Física Computacional - FSC-5705

MatPlotLib

Ao ler este documento provavelmente você já escutou falar de gnuplot e de xmgrace e deve se perguntar, outro???, sim, é outro de entre muitos. decidi escrever sobre esta biblioteca porque a diferencia do xmgrace (que é o que eu uso) dá para instalar no windows de uma forma relativamente fácil; comparado ao gnuplot ele é menos árido no que diz respeita a colocar formulas dentro do gráfico; finalmente ele uma faz parte do python, uma plataforma que está em franco crescimento.

Se você é usuario windows instalei o pythonxy esse pacote já traz tudo o que você precisa em python como um estudante de Física, se você se sente bem pagando pois acha que terá a segurança de alguma empresa, de uma olhada em enthought. Se você é usuário mac, o python está no mac e agorá é só instalar o matplotlib. Finalmente se você é usuário Linux a vida fica mais o menos fácil, usuários de ubuntu (e derivados):

```
[usuario@python: ]# sudo apt-get install python-matplotlib
```

Usuários Debian debem entrar como root e instalar:

```
[root@python: ]# apt-get install python-matplotlib
```

Usuário de derivados do Red Hat (fedora, centos, scientific Linux, etc...)

```
[root@python: ]# yum install python-matplotlib
```

Outros Linux-like (BSD) procure no google por matplotlib + sua distribuição.

Gráficos Simples

Trabalhar com o matplotlib lembra o gnuplot já que se cimentam numa linguagem de script, mas a diferença é que o matplotlib o faz sobre o python, onde se comporta como um outro módulo mais. Assim a leitura e tratamento de dados (ajustes polinomiais ou não lineares) é feito com o Python e no fim mandar a imprimir com o matplotlib.

Exemplo simples: lendo e plotando os dados

Nosso primeiro exemplo consiste em plotar um conjunto de dados correspondente a uma parábola, baixe esse dados clicando aqui ou crie sua própria tabela. Os script em python podem ser feitos como o kate (ou o bloco de notas do windows). O script abaixo deve estar na mesma pasta onde os dados estão colocados

```
1  #!/usr/bin/env python
2
3  #Modulos python
4  import numpy as np
5  import scipy.interpolate
6  import matplotlib as mpl
7  import matplotlib.pyplot as plt
8
9  #****************************************
10
11 #le os dados e armazena em 2 vetores: x, y.
12 x, y = np.loadtxt('dados01.dat', unpack=True)
13
14 #plotando os dados
15 plt.plot(x, y, '*')
16 plt.show()
```

Para executar no Linux você digita python *nome_do_script.py*, note que tradicionalmente se coloca **.py** como extensão dos scripts python.

```
[usuario@python: ]# python script01.py
```

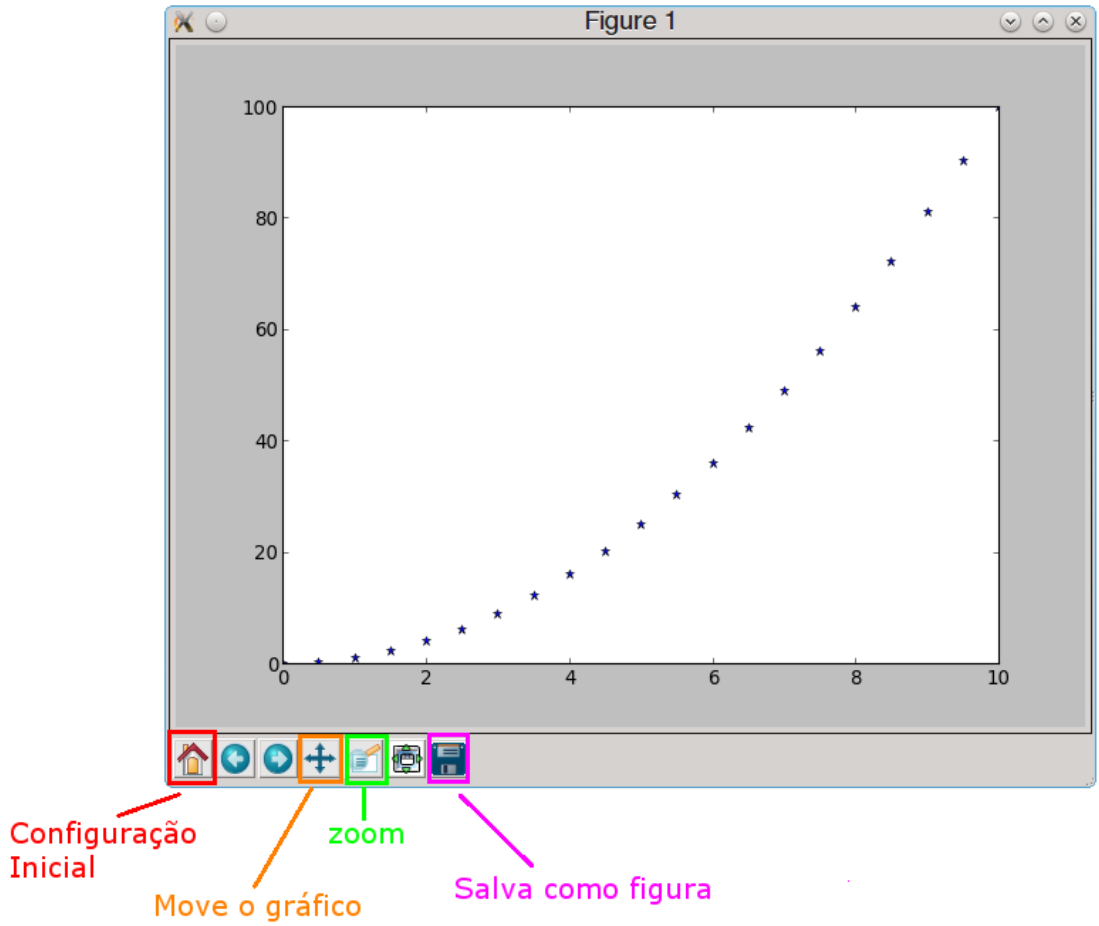

Figura 01: gráfico simples de um conjunto de dados com matplotlib.

O resultado desse rodar script é o gráfico é mostrado na figura 01.

Tipo de simbolo

Observe que na linha 15 do script se manda a plotar os dados de x contra y utilizando $*$ para representar os pontos, além de $*$ existe outras opções:

No script	No gráfico
.	ponto
o	circulo
,	pixel
v	Triangulo para baixo
^	Triangulo para acima
<	Triangulo para esquerda
1	Trípode para baixo
2	Trípode para cima
3	Trípode para esquerda
4	Trípode para direita

No script	No gráfico
s	quadrado
p	pentágono
*	estrela
h	hexágono
H	hexágono rotado
+	mais
x	cruz (x)
D	diamante
|	linha vertical
-	linha horizontal

Tipo de linhas

Em lugar de utilizar pontos, podemos unir os dados utilizando linhas

```
1  #!/usr/bin/env python
2
3  #Modulos python
4  import numpy as np
5  import scipy.interpolate
6  import matplotlib as mpl
7  import matplotlib.pyplot as plt
8
9  #****************************************
10
11 #le os dados e armazena em 2 vetores: x, y.
12 x, y = np.loadtxt('dados01.dat', unpack=True)
13
14 #plotando os dados
15 plt.plot(x, y, '-')
16 plt.show()
```

resulta no seguinte gráfico

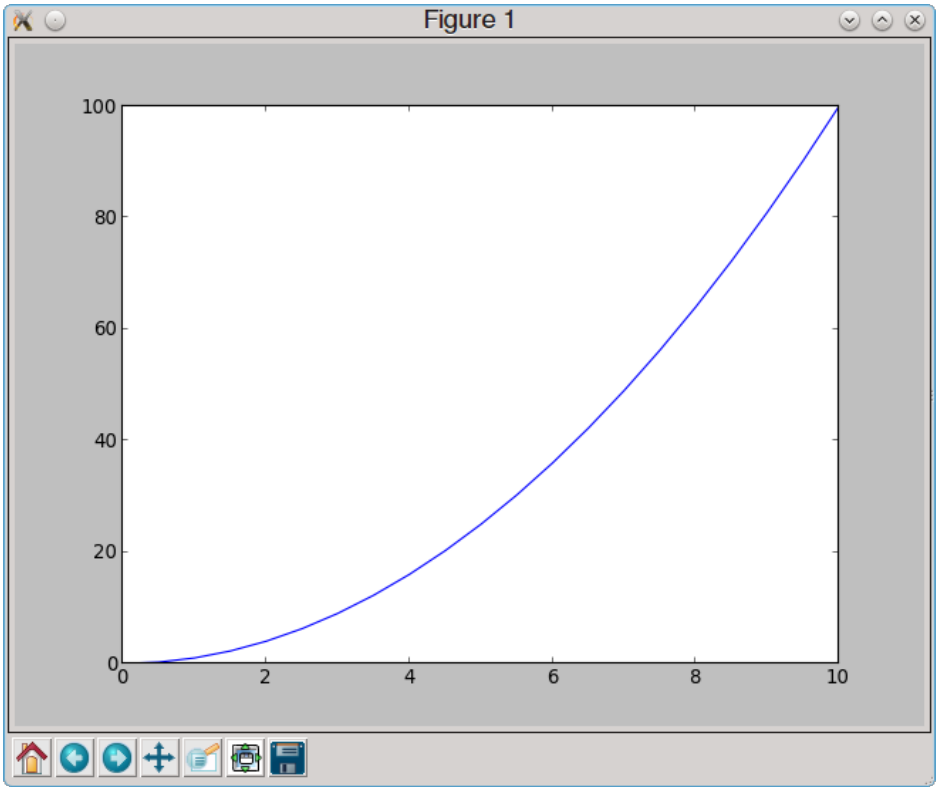

Figura 02: gráfico simples de um conjunto de dados com matplotlib.

Os tipos de linhas que o matplotlib suporta são

No script	No gráfico
-	linha sólida
--	linha tracejada
:	linha pontilhada
-.	linha traço-ponto

Cor

Para mudar a cor das linhas (símbolos) colocamos entre aspas a cor que desejamos

```
#!/usr/bin/env python

#Modulos python
import numpy as np
import scipy.interpolate
import matplotlib as mpl
import matplotlib.pyplot as plt

#****************************************

#le os dados e armazena em 2 vetores: x, y.
x, y = np.loadtxt('dados01.dat', unpack=True)

#plotando os dados
plt.plot(x, y, ':', color='m', marker="o")
plt.show()
```

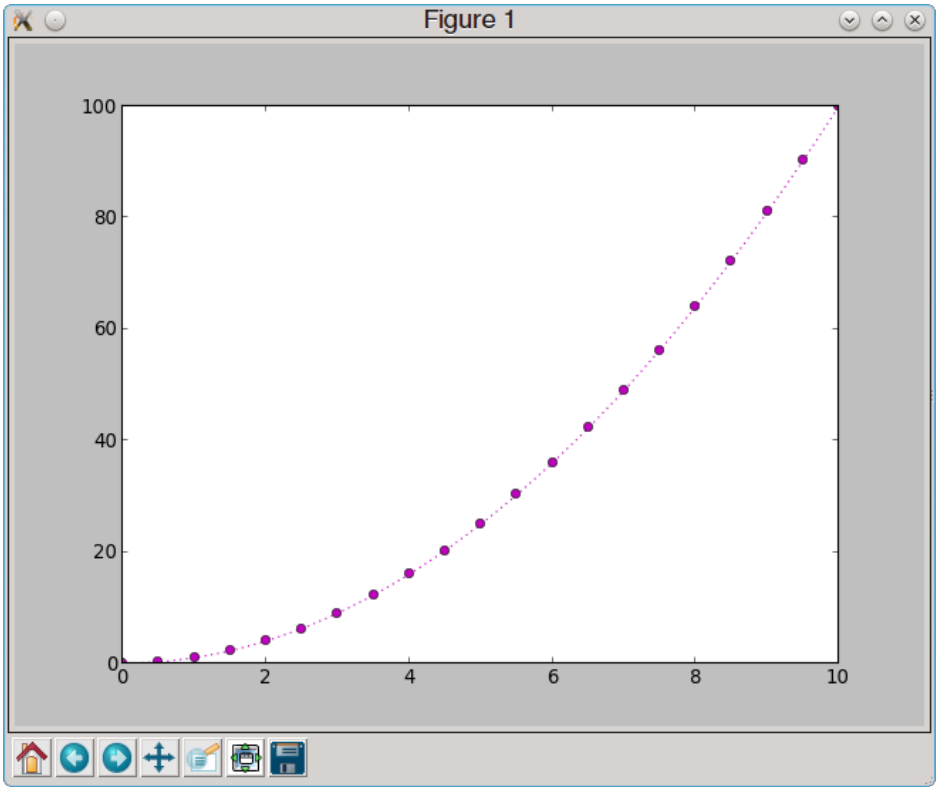

Figura 03: gráfico simples mudando a cor dos pontos.

As cores disponíveis são:

No script	No gráfico
b ou blue	azul
c ou cian	ciano
g ou green	verde
k ou black	preto
m ou magenta	magenta
r ou red	vermelho
w ou white	branco
y ou yellow	amarelo

Além dessas cores podemos utilizar cores ao estilo das usada em HTML: #d3ddff.

Atributos

No exemplo anterior vimos que a cor da linha foi passada como um atributo, de fato, todos os parâmetros que apresentamos até aqui são atributos do gráficos que podem ser explicitamente nomeados. O atributos que o matplotlib aceita estão dados na tabela a seguir:

No script	No gráfico
color ou c	cor
linestyle	tipo de linha

No script	No gráfico
linewidth	largura da linha
marker	tipo de símbolo
markeredgecolor	cor de borda do simbolo
markeredgewidth	largura da borda do símbolo
markerfacecolor	cor de recheio do símbolo
markersize	tamanho de símbolo

Vamos agora a plotar simultaneamente dois conjuntos de dados, utilizaremos a informação da tabela previa para poder identificar cada um das curvas. Para você testar terá que baixar o arquivo dados02.dat de aqui.

```
#!/usr/bin/env python

#Modulos python
import numpy as np
import scipy.interpolate
import matplotlib as mpl
import matplotlib.pyplot as plt

#*****************************************

#le os dados e armazena em 2 vetores: x, y.
x, y = np.loadtxt('dados01.dat', unpack=True)

#le os II dados e armazena em 2 vetores: x, y.
x1, y1 = np.loadtxt('dados02.dat', unpack=True)

#plotando os dados
plt.plot(x, y, linestyle='-', color='black', marker="o", \
     markersize=20,  markerfacecolor="w")

plt.plot(x1, y1, linestyle='-.', color='red', marker="o",\
     markersize=10, markeredgecolor="blue", markerfacecolor="w")

plt.show()
```

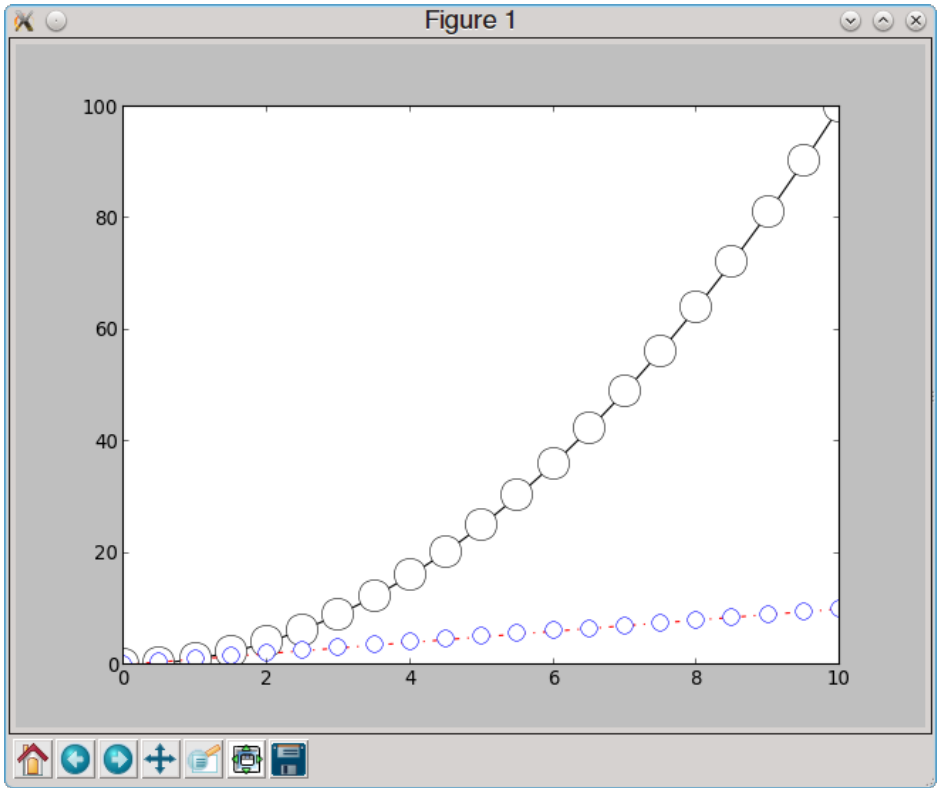

Figura 04: Dois gráfico simples com definições de atributos.

Títulos, eixos

Agora abordaremos o texto que se coloca no gráfico. Na figura script embaixo se mostra como se coloca a legenda nos eixo, o título, e a legenda do próprio gráfico

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#Modulos python
import numpy as np
import scipy.interpolate
import matplotlib as mpl
import matplotlib.pyplot as plt

#****************************************

#le os dados e armazena em 2 vetores: x, y.
x, y = np.loadtxt('dados01.dat', unpack=True)

#le os II dados e armazena em 2 vetores: x, y.
x1, y1 = np.loadtxt('dados02.dat', unpack=True)

#plotando os dados
plt.plot(x, y, linestyle='-', color='black',\
    marker="*", label=u"posição")

plt.plot(x1, y1, linestyle='-.', color='red',\
    marker="o", label="velocidade")

plt.legend(loc='upper left')

plt.grid(True)

plt.xlabel('t (s)')

plt.ylabel(r'$Y(m)$   e   $V_y(\frac{m}{s})$')

plt.title("Exemplo")

plt.show()
```

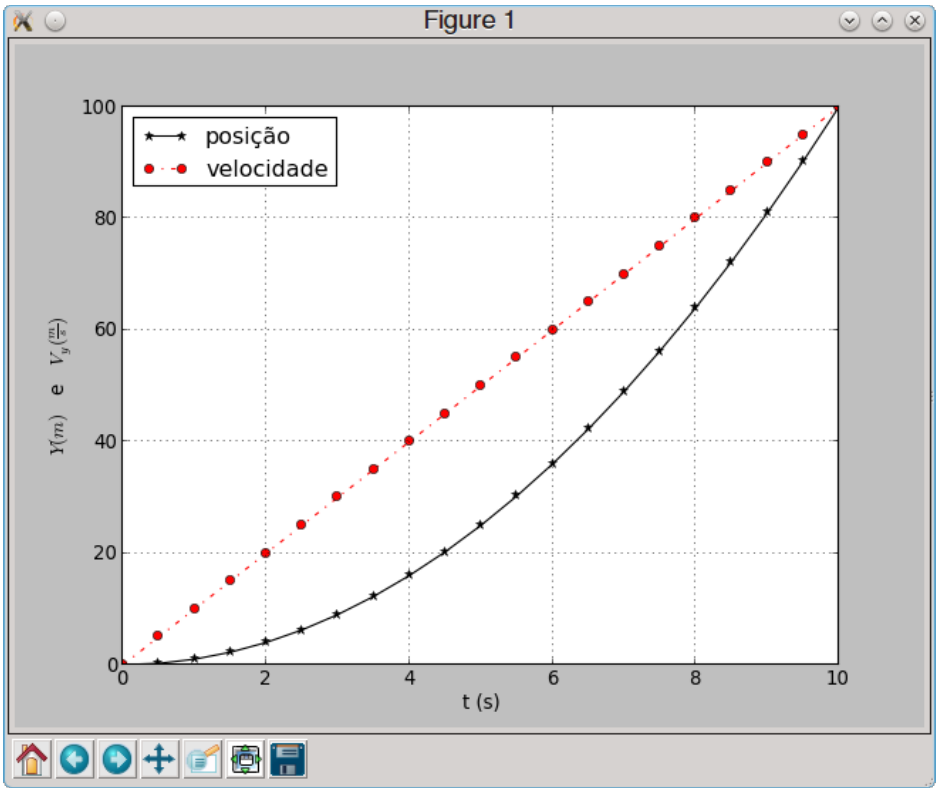

Figura 04: Legendas no matplotlib.

Note que a legenda associada à curva é colocada via o atributo **label**. Para que ele apareça no gráfico devemos colocar a linha 22 que nesse caso indica que a caixa com legenda deverá aparecer no canto superior esquerdo, além dessa posição temos:

No script	No gráfico
0 ou best	melhor posição
1 ou upper right	acima e à direita
2 ou upper left	acima e à esquerda
3 ou lower left	abaixo e à esquerda
4 ou lower right	abaixo e à direita
5 ou right	à direita
6 ou center left	centrado à esquerda
7 ou center right	centrado à direita
8 ou lower center	centrado embaixo
9 ou upper center	centrado acima
10 ou center	centralizado

Note que a tabela diz que podemos utilizar números ou letras (entre aspas) para indicar a posição (loc=10 é igual a loc='center'). De fato além de números podemos utilizar uma tupla de dados (coordenadas) onde por exemplo (0,0) é o canto inferior esquerdo e (0.5,0.5) o centro, assim um exemplo de posicionamento seria **loc=(0.05,0.8)** quase equivalente à **loc='upper left'**. Quando usamos português colocamos um **u** a frente da palavra como feito no label da linha 18 (**label=u'posição'**), para que isso funcione direito devemos salvar nosso script em *UTF-8* e colocar no cabeçalho **# -*- coding: utf-8 -*-**.

Para a legenda dos eixos x e y se utiliza **plt.xlabel('')** e **plt.ylabel('')**, respetivamente. Se você quer colocar uma equação ou letra grega, por exemplo no eixo x, devemos escrever entre $$ a equação (latex) é iniciar com r a legenda, como feito na linha 29 (para maiores informações de uma olhada em formulas com matplotlib)

Como se pode ver na figura acima o tamanhos das legendas nos eixos e o próprio título são pequenos demais, afortunadamente o *matplotlib* tem a opção ***fontsize*** que permite modificar esse comportamento padrão:

```
1  #!/usr/bin/env python                                                    ?
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
5  import numpy as np
6  import scipy.interpolate
7  import matplotlib as mpl
8  import matplotlib.pyplot as plt
9
10 #****************************************
11
12 #le os dados e armazena em 2 vetores: x, y.
13 x, y = np.loadtxt('dados01.dat', unpack=True)
14
15 #le os II dados e armazena em 2 vetores: x, y.
16 x1, y1 = np.loadtxt('dados02.dat', unpack=True)
17
18 #plotando os dados
19 plt.plot(x, y, linestyle='-', color='black',\
20     marker="*", label=u"posição")
21
22 plt.plot(x1, y1, linestyle='-.', color='red',\
23     marker="o", label="velocidade")
24
25 plt.legend(loc=(0.05,0.85))
26
27 plt.grid(True)
```

```
28
29  plt.xlabel('t (s)', fontsize=30)
30
31  plt.tick_params(axis='both', labelsize=20)
32
33  plt.ylabel(r'$Y(m)$   e   $V_y(\frac{m}{s})$', fontsize=30)
34
35  plt.title("Exemplo", fontsize=30)
36
37  plt.savefig('figura51.png')
38
39  plt.show()
```

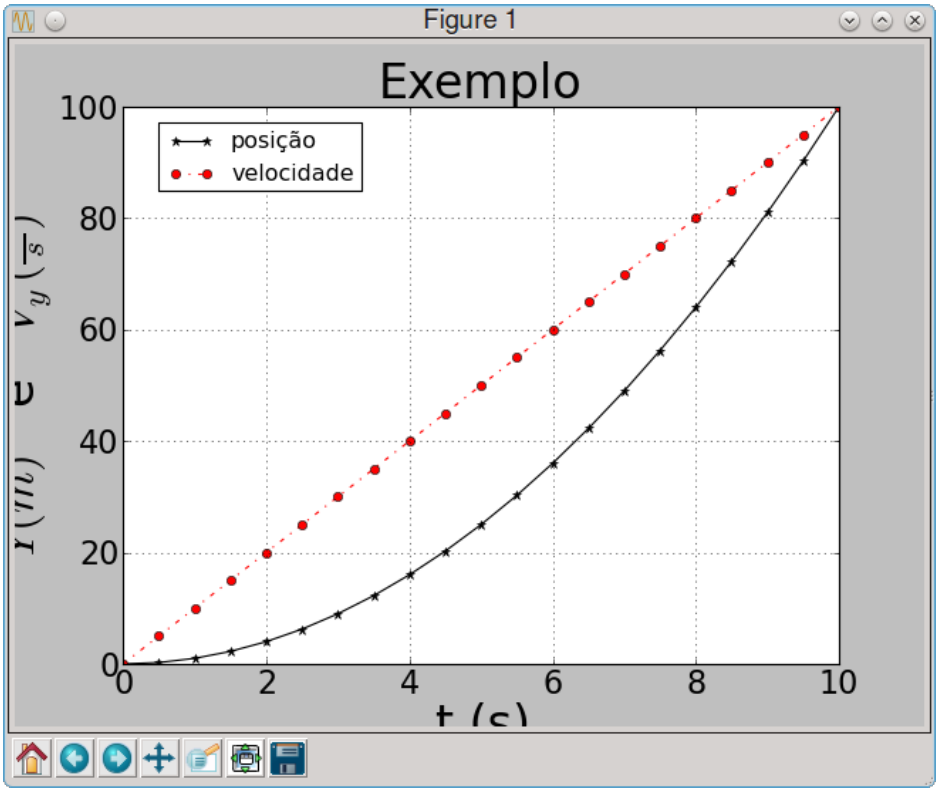

Figura 05-1: Tamanho das legendas no matplotlib.

Observe que agora temos um novo problema, as letras são maiores do que a área definida por padrão, para ajeitar esse comportamento indesejado utilizamos a função ***plt.tight_layout()***:

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4   #Modulos python
5   import numpy as np
6   import scipy.interpolate
7   import matplotlib as mpl
8   import matplotlib.pyplot as plt
9
10  #*****************************************
11
12  #le os dados e armazena em 2 vetores: x, y.
13  x, y = np.loadtxt('dados01.dat', unpack=True)
14
15  #le os II dados e armazena em 2 vetores: x, y.
16  x1, y1 = np.loadtxt('dados02.dat', unpack=True)
17
18  #plotando os dados
19  plt.plot(x, y, linestyle='-', color='black',\
20      marker="*", label=u"posição")
21
22  plt.plot(x1, y1, linestyle='-.', color='red',\
23      marker="o", label="velocidade")
24
25  plt.legend(loc=(0.05,0.85))
26
27  plt.grid(True)
28
```

```
29 plt.xlabel('t (s)', fontsize=30)
30
31 plt.tick_params(axis='both', labelsize=20)
32
33 plt.ylabel(r'$Y(m)$   e   $V_y(\frac{m}{s})$', fontsize=30)
34
35 plt.title("Exemplo", fontsize=30)
36
37 plt.savefig('figura51.png')
38
39 plt.tight_layout()
40
41 plt.show()
```

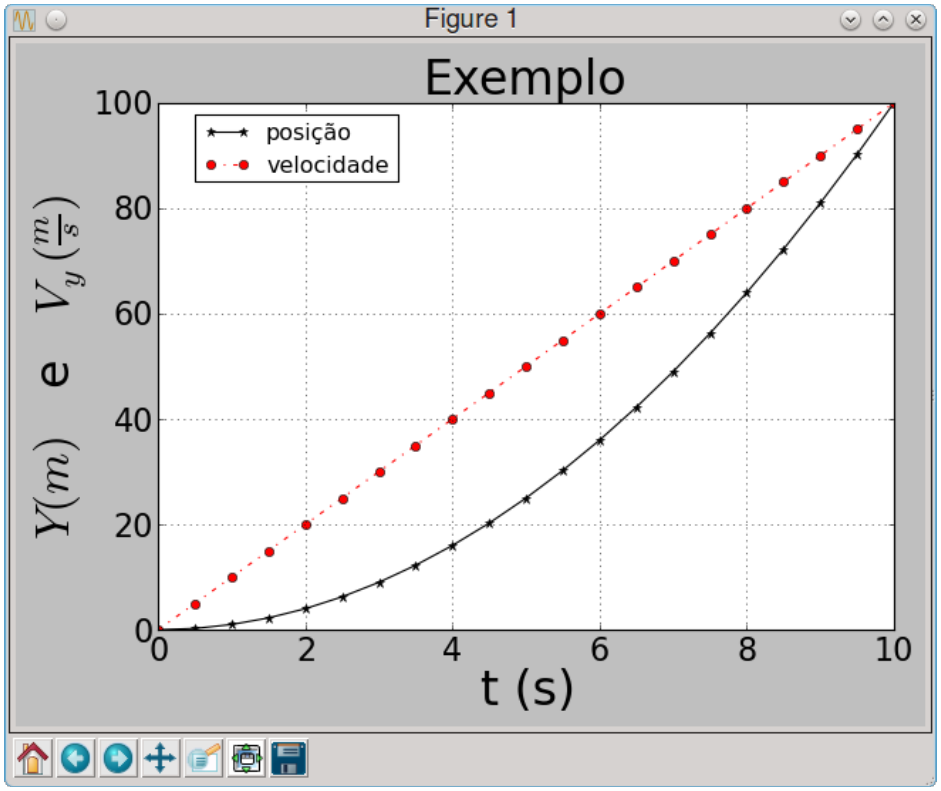

Figura 05-2: Tamanho das legendas no matplotlib.

Observe que além das legendas modificamos o tamanho dos números nos eixos, isso foi feito com a função ***plt.tick_params()*** na qual informamos o tamanho das novas letras (*labelsize*) e qual dos eixos será afetado (*axis*), que no caso escolhemos serem ambos (*'both'*), para maiores informações veja o manual no site.

Erro nos dados

Se temos dados resultantes de um experimento a eles está associado um erro resultante do processo de medição. Nesses casos não utilizamos a função **plt.plot** para imprimir e sim **plt.errorbar**, vejamos um exemplo, baixe os dados necessários para esse executar esse exemplo aqui

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
5  import numpy as np
6  import scipy.interpolate
7  import matplotlib as mpl
8  import matplotlib.pyplot as plt
9
10 #****************************************
11
12 #le os dados e armazena em 2 vetores: x, y.
13 x, y, err = np.loadtxt('dados04.dat', unpack=True)
14
15 #plotando os dados
16 plt.errorbar(x, y, yerr=err, linestyle='-', color='black', marker="*", label
17
```

```
18  plt.legend(loc='upper left')
19
20  plt.grid(True)
21
22  plt.xlabel(u'Indivíduos')
23
24  plt.ylabel(u'frequência')
25
26  plt.title("Grafico com erro em y")
27
28  plt.show()
```

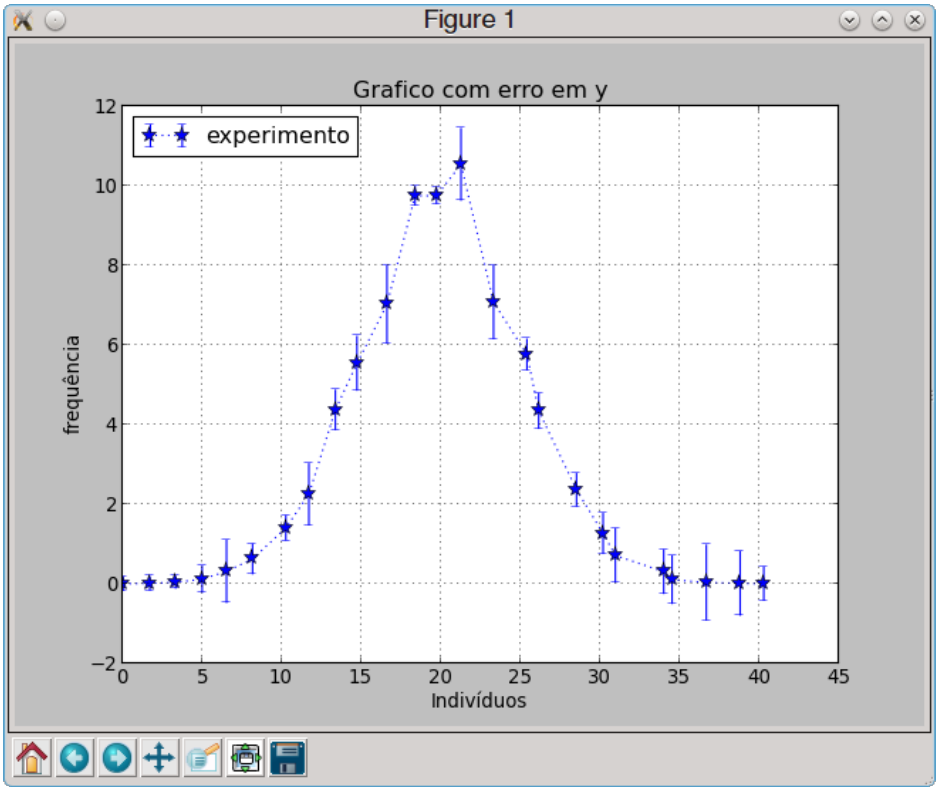

Figura 05b: Erro nos dados.

É possível associar um erro ao pontos nos eixo x (simultaneamente com y, dado um cruz no ponto), para isso utilizamos o parâmetro **xerr=** vetor com dados do erro.

Ajustes polinomial simples

No caso de um ajuste polinomial utilizamos a função **polyfit**:

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4   #Modulos python
5   import numpy as np
6   import scipy.interpolate
7   import matplotlib as mpl
8   import matplotlib.pyplot as plt
9
10  #****************************************
11
12  #le os dados e armazena em 2 vetores: x, y.
13  x, y = np.loadtxt('dados01.dat', unpack=True)
14
15  #calcula os coeficientes do polinomio
16  print 'coeficientes do polinomio'
17  coef = np.polyfit(x, y, 2)
18  print coef
19
20  #cria o polinomio
21  print '\nPolinomio ajustado'
22  pol = np.poly1d(coef)
23  print pol
```

```
24
25  #calcula os novos y com o juste
26  yajus = pol(x)
27
28  #cria o grafico
29  plt.plot(x, y, 'o')
30  plt.plot(x, yajus, '--')
31  plt.legend(['dados', 'ajuste'], loc=2)
32  plt.grid(True)
33  plt.xlabel('t')
34  plt.ylabel('V')
35  plt.title("Exemplo de ajuste")
36  plt.show()
```

```
[usuario@python: ]# python script06.py
coeficientes do polinomio
[ 1.00000000e+00 -3.92352444e-15 1.33070584e-14]


Polinomio ajustado
   2
1 x - 3.924e-15 x + 1.331e-14
```

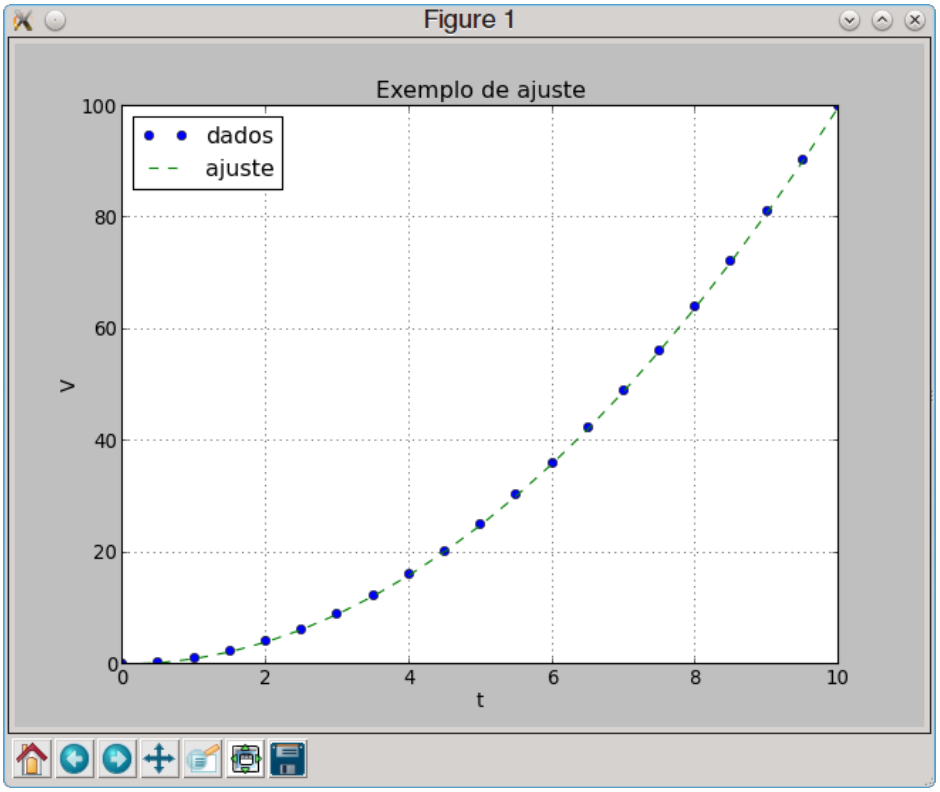

Figura 06: Ajuste polinomial.

A função **polyfit** devolve os coeficientes do ajuste polinomial, se esse coeficientes são passados à função **poly1d** ela vai montar a equação correspondente, como se mostra nas linhas 22 e 23. Com o polinômio criado na linha 22 podemos calcular os novos valores (yajus) como se mostra na linha 26.

Ajuste não linear

Para realizar o ajuste no linear de dados utilizamos a o conjunto de dados aqui armazenados, a esses dados aplicamos o próximo script

```
1  #!/usr/bin/env python                                                    ?
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
5  import numpy as np
6  import scipy.optimize
7  import matplotlib as mpl
```

```
8  import matplotlib.pyplot as plt
9
10 #*****************************************
11
12 #Função que desejamos ajustar
13 def gausiana(x, yo, xo, s):
14   y = yo * np.exp(-(x-xo)**2 / (2.0*s*s))
15   return y
16
17 #*****************************************
18
19 #le os dados e armazena em 2 vetores: x, y.
20 x, y = np.loadtxt('dados03.dat', unpack=True)
21
22 chuteInicial = np.array([7.0, 15.0, 3.0])
23
24 resAjuste, mcov = scipy.optimize.curve_fit(gausiana, x, y, p0=chuteInicial)
25
26 print "Resultado do ajuste: ", resAjuste
27 print u'\nMatriz de covariança (diagonal eh variança dos resultados):'
28 print mcov
29
30 #gera os y ajustados
31 yajus = gausiana(x, resAjuste[0], resAjuste[1], resAjuste[2])
32
33
34 #cria o grafico
35 plt.plot(x, y, 'o')
36 plt.plot(x, yajus, '-')
37 plt.legend(['dados', 'ajuste'], loc=2)
38 plt.grid(True)
39 plt.xlabel('x')
40 plt.ylabel('y')
41 plt.title(u'Exemplo de ajuste não linear')
42 plt.show()
```

```
[usuario@python: ]# python script07.py
Resultado do ajuste: [ 9.64515448 20.03009188 5.16476803]

Matriz de covariança (diagonal eh variança dos resultados):
[[ 1.10182138e-02 4.87617282e-05 -3.92403713e-03]
 [ 4.87617282e-05 4.21683166e-03 -1.99872232e-06]
 [ -3.92403713e-03 -1.99872232e-06 4.20210050e-03]]
```

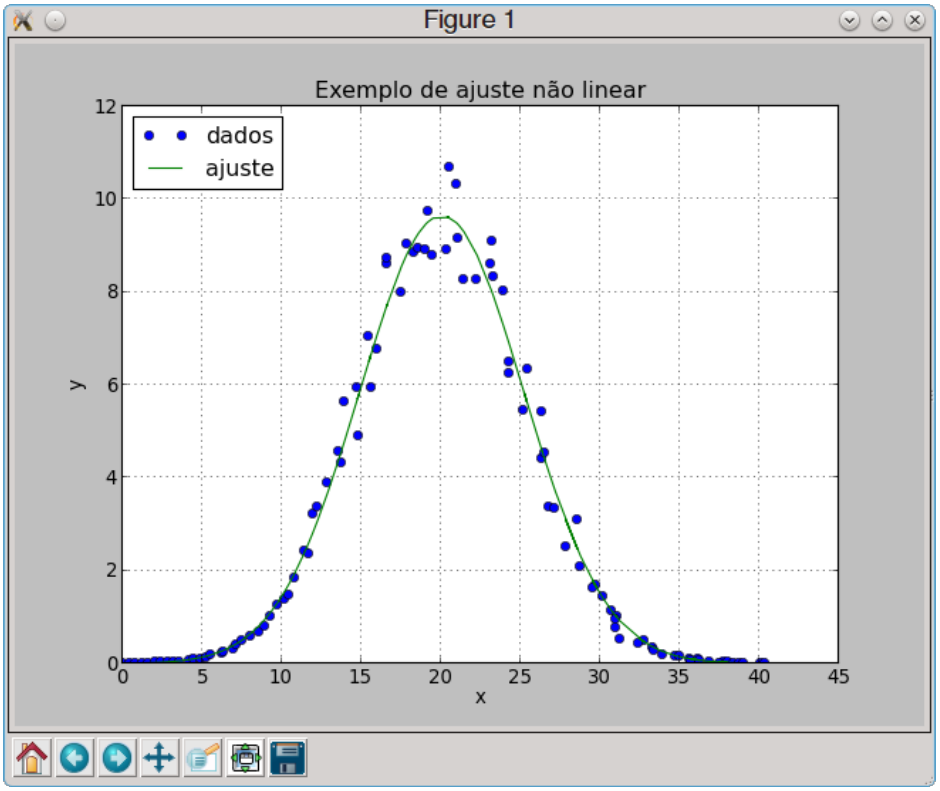

Figura 07: Ajuste não linear.

Para o ajuste não linear utilizamos a função scipy.optimize.curve_fit, essa função espera o nome de outra função a qual é o modelo a ser ajustado ao dados, no programa chamada de gausiana e definida entre as linhas 13 e 15; seguidamente são passado a scipy.optimize.curve_fit os vetores de dados x e y; finalmente damos para ela um chute inicial dos parâmetros que definem a função a ser ajustada. Como resultado do ajuste ela devolve um vetor e uma matriz, o vetor armazena parâmetros resultantes do ajuste na ordem que foram definidos na função, a matriz devolve a matriz de covariança.

Ajuste não linear com erro nos dados

Similar ao caso anterior, a diferença que agora a incerteza será colocada como dado da função **scipy.optimize.curve_fit**

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
5  import numpy as np
6  import scipy.optimize
7  import matplotlib as mpl
8  import matplotlib.pyplot as plt
9
10 #*****************************************
11
12 #Função que desejamos ajustar
13 def gausiana(x, yo, xo, s):
14   y = yo * np.exp(-(x-xo)**2 / (2.0*s*s))
15   return y
16
17 #*****************************************
18
19 #le os dados e armazena em 2 vetores: x, y.
20 x, y, yerr= np.loadtxt('dados04.dat', unpack=True)
21
22 chuteInicial = np.array([7.0, 15.0, 3.0])
23
24 resAjuste, mcov = scipy.optimize.curve_fit(gausiana, x, y, p0=chuteInicial,
25
26 print "Resultado do ajuste: ", resAjuste
27 print u'\nMatriz de covariança (diagonal eh variança dos resultados):'
28 print mcov
```

```
29
30  #gera os y ajustados
31  xajus = np.linspace(0.0, 40.0, 100) #gera 100 pontos desde 0 até 40
32  yajus = gausiana(xajus, resAjuste[0], resAjuste[1], resAjuste[2])
33
34
35  #cria o grafico
36  plt.errorbar(x, y, yerr, marker='o', linestyle=' ')
37  plt.plot(xajus, yajus, '-')
38  plt.legend(['dados', 'ajuste'], loc=2)
39  plt.grid(True)
40  plt.xlabel('x')
41  plt.ylabel('y')
42  plt.title(u'Exemplo de ajuste não linear')
43  plt.show()
```

```
[usuario@python: ]# python script07.py
Resultado do ajuste: [ 9.94095175 19.92305929 4.99153264]

Matriz de covariança (diagonal eh variança dos resultados):
[[ 0.00770007 0.00166504 -0.00244807]
 [ 0.00166504 0.00750754 -0.00107857]
 [-0.00244807 -0.00107857 0.0050674 ]]
```

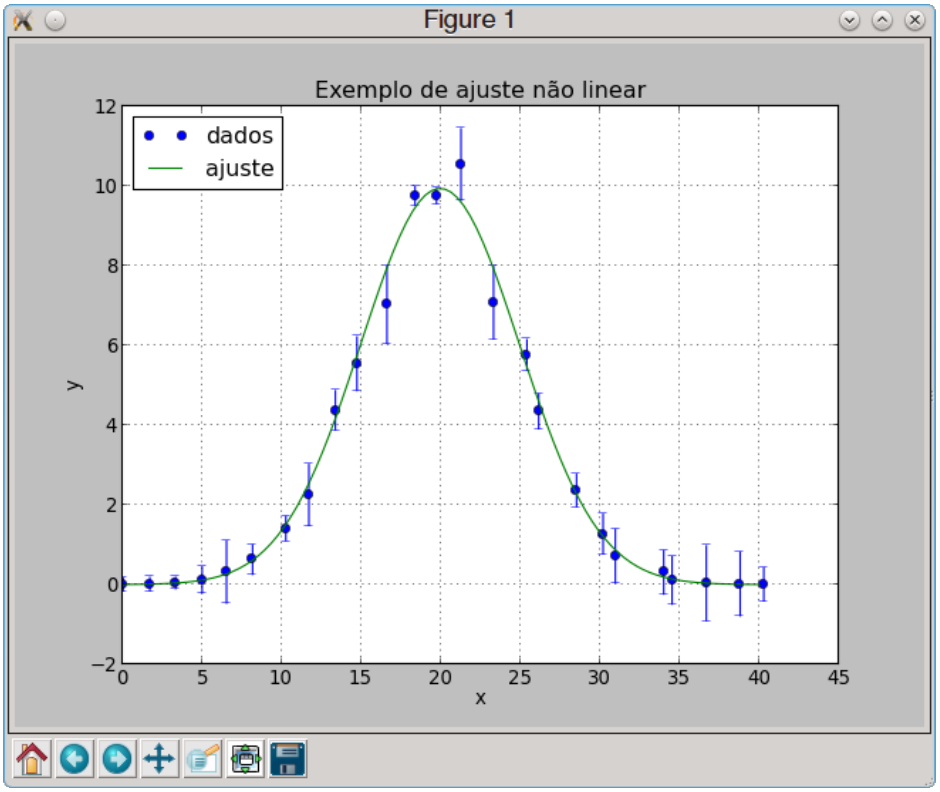

Figura 07: Ajuste não linear com erro.

Ajuste não linear de dados experimentais

Agora ajustaremos um conjunto de dados experimentais a uma função não linear. O dados resultaram de um experimento de difração por fenda simples onde a traves de algum tipo de ccd foi capturada a intensidade da figura de difração. Para o ajuste os dados foi necessário normalizar e deslocar os dados para centralizar na origem. Os dados foram ajustados utilizando o modelo de difração de Fraunhofer para fendas simples:

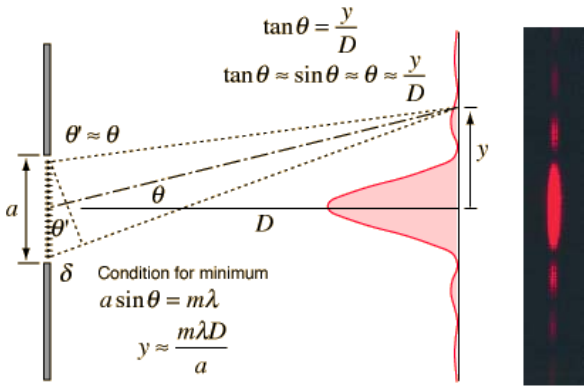

Figura 08: Analise da figura de difração.

$$I = I_0 \frac{\sin^2 \alpha}{\alpha^2}$$

onde

$$\alpha = \frac{\phi}{2} = \pi \frac{a \sin\theta}{\lambda}$$

Utilizando o seguinte script podemos ver os dados experimentais

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
5  import numpy as np
6  import scipy.optimize
7  import matplotlib as mpl
8  import matplotlib.pyplot as plt
9
10 #****************************************
11
12 #le os dados e armazena em 2 vetores: x, y.
13 x, y = np.loadtxt('dadosDifracao.dat', unpack=True)
14
15 #plotando os dados
16 plt.plot(x, y, '.')
```

Resulta na figura da esquerda, onde depois de se fazer um ampliação na região de interesse, resulta na figura da direita.

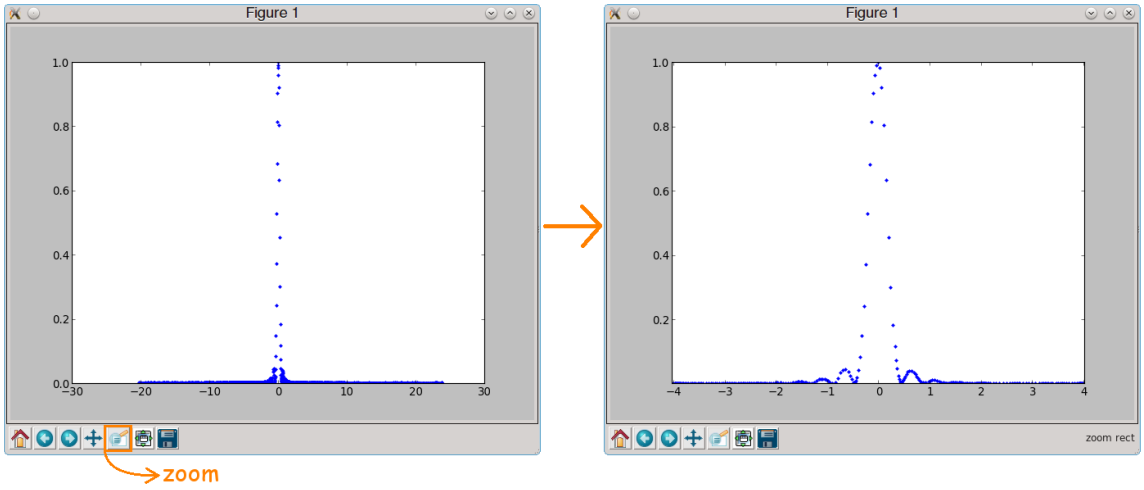

Figura 09: Dados e zoom ao dados.

Para realizar o ajuste vamos utilizar o seguinte script

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
```

```python
import numpy as np
import scipy.optimize
import matplotlib as mpl
import matplotlib.pyplot as plt

#*****************************************

#Função que desejamos ajustar
def difracao(x, a, yo):
  alphaD2 = np.pi * a * np.sin(x)/632.0
  y = yo * (np.sin(alphaD2)/alphaD2)**2
  return y

#*****************************************

#define duas matrizes vazias
x = []
y = []

#le os dados e armazena em 2 vetores temporarios: xtemp, ytemp.
xtemp, ytemp = np.loadtxt('dadosDifracao.dat', unpack=True)

for i in range(len(xtemp)): #percorre todos os elementos em xr
    if (np.abs(xtemp[i]) <= 1.30):  #limita os dados em x entre -1.3 e 1.3
        #coloca os dados recortados nos novos vetores x e y
        x.append(xtemp[i] + 0.01)  #desloca o grafico ligeiramente para a di
                                   #isso evita um dado exatamente em zero pois
        y.append(ytemp[i])

chuteInicial = np.array([1.0, 132.0])

resAjuste, mcov = scipy.optimize.curve_fit(difracao, x, y, p0=chuteInicial,

print "Resultado do ajuste: ", resAjuste
print u'\nMatriz de covariança (diagonal eh variança dos resultados):'
print mcov

#gera os y ajustados
yajus = difracao(x, resAjuste[0], resAjuste[1])

#cria o grafico
plt.plot(x, y, 'o')
plt.plot(x, yajus, '-')

plt.legend([r'$dados$', r'$I_0\frac{\sin \left( \pi\, \frac{a\,\sin \theta}{

plt.xlabel(r'$\pi\, \frac{a\,\sin \theta}{\lambda} (radiano)$', fontsize=20)
plt.ylabel(r'$I$', fontsize=20)

plt.title(u'Intensidade da luz que passa por uma fenda')
plt.show()
```

```
[usuario@python: ]# python script08.py
Resultado do ajuste: [ 1.41696333e+03 1.00950036e+00]


Matriz de covariança (diagonal eh variança dos resultados):
[[ 6.78262351e+01 2.18691091e-02]
 [ 2.18691091e-02 3.36704044e-05]]
```

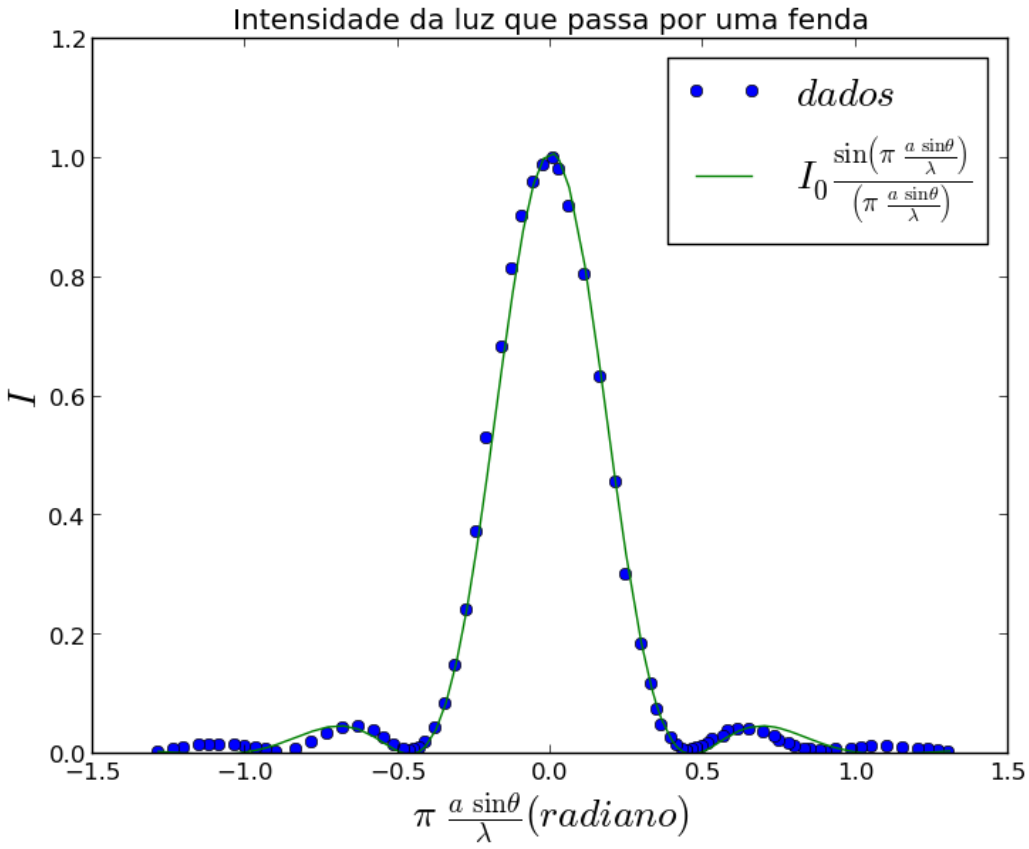

Figura 10: Ajuste aos dados experimentais.

Tópicos avançados

Aqui vamos tratar de assuntos que você muito provavelmente não terá que se preocupar tão cedo mas podem ser uteis no futuro próximo.

Para começar é necessário desconstruir, de certa forma, parte do que foi feito até aqui. Vamos criar um objeto (isso mesmo, de orientação a objeto) com as propriedades do módulo matplotlib.pyplot. Ao fazermos herdaremos as propriedades desse módulo e seu métodos. Para se conseguir isso primeiro criamos uma figura sobre a qual colocaremos nossos gráficos:

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  #Modulos python
5  import numpy as np
6  import scipy.optimize
7  import matplotlib as mpl
8  import matplotlib.pyplot as plt
9
10 #****************************************
11
12 papel = plt.figure()
```

Note que a variável que utilizei para armazenar o objeto criado chamei de papel pois tem o rol que um papel utilizado em um desenho. Os argumentos que podemos colocar dentro da função figura são

matplotlib.pyplot.figure(*num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=, **kwargs***)**

- *num:* é um numero utilizado para fazer referencia a uma dada figura, é isso que se armazena na variável papel.
- *figsize:* é o tamanho da figura *width x height* em polegadas
- *dpi:* resolução da figura
- *facecolor:* é a cor do papel

- *edgecolor:* cor da borda

Mais de um gráfico por folha

Exercícios 106

1. Utilizando matplotlib leia o seguinte conjunto de dados: dados01.dat e com eles cria um gráfico de pontos, coloque como título do gráfico: *exemplo 01*, no eixo das abscisas coloque *X* e no eixo das ordenadas *Y*, a legenda deve ser colocada no canto inferior esquerdo e informar que os pontos são *dados01.dat*. Realize o ajuste dos dados e plote o ajuste em conjunto com os dados, a legenda associada ao ajuste deve aparecer $y = \alpha x^2 + \beta x + \gamma$. Os pontos associados ao ajuste devem ser triangulo para acima de tamanho 7 e de cor azul; o ajuste deve ser em linha tracejada verde. Salve o resultado do ajuste como figura em formato png.
2. Utilizando matplotlib leia os seguintes conjuntos de dados: dados02.dat e dados03.dat, plote no mesmo gráfico ambos conjuntos de dados utilizando para ambos as seguintes propriedades: símbolo círculo em branco e borda vermelha, linha continua vermelha.
3. Crie um programa que leia os dados dados01.dat (são 25 pontos). Considere que esses dados correspondem a uma parábola, a qual tem por equação $y = a\,x^2 + b\,x + c$, calcule o vértice da parábola (x_0, y_0) e a posição do foco (x_f, y_f) sabendo que

$$x_0 = -\frac{b}{2.0a}$$
$$y_0 = \frac{4ac - b^2}{4a}$$
$$f = \frac{1.0}{4.0a}$$

   Aqui f é a distância do vértice ao foco (esta parábola é uma parábola com eixo paralelo ao eixo x).
   Dica: Se você pegar 3 pontos quaisquer dos dados (pegue os pontos 5, 15 e 20) poderá formar um sistema de 3 equações simultâneas. De uma dessas equações isole o b e substitua nas outras duas equações, dessa forma ficará com 2 equações e duas incógnitas (a e c). Sistemas de 2 equações são fáceis de resolver (caneta e papel para encontrar as equações finais).
4. Por definição a hipérbole é o lugar geométrico de todos os pontos do plano tais que a diferença da suas distâncias a dois pontos dados (focos) é constante. Sabendo que a equação de uma hipérbole é

$$\frac{(y - y_0)^2}{a^2} - \frac{(x - x_0)^2}{b^2} = 1$$

   onde (a, b) e $(-a, -b)$ são os vértices da hipérbole e os focos está a $c^2 = b^2 + a^2$ distantes de cada vértice. Faça um programa que mostre que todos os dados dados02.dat e dados03.dat correspondem a pontos sobre uma hipérbole na qual $x_0 = y_0 = 5.0$, $a = 4.0$ e $b = 2.0$.
5. Escreva um programa que leia desde um arquivo quatro números de ponto flutuante, os dois primeiros representam a amplitude e os dois últimos representaram o período de oscilação de duas ondas sinusoidais. Matematicamente essa ondas podem ser escritas como

$$y_i = y_{0i}\ \sin(\omega_i t)$$

   onde $\omega_i = 2\pi f_i$, sendo f_i a frequência (inverso do período) da onda (que pode ser $i = 1$ ou $i = 2$); y_{0i} é a amplitude de oscilação da onda $i = \{1, 2\}$. Faça um programa que crie 2 arquivos, *onda1.dat* e *onda1.dat* onde serão armazenado $n = 200$ pontos de y_i distribuídos uniformemente em 10 períodos de oscilação. O programa também deve imprimir a superposição de ambas ondas $y = y_1 + y_2$, com a mesma quantidade de pontos e no mesmo intervalo de tempo, no arquivo *superposicao.dat*. Com esses dados crie um gráfico com matplotlib onde apareçam as três curvas com legendas de cada curva e rotulo nos eixos do gráfico, o título do gráfico deve ser *Princípio de superposição*.