customizando linhas

Vamos agora criar essa mesma figura com uma cor diferente e estilo diferente de linha.

```
In [5]: plt.plot(x, y, linewidth=2, linestyle='--',  color='b', marker='o', markersize=4, marker
        plt.xlabel('x')
        plt.ylabel('y')
```

```
plt.title('Parábola')

ymin = y.min()
# Como x e y tem o mesmo comprimento, o valor da posição do mínimo em y
# é o mesmo da posição de mínimo em x
xmin = x[y.argmin()]

plt.text(xmin, ymin, '|', horizontalalignment='right')
plt.text(0, 100, r'$x^2 - 45 \cdot x + 5$')  # Escolhidos a olho
```

```
Out[5]: Text(0,100,'$x^2 - 45 \\cdot x + 5$')
```



Bastante coisa foi alterada na linha para a criação dessa parábola. Os argumentos opcionais, após os posicionais de x e y, contém informações sobre o estilo do objeto linha. Esse objeto pode conter tanto linhas quanto símbolos. Se você tentar acessar as informações pelo Shift+Tab, verá algo muito menos útil do que, por exemplo, o read_csv do pandas. Para isso, é necessário buscar as informações na documentação online, por exemplo, nesta página.

Vamos item por item.

```
plt.plot(x, y, linewidth=2, linestyle='--',  color='b', marker='o', markersize=4, markerfacecolo
```

- linewidth: especifica a espessura da linha do gráfico, em *pontos*.
- linestyle: especifica o estilo da linha. Pode ser dado uma abreviação ou o nome do estilo.
  - '--'/'dashed'

  - 'solid'/'-' (padrão)
  - 'dashdot'/'-.'
  - 'dotted'/':'

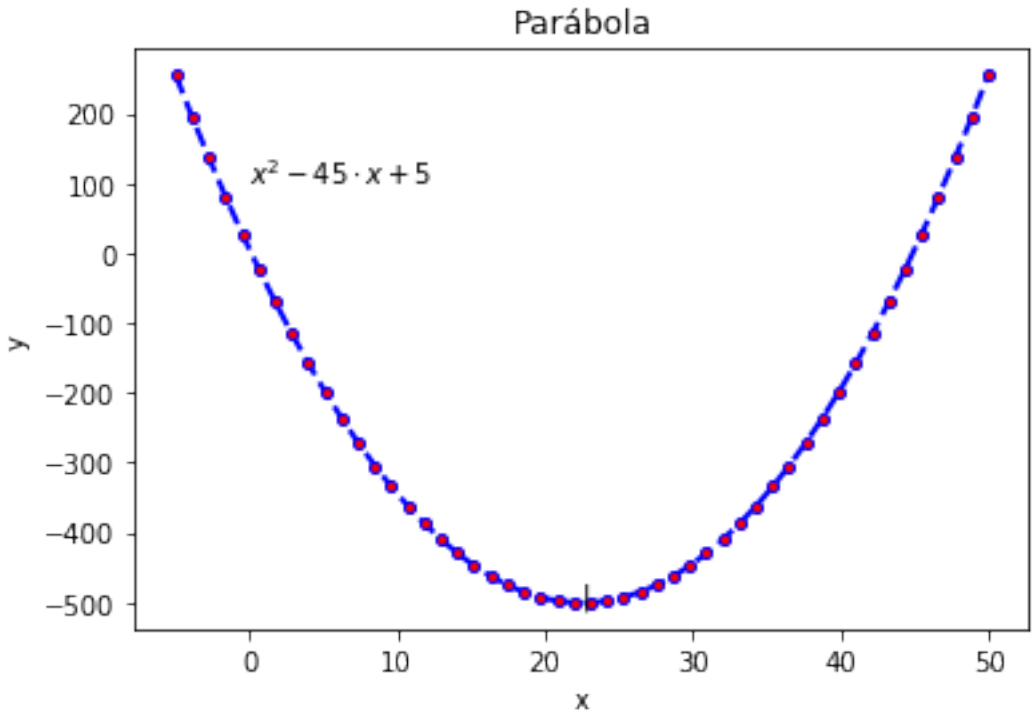
- color (ou c) fornece a cor da linha e dos marcadores. Aceita:
  - Valores hexadecimais, como #00CC00 para um verde gritante.
  - Inicial de uma das cores básicas da paleta:
    * b: blue
    * g: green
    * r: red
    * c: cyan
    * m: magenta
    * y: yellow
    * k: black (key)
    * w: white
  - Cores de html
    * lime
    * aquamarine
    * darkblue
    * ...
  - C{num}, como C0, C1, etc. Informa a cor do ciclo de cor atual. Nas figuras mostradas, o padrão é uma linha azul, então C0 é aquele tom de azul. Mais informações
- marker fornece o estilo do marcador. Aqui há uma lista dos marcadores disponíveis. Alguns exemplos:
  - o: bola
  - s: quadrado
  - v, <, ^, >: triângulos
  - 8: octágono
  - p: pentágono
  - *: estrela
  - d: losango
- markersize: assim como linewidth, fornece o tamanho dos marcadores, em pontos.
- markerfacecolor (ou mfc): fornece a cor para o interior dos marcadores. Aceita uma das cores de *color*. Note que o interior ficou vermelho, mas a borda não, pois ela foi definida como azul anteriormente. Caso você queira trocar a cor da borda, utilize markeredgecolor (ou mec). Se quiser remover a borda, coloque markeredgewidth=0.

É possível utilizar um dicionário para fornecer esses parâmetros, assim definindo um estilo "padrão"para os gráficos que você irá criar. Veja o mesmo exemplo anterior, utilizando um dicionário e um detalhe de sintaxe utilizando **.

```
In [6]: estilo = {'linewidth':2, 'linestyle':'--', 'color':'b', 'marker':'o', 'markersize':4, 'm
        plt.plot(x, y, **estilo)
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Parábola')
```

```
        ymin = y.min()
        # Como x e y tem o mesmo comprimento, o valor da posição do mínimo em y
        # é o mesmo da posição de mínimo em x
        xmin = x[y.argmin()]

        plt.text(xmin, ymin, '|', horizontalalignment='right')
        plt.text(0, 100, r'$x^2 - 45 \cdot x + 5$')  # Escolhidos a olho
```

```
Out[6]: Text(0,100,'$x^2 - 45 \\cdot x + 5$')
```



A sintaxe de **dict significa que os itens do dicionário são separados como se fossem argumentos opcionais de uma função. Então, **estilo basicamente se transforma em: 'linewidth'=2, 'linestyle'='--', ...

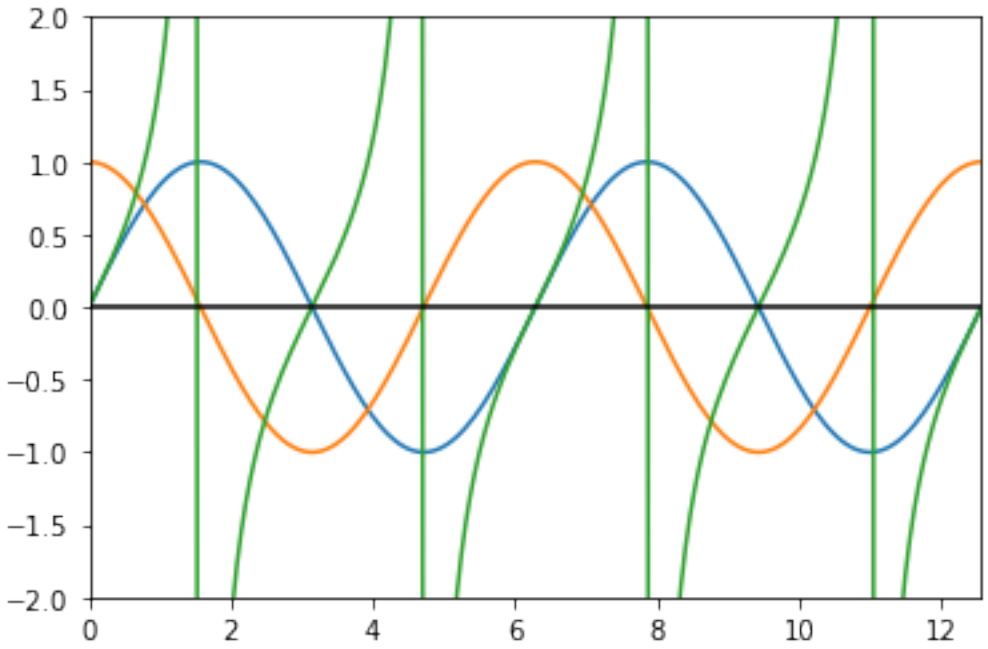
2.4 Colocando mais linhas

Para criar mais linhas num gráfico, é necessário somente colocar mais comandos plot.

```
In [7]: x = np.linspace(0, 4 * np.pi, 200)
        y1 = np.sin(x)
        y2 = np.cos(x)
        y3 = np.tan(x)

        plt.plot(x, y1)
```

```
plt.plot(x, y2)
plt.plot(x, y3)
plt.ylim((-2, 2)) # ylim aceita uma lista de (min, max), logo os (())
plt.xlim((0, 4 * np.pi))
plt.axhline(0, color='k') # cria uma linha horizontal no valor de y fornecido.
```

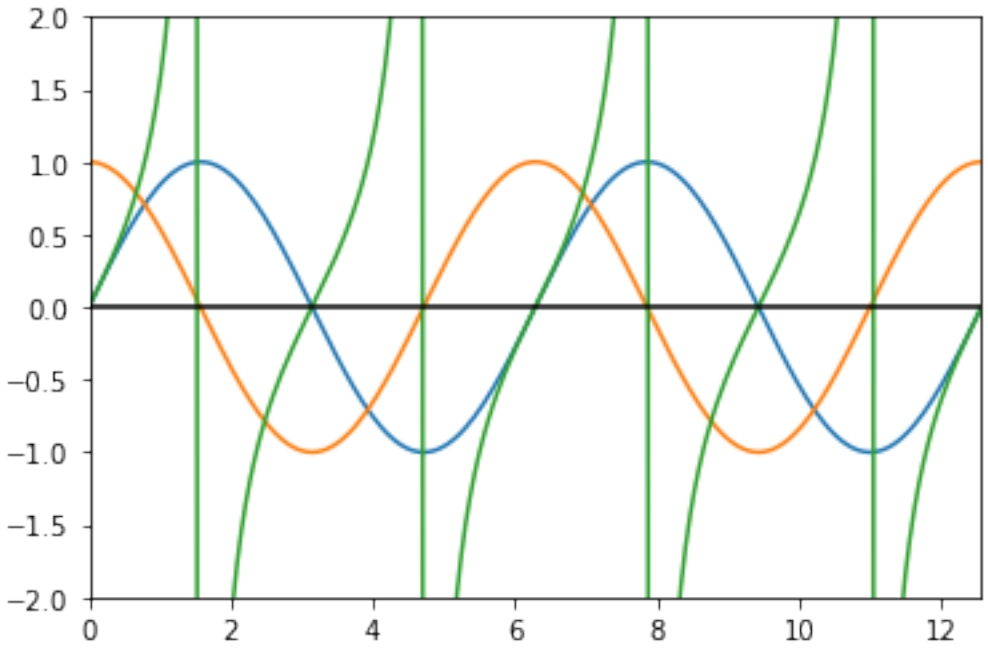
Out[7]: <matplotlib.lines.Line2D at 0x7f82229f8860>



In [8]: # Maneira alternativa, utilizando outro tipo de sintaxe do pyplot

```
plt.plot(x, y1, x, y2, x, y3)
plt.ylim((-2, 2))
plt.xlim((0, 4 * np.pi))
plt.axhline(0, color='k')
```

Out[8]: <matplotlib.lines.Line2D at 0x7f8222af4a90>

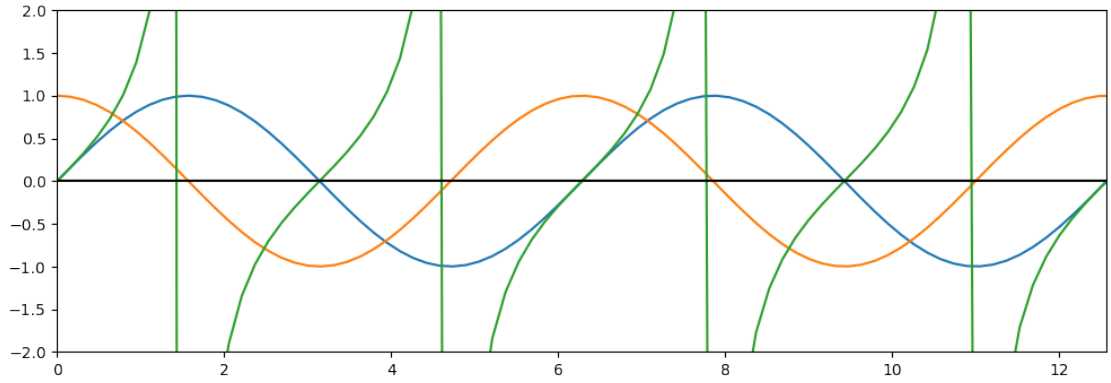


2.5 Alterando a figura

Vamos extender o plot das funções trigonométricas para ver sua periodicidade. Para isso, devemos alterar as propriedades da figura atual. Isso é feito com o comando *figure*. Esse comando cria uma nova figura. Uma figura possui um número, que é incrementado em relação ao anterior, caso nada seja fornecido. Algumas propriedades de figure:

- figsize: estabelece o tamanho da figura, em polegadas. (comprimento,altura)
- dpi: resolução (dots per inch). Utilize o tamanho da figura e a resolução dpi para ajustar a resolução final (p.e. 1920x1080).

```
In [9]: x = np.linspace(0, 10 * np.pi, 200)
        y1 = np.sin(x)
        y2 = np.cos(x)
        y3 = np.tan(x)


        plt.figure(figsize=(12,4), dpi=100)
        plt.plot(x, y1, x, y2, x, y3)
        plt.ylim((-2, 2))
        plt.xlim((0, 4 * np.pi))
        plt.axhline(0, color='k')

Out[9]: <matplotlib.lines.Line2D at 0x7f822298fa20>
```



2.6 Alterando o separador de decimal

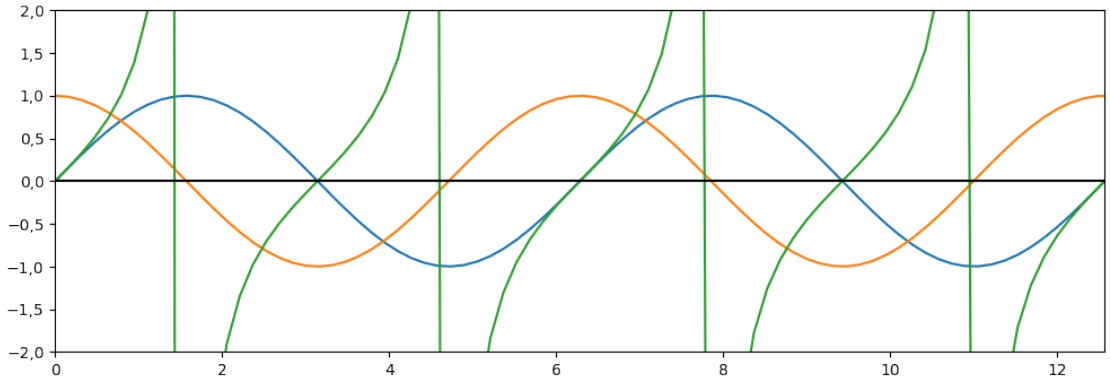
Infelizmente, essa é uma tarefa muito pouco trivial no matplotlib, especialmente porque, como brasileiros, a vírgula deve ser o separador de decimal.

O matplotlib consegue detectar o seu local atual. Local é uma variável do ambiente do seu sistema operacional que lida com características locais dos países e línguas. Por exemplo, nos EUA, o local informaria que a unidade de temperatura é Fahrenheit, qual o tipo de teclado e que o separador decimal é um ponto. No Brasil, temos Celsius e vírgula.

Para alterar isso, devemos importar outro tipo de módulo e depois mandar o matplotlib reconhecer o local, alterando a variável global **rcParams**. Mais informações sobre essa variável podem ser encontradas aqui. É possível alterar o arquivo padrão para que o matplotlib sempre comece com o local certo.

Outra maneira é utilizando um *FuncFormatter* e um *Ticker* mas, sinceramente, são métodos tão mais complicados que não vale a pena mostrar aqui.

```
In [10]: import locale
         locale.setlocale(locale.LC_ALL, '') # Permite o Python determinar o Local atual

         import matplotlib as mpl
         mpl.rcParams['axes.formatter.use_locale'] = True # Faz o MPL utilizar o local atual

         plt.figure(figsize=(12,4), dpi=100)
         plt.plot(x, y1, x, y2, x, y3)
         plt.ylim((-2, 2))
         plt.xlim((0, 4 * np.pi))
         plt.axhline(0, color='k')

Out[10]: <matplotlib.lines.Line2D at 0x7f822293b940>
```

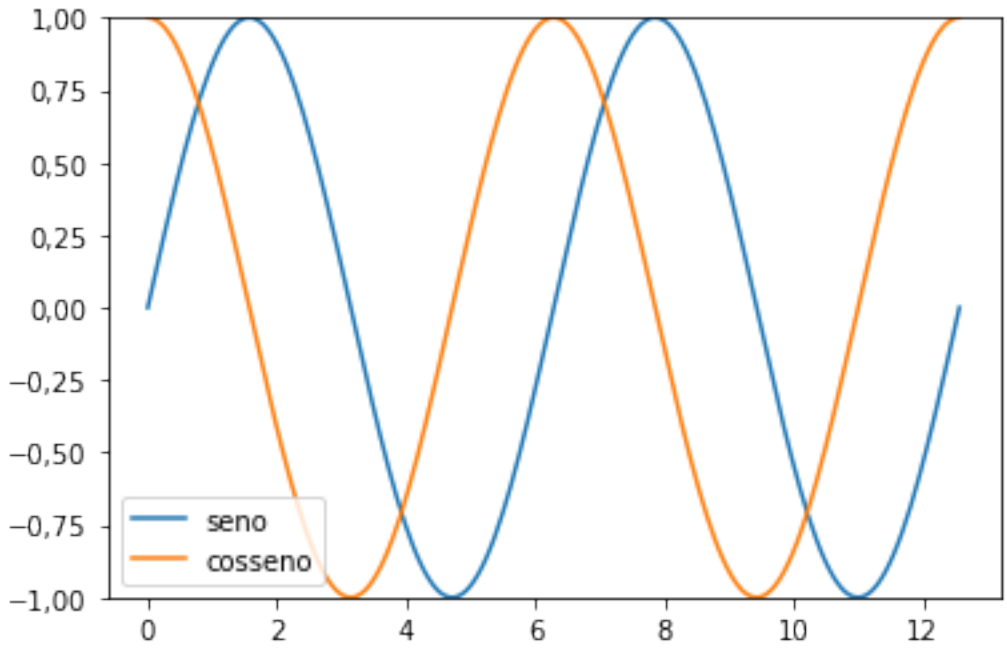


2.7 Colocando legendas nas figuras

Para criar uma legenda, é necessário atribuir **labels** para os plots, e depois chamar a função **legend** para criar e colocar a legenda.

```
In [11]: x = np.linspace(0, 4 * np.pi, 200)
         y1 = np.sin(x)
         y2 = np.cos(x)

         plt.plot(x, y1, label='seno')
         plt.plot(x, y2, label='cosseno')
         plt.legend()
         plt.ylim((-1, 1))
```

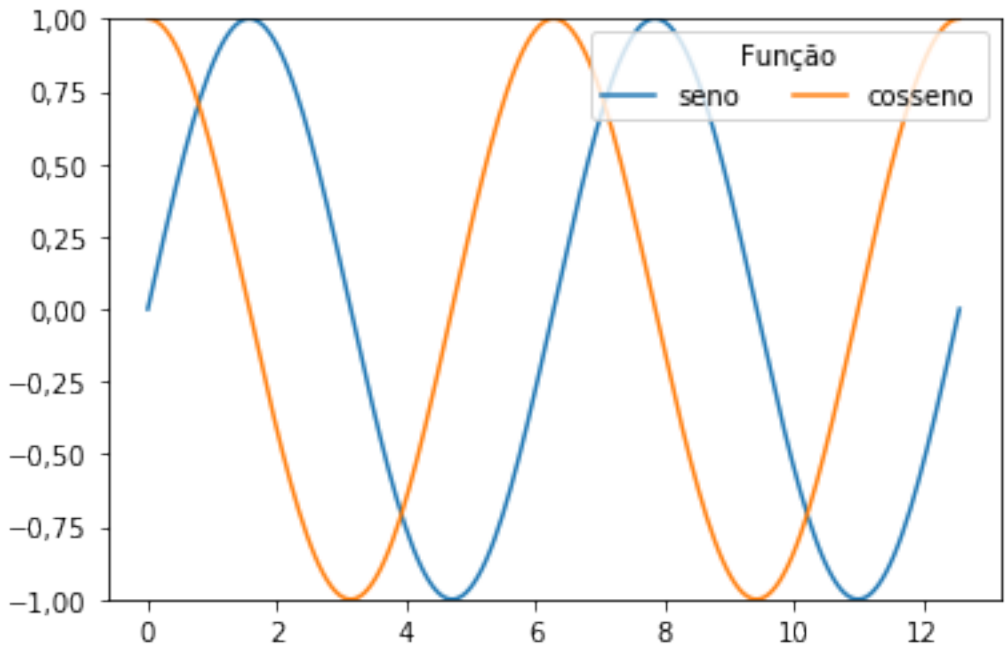
```
Out[11]: (-1, 1)
```



Por padrão, a legenda é colocada na "melhor"região do gráfico, isso é, onde há menor sobreposição da legenda com as linhas na figura. É possível alterar isso com o argumento **loc**. Além disso, podemos alterar o número de colunas na legenda, o título dela e, inclusive, associar símbolos específicos com nomes específicos, mas isso requer um pouco mais de trabalho, e é fora do escopo deste curso.

```
In [12]: x = np.linspace(0, 4 * np.pi, 200)
         y1 = np.sin(x)
         y2 = np.cos(x)

         plt.plot(x, y1, label='seno')
         plt.plot(x, y2, label='cosseno')
         plt.legend(title='Função', ncol=2, loc='upper right')
         plt.ylim((-1, 1))
```

```
Out[12]: (-1, 1)
```



2.8 Salvando figuras

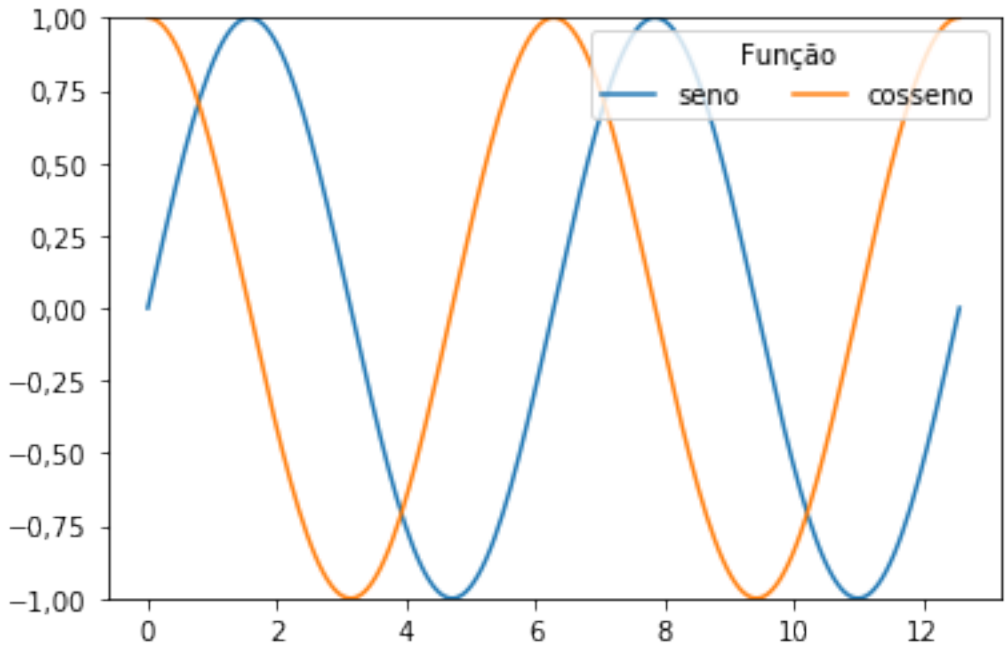
Não pode faltar, claro, a função de salvar uma figura. A função **savefig** faz justamente isso. O único parâmetro obrigatório é o nome/caminho da figura. A extensão da figura de destino pode ser definida tanto no nome quando como pelo argumento *format*, e os valores possíveis são:

- png

- pdf
- ps
- eps
- svg

É possível também alterar a resolução da figura pelo argumento *dpi*. Senão, é utilizado o valor da figura criada que, se não foi definido, é o valor padrão.

```
In [13]: x = np.linspace(0, 4 * np.pi, 200)
         y1 = np.sin(x)
         y2 = np.cos(x)

         plt.plot(x, y1, label='seno')
         plt.plot(x, y2, label='cosseno')
         plt.legend(title='Função', ncol=2, loc='upper right')
         plt.ylim((-1, 1))

         plt.savefig('seno_cosseno.png', dpi=150)
```



3 Método implícito e método explícito

Até agora temos utilizado o método implícito para criar figuras e plotar gráficos. Esse método supõe que as alterações estão sendo feitas em um eixo ativo específico. Então, para mudar de eixos, é necessário alterar o eixo ativo. Isso pode ser um pouco confuso. Por isso foi desenvolvido o método explícito, que utiliza os princípios da programação orientada a objetos para tornar o código mais claro. Além disso, é a maneira recomendada para criar gráficos mais complexos.

3.1 Criação de uma figura com múltiplos gráficos (eixos)

Para isso, criamos um objeto figura e um conjunto de objetos eixos (*Axes*), que contém um eixo cada (*Axis*). Nessa criação, podemos informar quantas linhas e quantas colunas de eixos utilizaremos. Então, cada eixo possui as mesmas funções de plot. De fato, chamar *plt.plot* significa, por trás das cortinas, chamar a função plot no eixo atual. Vamos mostrar aqui como criar um plot com as três funções trigonométricas, cada uma em seu próprio gráfico, lado a lado em uma linha, da maneira explícita e implícita.

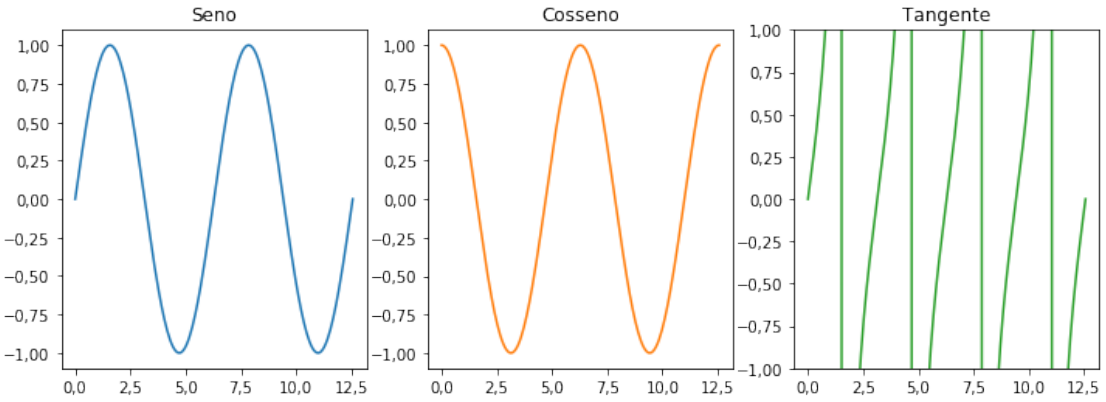
3.1.1 Implícita

Utiliza-se o comando subplot para especificar qual é o subplot a ser ativado. A sintaxe é

```
subplot(nrows, ncols, index)
```

Ou seja, para criar uma figura com 1 linha e 3 colunas, devemos utilizar subplot(1, 3, x). Aí, x é alterado para um valor entre 1, 2 ou 3, dependendo do subplot específico que desejamos ativar. Essa sintaxe é legado do Matlab. É possível também abreviar essa funçã, tirando as vírgulas, logo um *subplot(1,3,1)* vira *subplot(131)*

```
In [14]: x = np.linspace(0, 4 * np.pi, 200)
         y1 = np.sin(x)
         y2 = np.cos(x)
         y3 = np.tan(x)

         plt.figure(figsize=(12,4))

         plt.subplot(131)
         plt.plot(x, y1, c='C0')
         plt.title('Seno')

         plt.subplot(132)
         plt.plot(x, y2, c='C1')
         plt.title('Cosseno')

         plt.subplot(133)
         plt.plot(x, y3, c='C2')
         plt.ylim((-1, 1))
         plt.title('Tangente')

Out[14]: Text(0.5,1,'Tangente')
```



3.1.2 Explícita

Para a criação de algo explícito, chamamos a função subplots. Ela requer o número de linhas e colunas, e também aceita parâmetros da figura. Essa função retorna um objeto figura e uma lista com objetos axis, cuja conformação é dependente do número de linhas e colunas. Por exemplo, para reproduzir a figura acima, teremos uma lista de eixos assim:

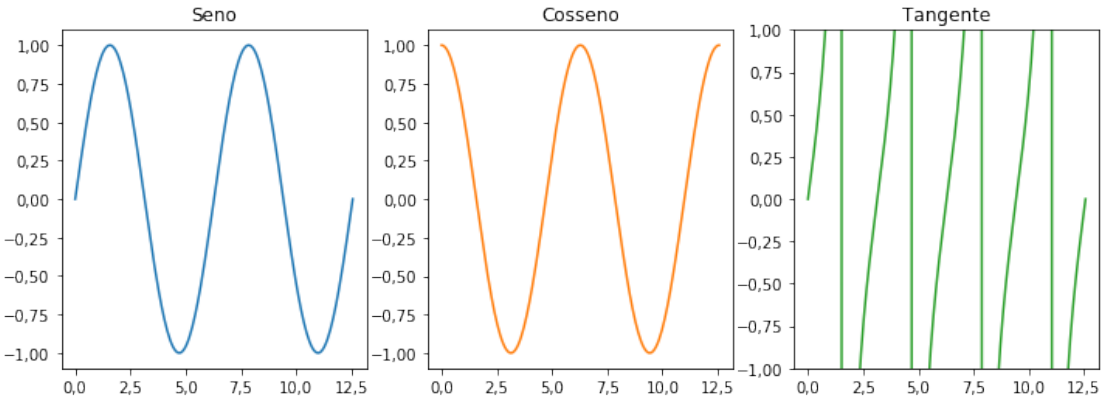
```
[eixo0, eixo1, eixo2]
```

Caso desejemos fazer uma figura com 2 linhas e 2 colunas, teremos uma lista assim:

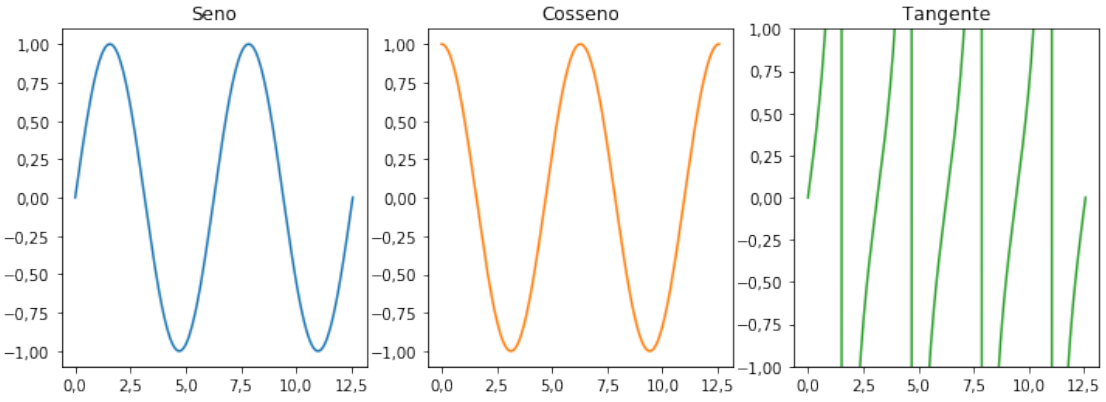
```
[[eixo00, eixo01],
 [eixo10, eixo11]]
```

É possível utilizar os princípios de *tuple assignment*, extendidos para um *list assignment*, para nomear diretamente os eixos. Mostraremos os dois métodos.

```
In [15]: fig, axes = plt.subplots(nrows = 1, ncols=3, figsize=(12,4))
         axes[0].plot(x, y1, c='C0')
         axes[1].plot(x, y2, c='C1')
         axes[2].plot(x, y3, c='C2')

         axes[0].set_title('Seno')
         axes[1].set_title('Cosseno')
         axes[2].set_title('Tangente')

         axes[2].set_ylim((-1, 1))

Out[15]: (-1, 1)
```




```
In [16]: fig, [seno, cosseno, tg] = plt.subplots(nrows = 1, ncols=3, figsize=(12,4))
         seno.plot(x, y1, c='C0')
         cosseno.plot(x, y2, c='C1')
         tg.plot(x, y3, c='C2')

         seno.set_title('Seno')
         cosseno.set_title('Cosseno')
         tg.set_title('Tangente')

         tg.set_ylim((-1, 1))

Out[16]: (-1, 1)
```



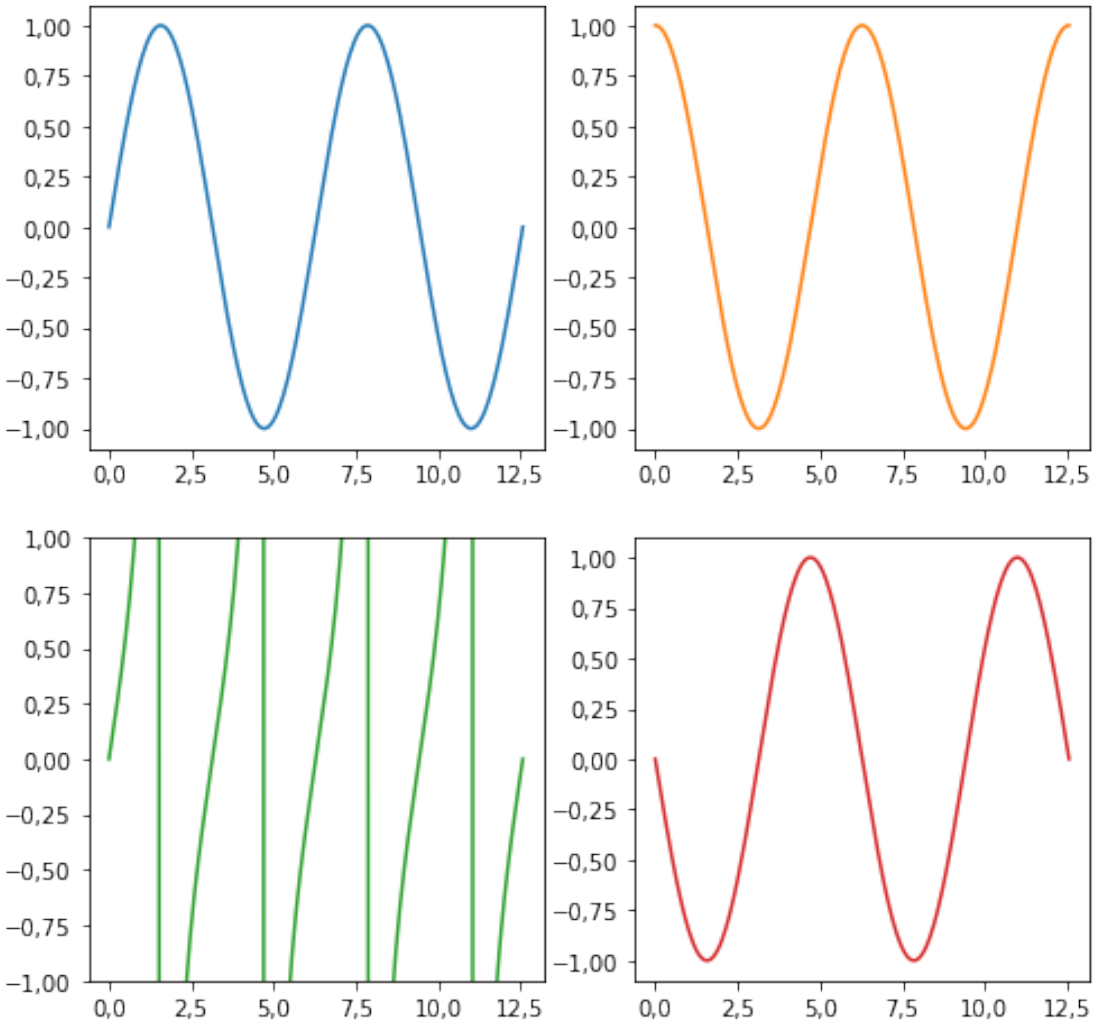
Note que no método implícito, foi necessário colocar todas as operações em cada eixo de uma vez só. Na opção explícita, foi possível agrupar as operações de plot juntas, e colocar os títulos em outro conjunto de código. No final das contas, é uma questão da complexidade do gráfico que você deseja fazer, e de preferência pessoal.

De agora em diante, será utilizado o método explícito.

Como exemplo, vamos mostrar como realizar um plot de 2 linhas e 2 colunas.

```
In [17]: x = np.linspace(0, 4 * np.pi, 200)
         y1 = np.sin(x)
         y2 = np.cos(x)
         y3 = np.tan(x)
         y4 = np.cos(x + np.pi/2)

         fig, axes = plt.subplots(nrows = 2, ncols=2, figsize=(8,8))

         axes[0, 0].plot(x, y1, c='C0')
         axes[0, 1].plot(x, y2, c='C1')
         axes[1, 0].plot(x, y3, c='C2')
         axes[1, 1].plot(x, y4, c='C3')

         axes[1, 0].set_ylim((-1, 1))
```

```
Out[17]: (-1, 1)
```



4 Recursos mais avançados

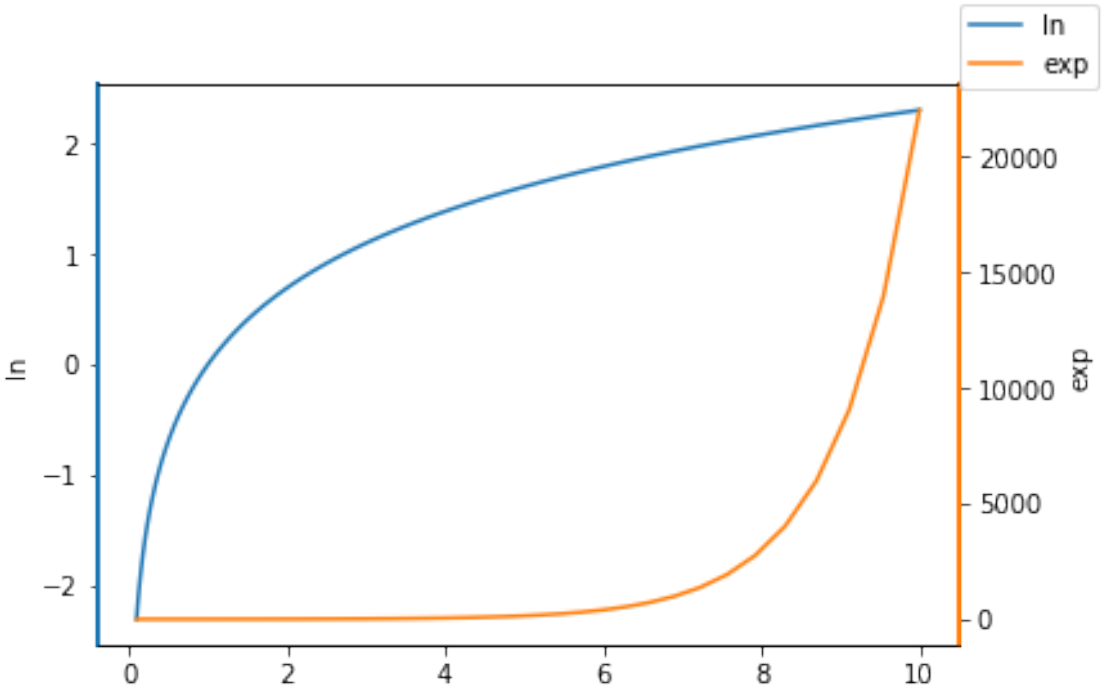
4.1 Dois eixos y para um eixo x

Para isso, utilizamos a função **twinx** sobre um outro eixo. Isso cria um novo conjunto de eixos imediatamente acima do primeiro, porém somente com valores para os *ticks* no eixo y do lado direito. Isso geralmente é necessário quando temos dois gráficos com valores muito discrepantes de y, e queremos compará-los.

Neste exemplo, serão comparadas as funções log e exponencial. Além disso, utilizaremos a função *legend* na figura, ao invés do eixo. Isso cria uma legenda utilizando todas as linhas em uma figura, e sua posição é refente à figura, não a um eixo. Caso a função legend seja chamada sobre um eixo, aparecerá somente a legenda da única linha naquele eixo.

Alteraremos também as cores dos eixos. Note que é necessário alterar a cor dos eixos do segundo eixo, pois ele está sobreposto ao primeiro. Caso isso não seja feito, haverá uma linha preta correndo no meio.

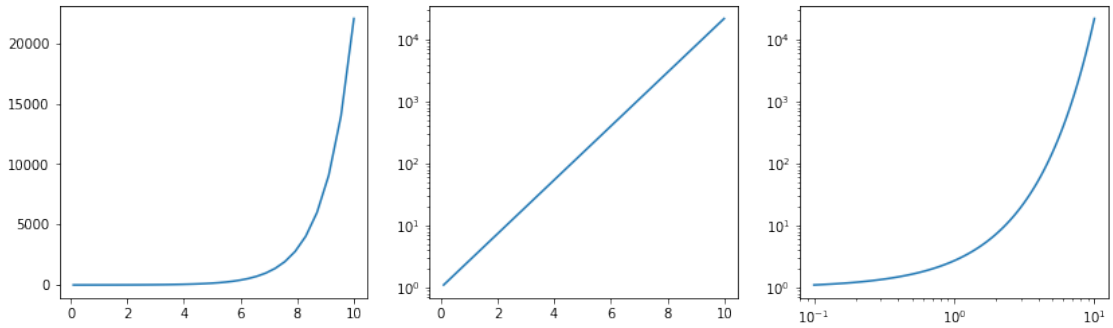
```
In [18]: x = np.logspace(-1, 1, 100)
         y1 = np.log(x)
         y2 = np.exp(x)

         fig, ax = plt.subplots(1, 1)
         ax2 = ax.twinx()

         ax.plot(x, y1, label='ln', c='C0')
         ax2.plot(x, y2, label='exp', c='C1')

         ax.set_ylabel('ln')
         ax2.set_ylabel('exp')

         ax2.spines['left'].set_color('C0')
         ax2.spines['left'].set_linewidth(2)
         ax2.spines['right'].set_color('C1')
         ax2.spines['right'].set_linewidth(2)

         fig.legend()

Out[18]: <matplotlib.legend.Legend at 0x7f8220a67dd8>
```



4.2 Alterando a escala de um eixo

Vários tipos de dados são melhor visualizados na escala logarítmica. O campo da reologia, por exemplo, utiliza muitos eixos logaritmicos. Vamos comparar aqui o que acontece quando essa transformação é aplicada criando um gráfico com dois eixos, um semilog em x e o outro não.

```
In [19]: x = np.logspace(-1, 1, 100)
         y1 = np.log(x)
         y2 = np.exp(x)

         fig, axes = plt.subplots(1, 3, figsize=(14, 4))

         axes[0].plot(x, y2)
         axes[1].plot(x, y2)
         axes[2].plot(x, y2)

         #axes[1].set_xscale('log')
         axes[1].set_yscale('log')
         axes[2].set_yscale('log')
         axes[2].set_xscale('log')
```

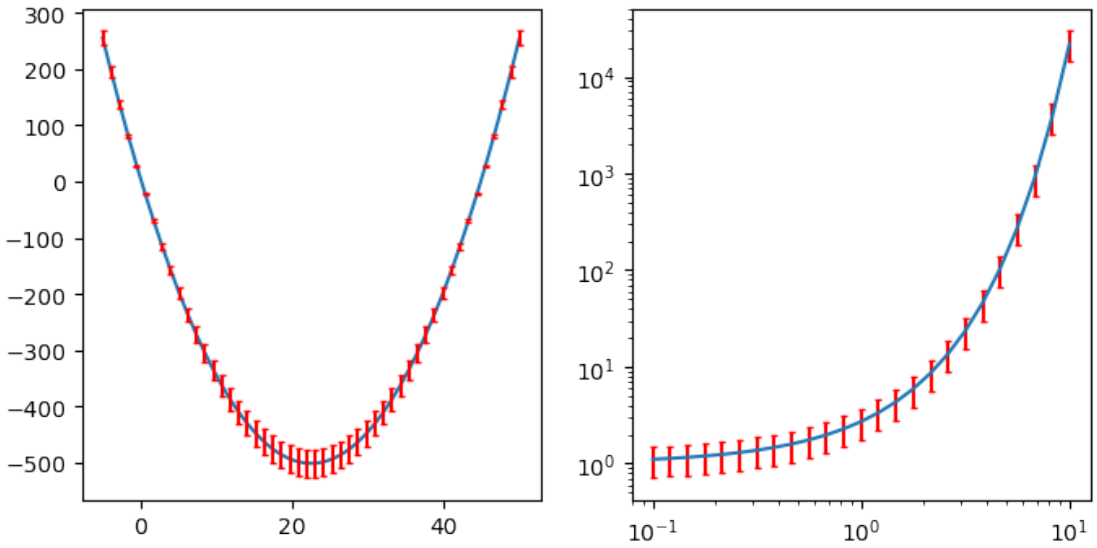


Vemos então que uma curva exponencial se torna uma reta. No gráfico duplo log, o formato da curva se manteve bastante, mas não totalmente.

4.3 Barras de erro

Há várias funções além de plot disponíveis no matplotlib. Por exemplo, há bar, histogram, boxplot. Porém, esses outros tipos não serão mostrados aqui, com exceção do errorbar. Essa função pode receber valores de erro em x e y, que podem ser constantes ou listas.

```
In [20]: x1 = np.linspace(-5, 50)
         y1 = x1 ** 2 - 45 * x1 + 5
         y1_err = 0.05 * y1  # 5% do ponto de erro

         x2 = np.logspace(-1, 1, 25)
         y2 = np.exp(x2)
         y2_err = 0.35 * y2

         fig, [par, exp] = plt.subplots(1, 2, figsize=(8, 4), dpi=100)

         par.errorbar(x1, y1, yerr=y1_err, capsize=1.5, ecolor='r', barsabove=True)
         exp.errorbar(x2, y2, yerr=y2_err, capsize=1.5, ecolor='r')
         exp.set_xscale('log')
         exp.set_yscale('log')
```



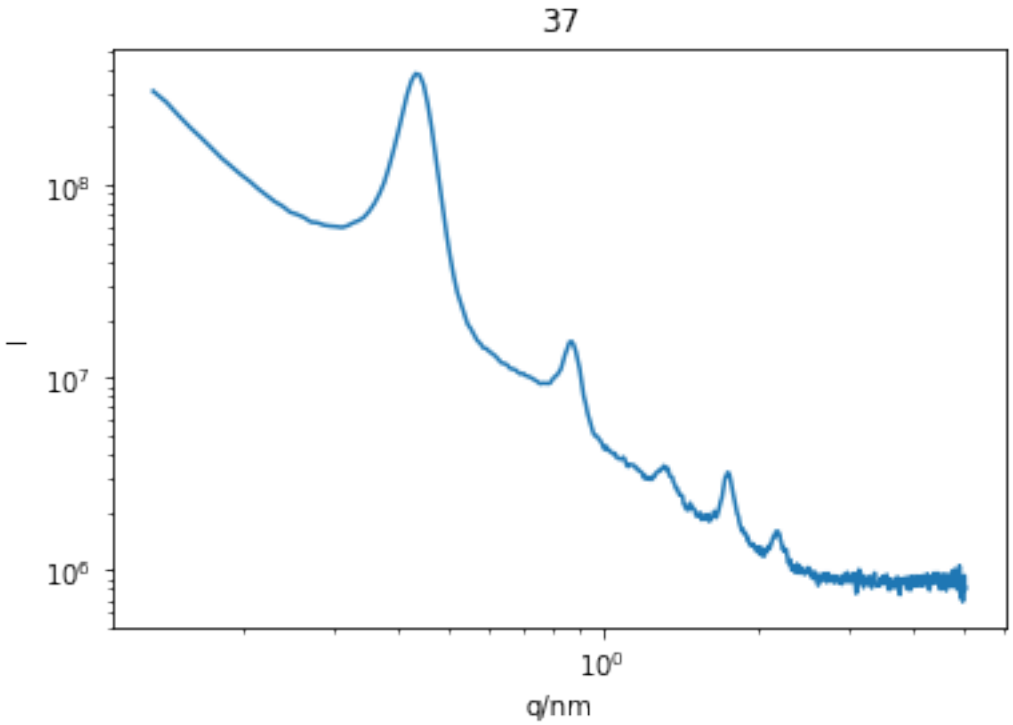
Podemos observar duas coisas interessantes.

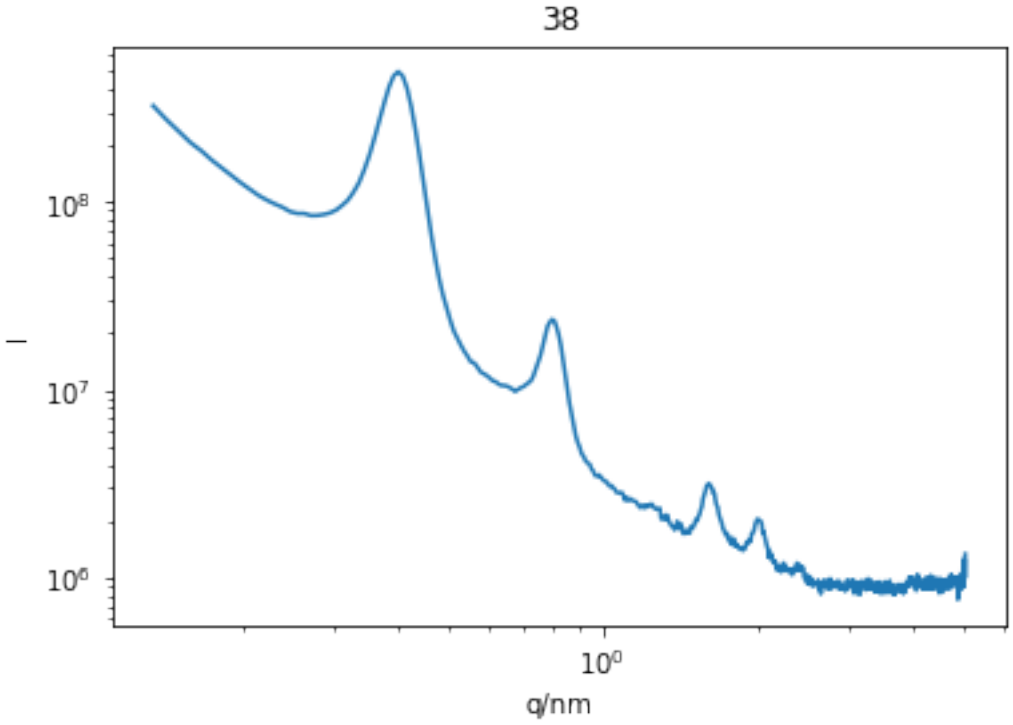
1. Por padrão, as barras de erro aparecem abaixo da linha, possuem a mesma cor da linha e não há uma barra horizontal nas pontas. Todos esses parâmetros podem ser alterados.
2. As barras de erro do gráfico log parecem estar assimétricas. Isso é devido justamente ao espaçamento em y não ser constante.

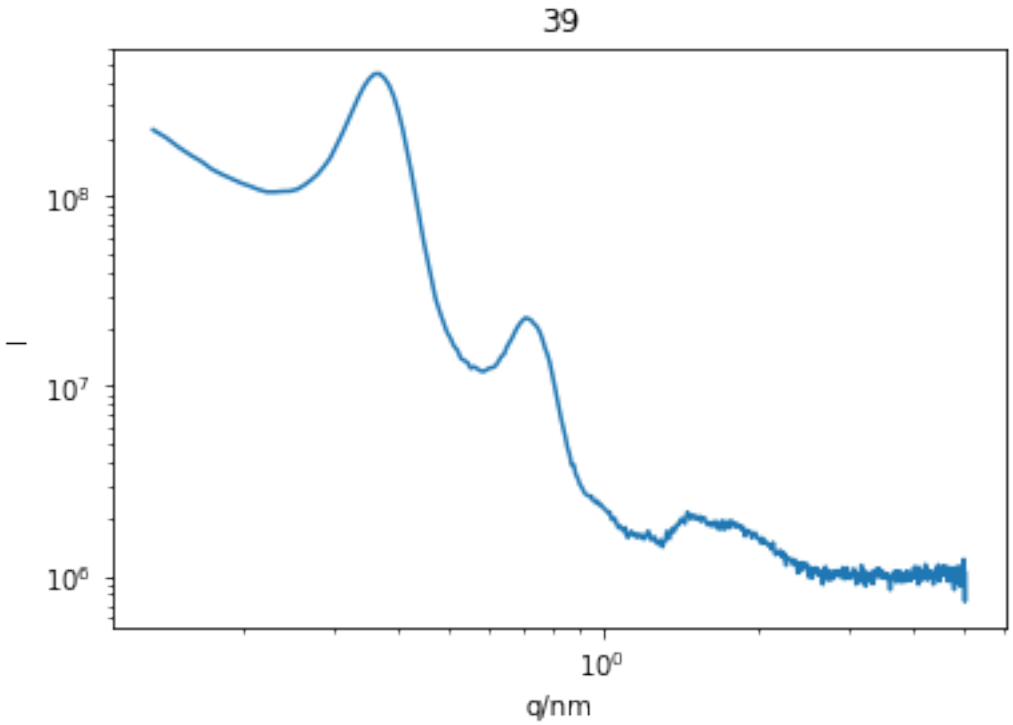
5 Utilizando pandas e pyplot juntos

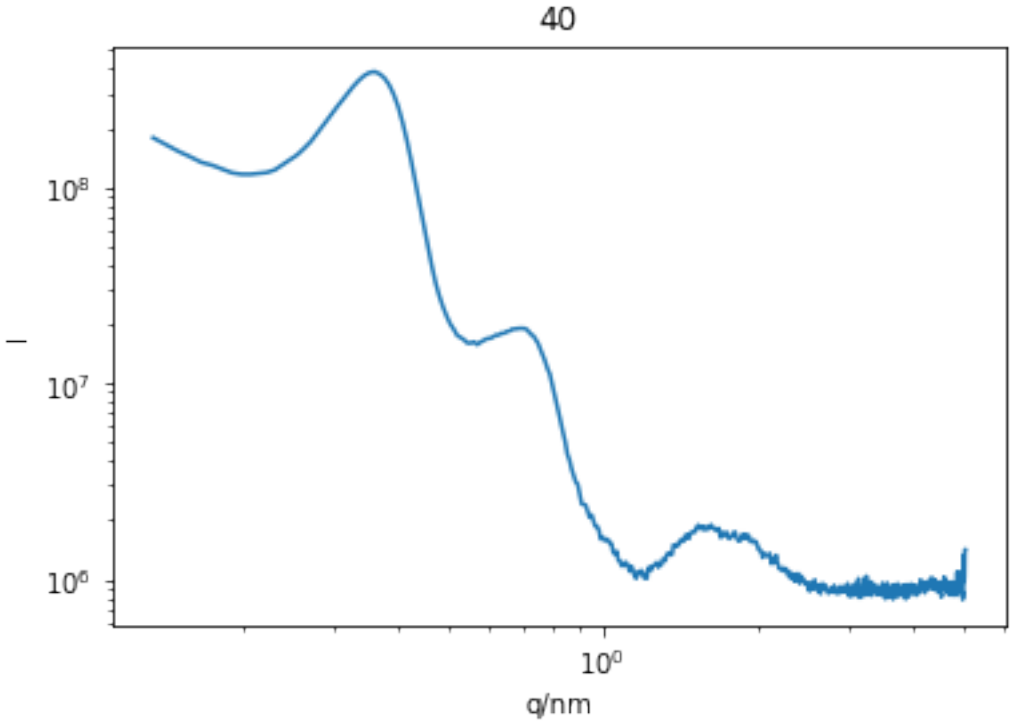
Agora juntaremos todos os conhecimentos de todas as aulas anteriores para utilizar duas ferramentas bastante poderosas para tratamento e visualização de dados. Utilizaremos o pandas para abrir os dados da pasta dados-1 e depois plotá-los. Os dados são de SAXS, então geralmente são plotados na escala log. Além disso, utilizaremos um loop para plotar todos os dados em figuras diferentes e na mesma figura, com e sem stacking.

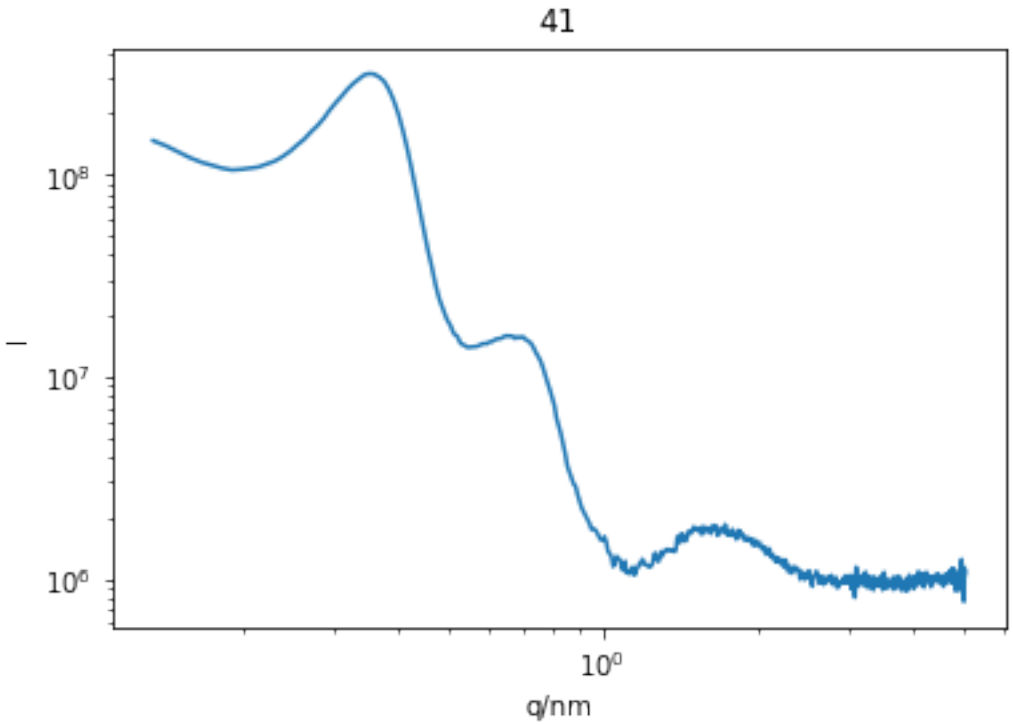
```
In [21]: import glob
         import pandas as pd
         import os
         os.chdir('./dados-1')

In [22]: # Figuras diferentes

         saxs = glob.glob('*.dat')
         saxs.sort() # Vai do menor para o maior

         for dado in saxs:
             df = pd.read_csv(dado, sep='  ', engine='python', names=['q', 'I'])
             fig, ax = plt.subplots(1, 1)
             ax.plot(df['q'], df['I'])
             ax.set(xscale='log', yscale='log', title=dado[:-4], xlabel='q/nm', ylabel='I')
```

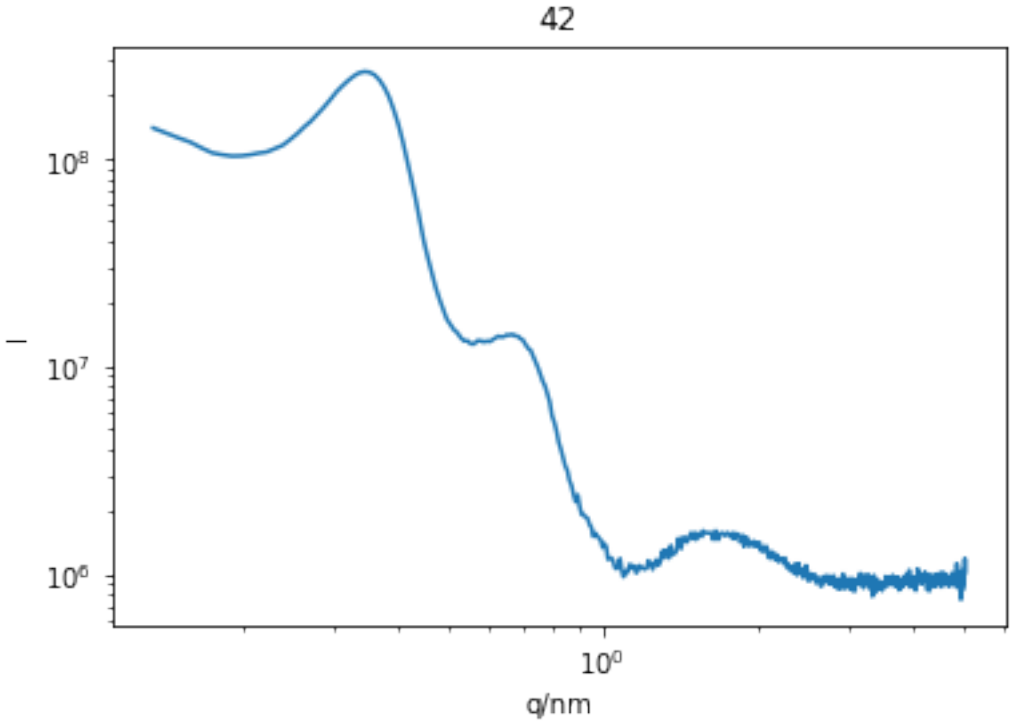

37
10^8
10^7
10^6
10^0
q/nm

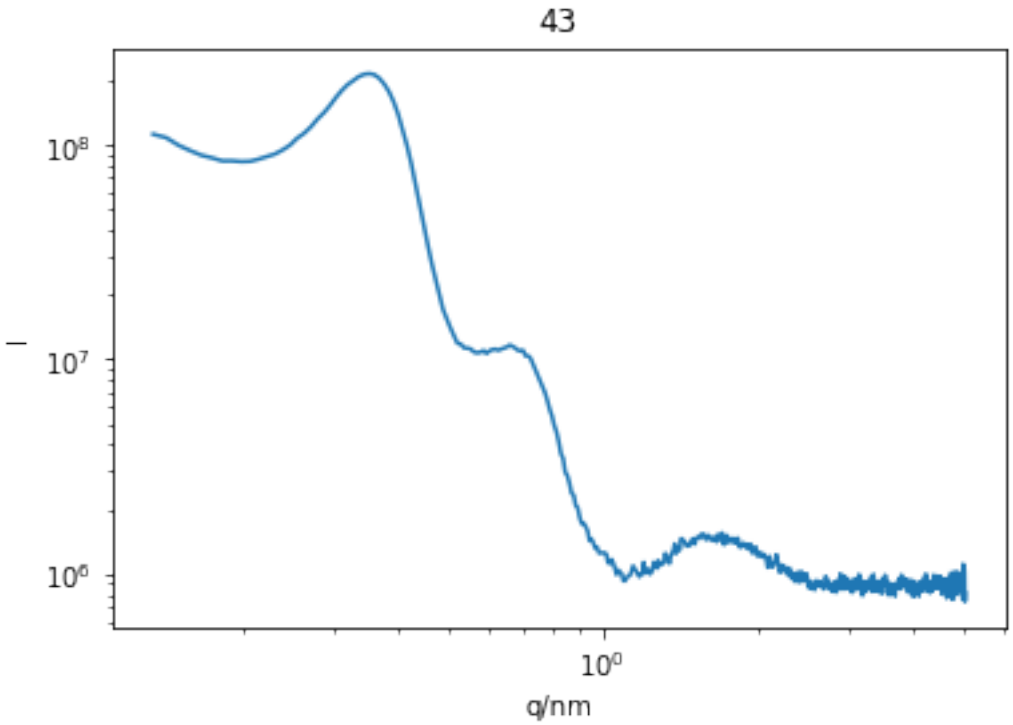

38
10^8
10^7
10^6
10^0
q/nm

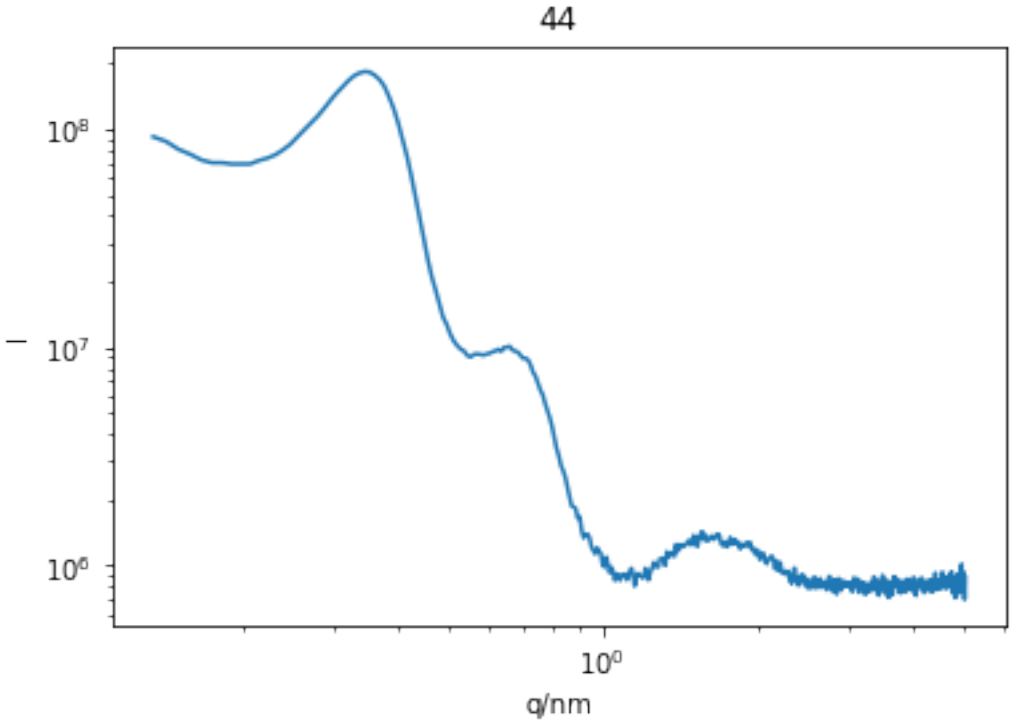

39
10^8
10^7
10^6
10^0
q/nm

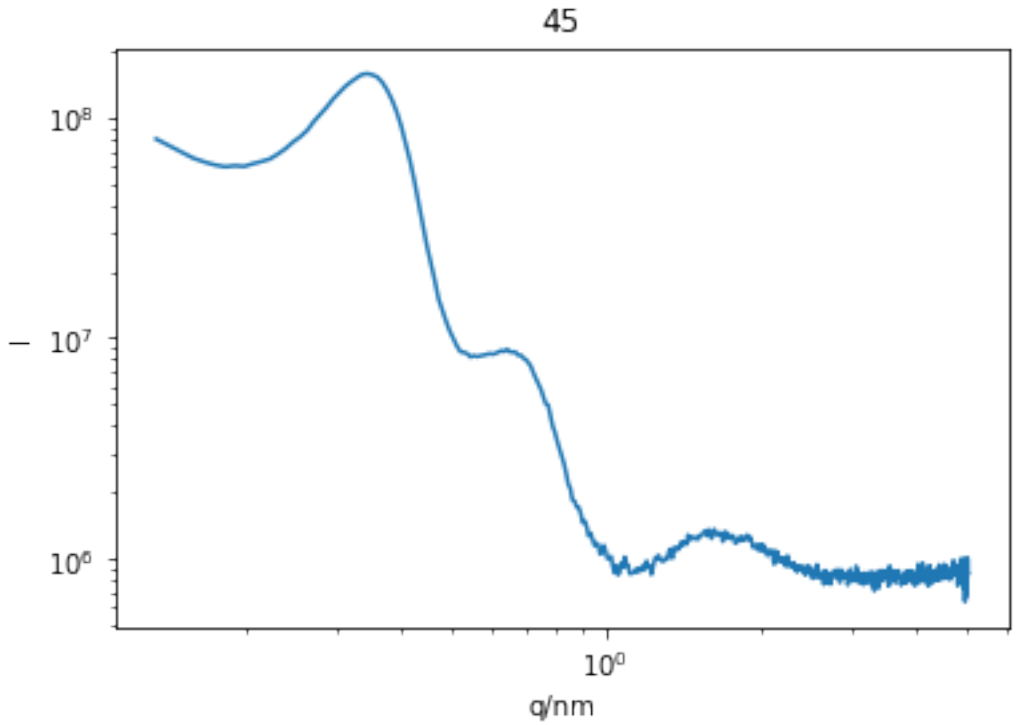

40
10^8
10^7
10^6
10^0
q/nm


41
10^8
10^7
10^6
10^0
q/nm


42
10^8
10^7
10^6
10^0
q/nm

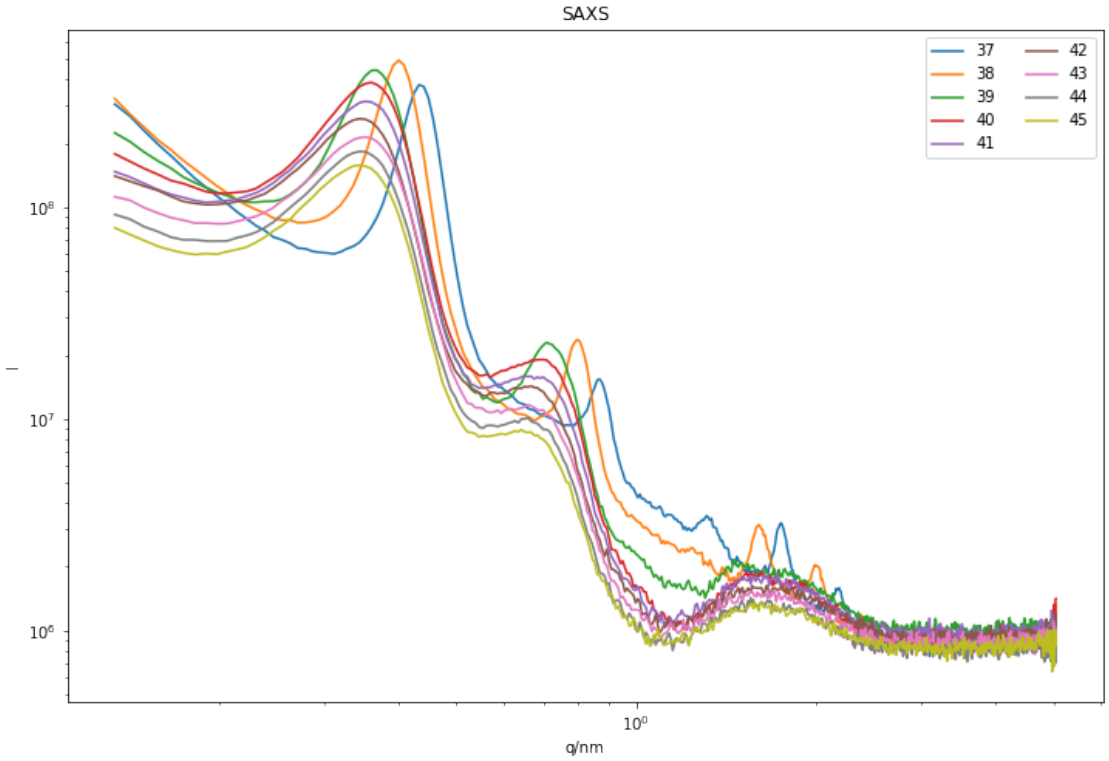

43
10^8
10^7
10^6
10^0
q/nm


44
10^8
10^7
10^6
10^0
q/nm

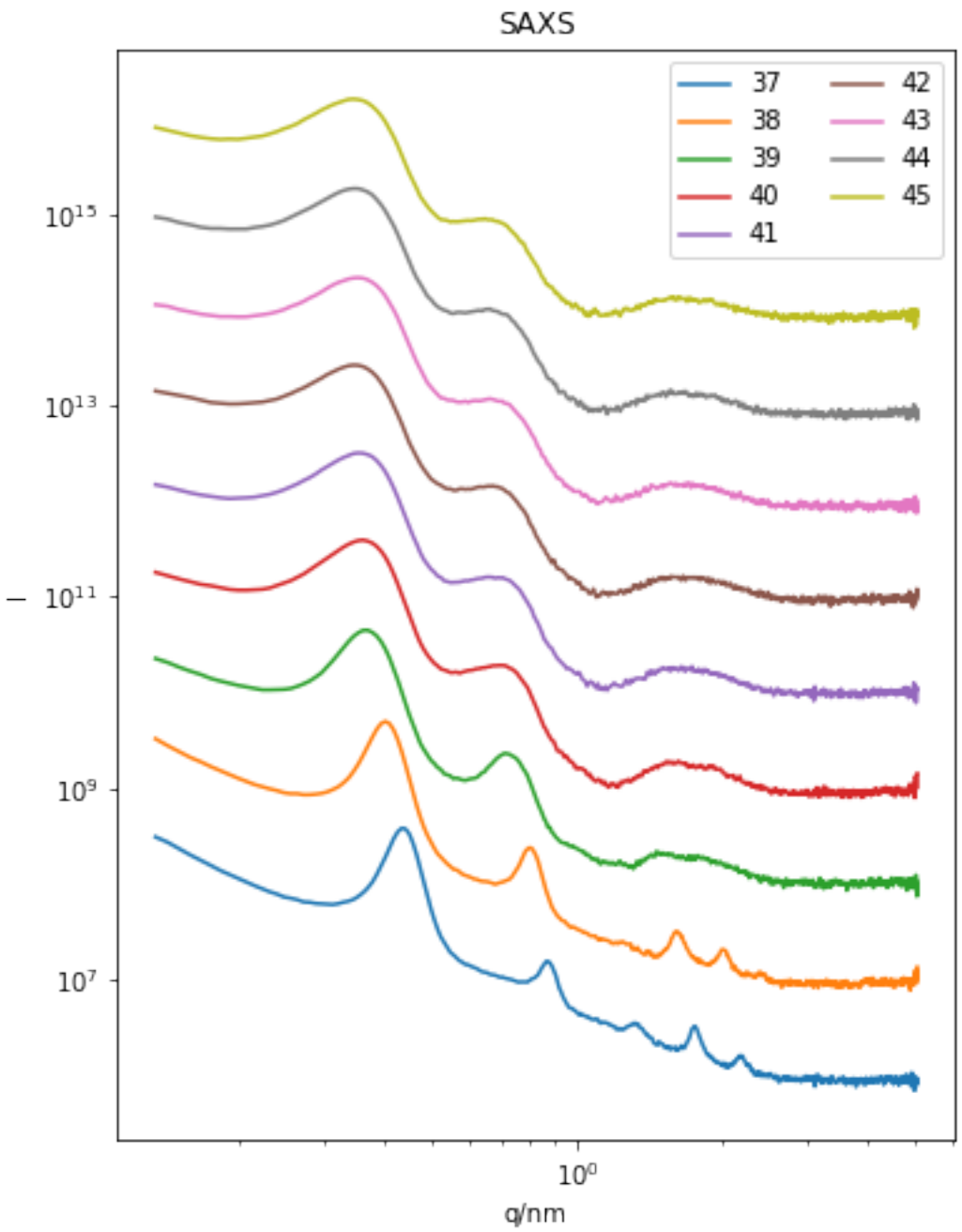


In [23]:
```
# Mesma figura

saxs = glob.glob('*.dat')
saxs.sort() # Vai do menor para o maior
fig, ax = plt.subplots(1, 1, figsize=(12,8))

for dado in saxs:
    df = pd.read_csv(dado, sep='  ', engine='python', names=['q', 'I'])
    ax.plot(df['q'], df['I'],  label=dado[:-4])

ax.set(xscale='log', yscale='log', title='SAXS', xlabel='q/nm', ylabel='I')
ax.legend(ncol=2)
```

Out[23]:
```
<matplotlib.legend.Legend at 0x7f821692e2e8>
```



In [27]:
```
# Mesma figura, com espaçamento entre os dados

saxs = glob.glob('*.dat')
saxs.sort() # Vai do menor para o maior
fig, ax = plt.subplots(1, 1, figsize=(6,8))

for i, dado in enumerate(saxs):
    df = pd.read_csv(dado, sep='  ', engine='python', names=['q', 'I'])
    ax.plot(df['q'], df['I'] * 10 ** i,  label=dado[:-4])

ax.set(xscale='log', yscale='log', title='SAXS', xlabel='q/nm', ylabel='I')
ax.legend(ncol=2)
```



Veja que a mudança entre plotar várias figuras separadas e plotar uma figura única com todos os dados é muito pequena. Essencialmente, a linha de criação da figura foi removida do loop. Como está fora, somente foi criada uma figura e um eixo. Dentro do loop, a cada iteração, uma nova figura e um novo eixo são criados, então aparecem várias figuras diferentes no final.

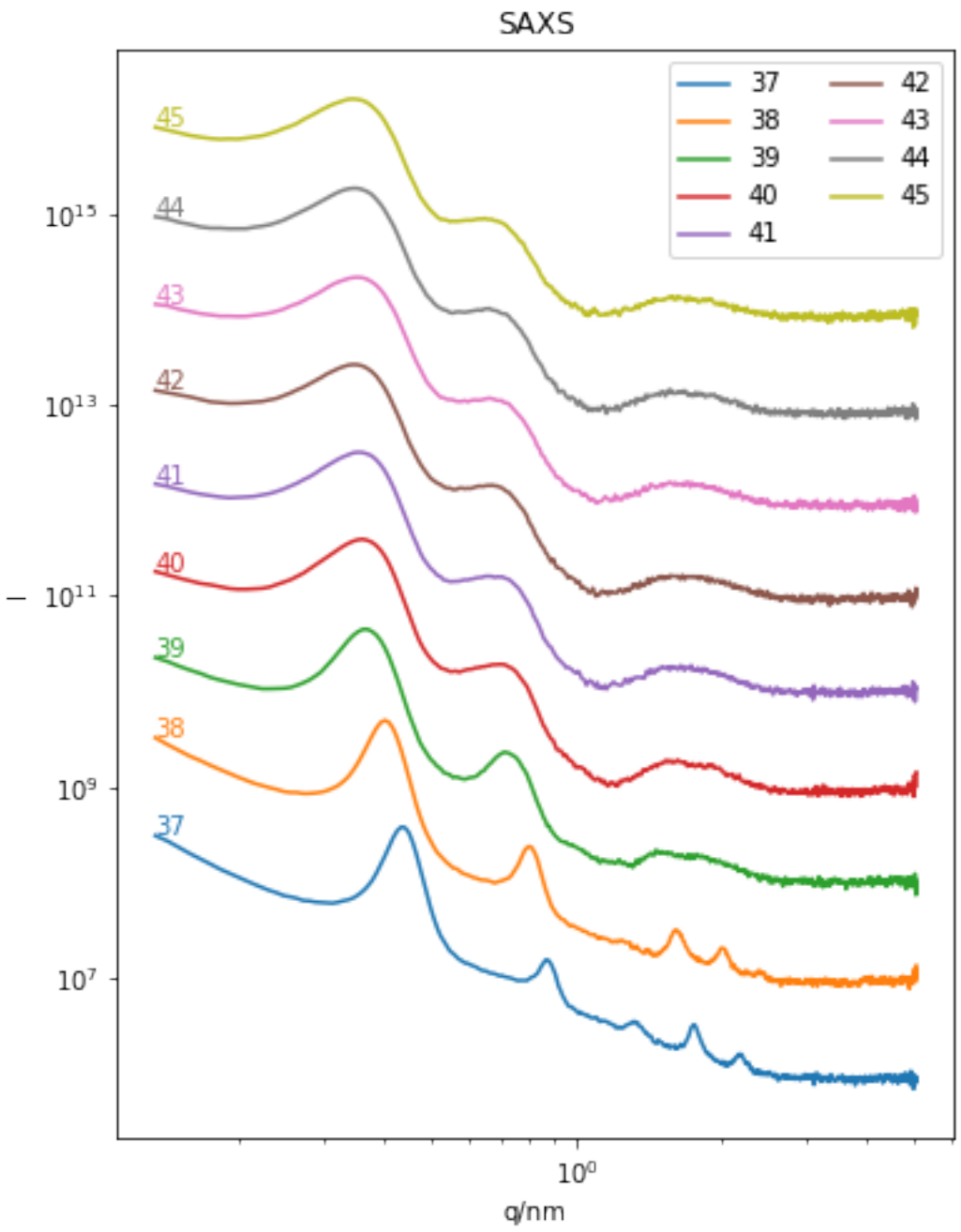
Para realizar um stack, utilizamos uma propriedade que tanto numpy arrays e pandas series possuem: quando uma operação aritmética é aplicada num desses objetos, essa operação é aplicada em todos os membros desse objeto. Assim, cada curva foi multiplicada por uma potência de 10 relativa a sua posição na lista. Então o primeiro item foi multiplicado por 10^0, o segundo por

10^1, etc.

Vamos agora utilizar nossos conhecimentos para colocar um texto no começo de cada linha desse gráfico com stack, com o texto da legenda, na cor da linha.

In [25]:
```
# Mesma figura, com espaçamento entre os dados

saxs = glob.glob('*.dat')
saxs.sort() # Vai do menor para o maior
fig, ax = plt.subplots(1, 1, figsize=(6,8))

for i, dado in enumerate(saxs):
    df = pd.read_csv(dado, sep='  ', engine='python', names=['q', 'I'])
    ax.plot(df['q'], df['I'] * 10 ** i,  label=dado[:-4])
    ax.text(df.loc[0, 'q'], df.loc[0, 'I'] * 10 ** i, dado[:-4], color=f'C{i}')

ax.set(xscale='log', yscale='log', title='SAXS', xlabel='q/nm', ylabel='I')
ax.legend(ncol=2)
```

Out[25]:
```
<matplotlib.legend.Legend at 0x7f82167d3eb8>
```



6 Exercícios

Utilizando o que você aprendeu até agora, plote os dados da pasta 'dados-2' utilizando os métodos para separar os dados obtidos nos exercícios da aula de pandas. Os gráficos de calorimetria e DSC são na escala linear, já os de reologia são na escala log.