

Todo o poder do Gnuplot temperado com expressões regulares

Gráficos com Python

Uma das características mais marcantes das linguagens interpretadas é a facilidade com que podem "falar" com outros programas que estejam rodando. Este mês, vamos colocar o Python e o Gnuplot para conversar em uma sala trancada e ver que samba de irlandês maluco sai dessa briga.

POR JOSÉ MARÍA RUIZ E PEDRO ORANTES

www.sxc.hu - Trisha Shears

Existem muitos programas que geram gráficos a partir de informações relevantes. Por exemplo, muitos sites por aí apresentam ao internauta gráficos que mostram quantos acessos o site recebeu naquele mesmo dia. Também seria interessante controlar a quantidade de acessos por SSH na nossa máquina – uma excelente medida de segurança.

Gráficos até onde a vista alcança

Já que vamos falar de gráficos, por que não fazer algo de efeito? Uma das coisas que mais nos preocupa no dia-a-dia é a taxa com que cresce (ou decresce) o número de arquivos guardados no disco rígido, bem como o espaço que eles ocupam. Um programinha assim seria

útil? Não sabemos ao certo, mas pensar numa solução para esse problema será bastante divertido.

Nosso objetivo este mês é o seguinte: criar um programa em Python que, por meio de expressões regulares, separe cada uma das linhas em um arquivo de log, agrupe-as por data e gere um histograma com elas. Parece complicado, mas na verdade é bem simples – não será preciso um diploma em estatística para isso.

Em primeiro lugar, vamos conhecer as expressões regulares que o Python oferece. Com elas, poderemos facilmente processar o texto em busca dos padrões de data. Uma vez que tenhamos um conhecimento básico sobre o assunto, passaremos os olhos sobre o Gnuplot [1] e nas formas de usá-lo dentro do Python.

Expressões regulares: ame-as ou deixe-as

As expressões regulares trabalham com símbolos – em nosso caso, caracteres – e definem padrões de combinação com esses símbolos. Podemos, por exemplo, criar um padrão de letras para tentar, numa lista de palavras, separar aquelas que combinem com nosso padrão. Ou criar filtros que combinem com um determinado número IP.

Podemos fazer uma busca por parte de uma frase. Veja esta expressão regular: `From: algo.tw`. Ela combina com apenas uma frase, que é idêntica à expressão. Isso não é lá muito útil. Precisamos de algo que seja um pouco mais abrangente. O que nos interessa mesmo é definir padrões de pesquisa ou filtragem. Por exemplo, vamos definir um padrão

para capturar todos os endereços de correio eletrônico. Normalmente, um endereço de email será formado por:

- ⇒ Uma seqüência de caracteres pertencentes a um conjunto bastante restrito (por exemplo, não são permitidos acentos);
- ⇒ Uma arroba;
- ⇒ Outra seqüência com as mesmas restrições da primeira;
- ⇒ Uma última seqüência de domínio, que pode conter um ou dois pontos e três (.com) ou cinco (.com.br) caracteres.

Ou seja, algo como `spamer@spam.tw`. Para criar um padrão para esse tipo de cadeia de símbolos necessitamos seguir algumas regras.

As expressões regulares permitem que se imponham restrições. Podemos testar a ocorrência de determinados símbolos, bem como sua quantidade e posição. O padrão mais simples é o que busca a ocorrência obrigatória de um único caracter: basta indicar `q` para que se procure por esse caracter.

Podemos também indicar um conjunto de caracteres, em vez de um só. Por exemplo, se quisermos que a expressão regular busque a letra "K" ou a letra "c", encerramos o conjunto todo dentro de colchetes: `[Kc]`. Isso indica que esperamos um único caracter, e que esse único caracter deve ser ou "K" ou "c". Observe que, apesar de a ocorrência poder ser tanto de um quanto de outro caracter, é obrigatório que pelo menos um deles ocorra. Para que a ocorrência deixe de ser obrigatória e passe a ser opcional, basta colocar após os colchetes um símbolo de interrogação: `[Kc]?`.

Isso dá conta da ocorrência, mas e a quantidade? Em nossa introdução básica veremos apenas as regras mais simples, ou seja, as que indicam se o caracter apareceu "zero ou mais vezes" ou "uma ou mais vezes". No primeiro caso a seguir ("zero ou mais") colocamos o caracter

ou conjunto de caracteres dentro dos colchetes (`[]`) com um asterisco (`*`) logo depois. Para o segundo caso ("um ou mais") colocamos um sinal de adição (`+`) após os colchetes:

```
* [a]*: "", "a", "aa", "aaa"...
* [a]+: "a", "aa", "aaa",...
```

Como vimos, no colchete podemos ter uma ou mais letras. Agora imagine: seria bastante cansativo escrever todo o abecedário dentro dos colchetes caso queiramos filtrar apenas as palavras, sem pontuação ou acentuação, certo? Para isso, as expressões regulares dispõem de grupos de caracteres. O grupo `a-z` define todas as letras minúsculas do alfabeto, enquanto `0-9` define todos os dez algarismos arábicos. Dessa forma, o padrão `[a-z]*` representa uma seqüência de zero ou mais letras minúsculas quaisquer, enquanto `[0-9]+` define um número de pelo menos uma "casa" (ou seja, obrigatoriamente haverá um algarismo, que pode ser o algarismo "0").

Para o abecedário em maiúsculas, usamos o padrão `[A-Z]`. Existem também padrões auxiliares que definem, por exemplo, quais caracteres podem formar uma palavra e quais são usados como separadores. Quando um caracter tiver a possibilidade de ser mal interpretado – por exemplo, porque é, justamente, um dos caracteres que participam da sintaxe da expressão regular – ele precisa ser "escapado". Fazemos isso colocando uma barra invertida antes do tal caracter. É o caso típico do "-", que deve ser representado por `\-`. O próprio caracter de escape precisa ser "escapado" se precisarmos usá-lo como caracter comum: `\\`.

Depois dessa breve introdução, já conseguiríamos "detectar" um email com domínio de Formosa (que é o nome correto, em português, para Taiwan, `.tw`), por exemplo, com:

```
"[\-a-zA-Z0-9]+@[a-zA-Z0-9]+.tw".
```

Vamos imaginar agora um padrão que case com qualquer símbolo disponível no computador, à exceção do "9". Para isso, usamos um caracter "^" dentro dos colchetes, precedendo o "9": `[^9]`. Isso quer dizer: "qualquer caracter, menos o 9".

Dentro dos colchetes, o "^" define um padrão inverso. Dizemos o que não queremos e o padrão se corresponde com o resto. Mas lembre-se: isso é dentro dos colchetes. Se o "^" estiver para fora, o significado é totalmente diverso.

O "^" do lado de fora dos colchetes significa "antes de" ou "no início de". A expressão regular `^[0-9]` corresponde a uma cadeia de números que está no início da cadeia que buscamos: "3xuxu" casará com o padrão, "delegacia4" não casará com o padrão. Se os caracteres numéricos não estiverem no início, não serão "detectados" pela expressão.

Outro exemplo: `^[^0-9]+` corresponde a uma cadeia com pelo menos um símbolo que pode ser tudo, menos um algarismo. Portanto, temos que ser cuidadosos para

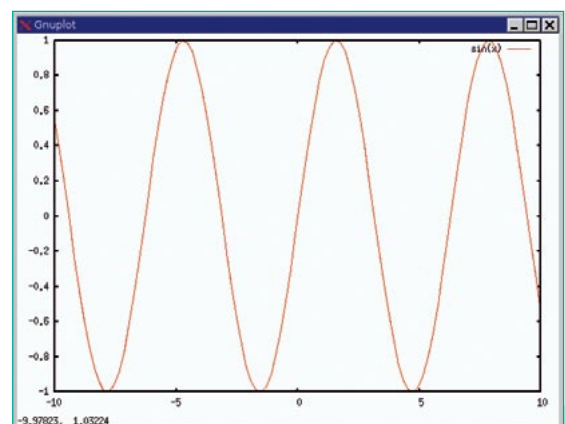


Figura 1: A função trigonométrica seno, desenhada pelo Gnuplot.

Listagem1: Gnuplotter.py

```

01 #!/usr/bin/python
02
03 from Numeric import *
04 import Gnuplot, Gnuplot.funcutils
05 import re;
06
07 class Grafico:
08
09     def __init__(self, cadeia):
10         self.g = Gnuplot.Gnuplot(debug=1)
11         self.g.title('Meu lindo gr ico')
12         self.g('set data style linespoints')
13         self.g('set boxwidth 0.9 absolute')
14         self.g('set style fill solid 1.000000
border -1')
15         self.g('set style histogram clustered gap
5 title 0, 0')
16         self.g('set style data histograms')
17         self.g('set title \"Linhas com ' + cadeia
+' \")')
18
19         self.hash = {}
20         self.cadeia = cadeia
21
22     def carregaDados(self, arquivo):
23
24         descritorArquivo = file(arquivo,'r')
25
26         self.hash = {}
27
28         expr = re.compile(self.cadeia);
29
30         for Linha in descritorArquivo:
31             if( re.search(expr, Linha)):
32                 lista = Linha.split(' ')
33                 data = lista[0]+' '+lista[1]
34                 if (not self.hash.has_key(data)):
35                     self.hash[data] = 1
36                 else:
37                     self.hash[data] =
self.hash[data] + 1
38
39                 descritorArquivo.close()
40
41
42             def desenha(self):
43
44                 chaves = self.hash.keys()
45                 chaves.sort()
46
47
48                 etiquetas = "set xtics ("
49
50                 i = 0
51                 lista = []
52
53                 for chave in chaves:
54                     etiquetas = etiquetas + "\" " + chave
+ "\" " + str(i) + " , "
55                     lista.append([self.hash[chave],i])
56                     i = i+1
57
58                 etiquetas = etiquetas + "\"fin\" " +
str(i) + ")"
59
60
61                 self.g('set xrange [0:'+str(i-1)+']')
62                 self.g('set yrange [0:30]')
63
64                 self.g(etiquetas)
65
66                 self.g.plot(lista)
67                 raw_input('Clique em INTRO...\n')
68                 self.g.reset()
69
70
71 if __name__ == "__main__":
72     g = Grafico('ssh')
73     g.carregaDados('teste.txt')
74     g.desenha()

```

escolher exatamente o lugar onde vamos enfiar os modificadores.

Esse é o básico do básico das expressões regulares. Com o que temos até aqui, já podemos brincar de gráficos com o Gnuplot e o Python. A quantidade de modificadores que existem, entretanto, é vastíssima, e recomendamos ao leitor que estude o

assunto com bastante afinco. Há uma grande quantidade de tutoriais sobre expressões regulares na Internet. A maioria deles tem o inconveniente de estar em inglês (como este [2]) mas, como sempre, vem a nosso auxílio o impecável Aurélio Marinho Jargas [3] e seus excelentes artigos e tutoriais sobre o assunto em português.

Expressões regulares em Python

Na linguagem Python podemos usar as expressões regulares em nossos programas por intermédio do pacote *re* (de *Regular Expressions*). Esse pacote já faz parte do sistema base do Python, portanto não será necessário instalá-lo. A primeira coisa que vamos fazer é definir,

no código, uma expressão regular bem simples e brincar bastante com ela durante o exercício.

```
>>> import re
>>> p = re.compile("[ab]+")
>>> print p
<_sre.SRE_Pattern object at 0x81418c0>
>>>
```

Criamos um objeto que representa a expressão regular `[ab]*` – ou seja, uma seqüência de "a" e "b" que pode ter tamanho zero ou maior. Ao imprimir o valor de "p", vimos que ele nada mais é do que um objeto como qualquer outro em Python.

Vamos, então, passar a esse objeto algumas cadeias para ver o que ele nos diz:

```
>>> print p.match("34",1)
None
>>> print p.match("abaaab",1)
<_sre.SRE_Match object at 0x81b4800>
>>> print p.match("abaaabc",1)
<_sre.SRE_Match object at 0x81b4800>
>>>
```

No primeiro exemplo vemos que o método `match` do objeto `p` nos devolve a mensagem `None`. Isso significa que não existe nenhuma subcadeia dentro da cadeia que passamos ao método que corresponda ao padrão definido. No segundo exemplo vemos como o objeto se comporta quando toda a cadeia passada "bate" com o padrão.

No terceiro exemplo, vemos que o objeto se comporta da mesma maneira que no exemplo anterior. Entretanto, na cadeia que passamos ao objeto no terceiro exemplo há a letra "c", que não combina com o padrão. O que ocorre aqui é que o Python procura alguma subcadeia que passe pelo crivo da expressão regular, mesmo que nem toda a cadeia case com o padrão.

Usamos, nesse exemplo inicial, o método `match()`, que simplesmente busca a correspondência no início da cadeia ou a partir da posição arbitrada por nós (em nosso caso, indicamos a posição 1). Veja os exemplos abaixo, em que arbitramos a posição zero.

```
>>> print p.match("cabaab",0)
None
>>> print p.match("abaaab",0)
<_sre.SRE_Match object at 0x81c4c60>
>>>
```

Existe outro método chamado `search()`, que se comporta de maneira parecida com as expressões regulares em *Perl*. Esse método busca as correspondências entre a cadeia informada e a expressão regular e permite gerar uma lista com todas elas. O mais útil, entretanto, é o método `findall()`, que devolve uma lista com todas as ocorrências do padrão.

```
>>> resultado = p.search("abaaabcaaaa",1)
<_sre.SRE_Match object at 0x81b4800>
>>>
>>> p.findall("abaaabcaaaa",0)
['abaaab', 'aaa']
```

Introdução ao Gnuplot

O Gnuplot [1] é um programa que cria gráficos a partir de dados. É possível inserir nele tanto uma tabela com valores já computados ou uma função matemática. Com o comando `plot`, o programa gera um gráfico a partir deles e o despeja na tela em uma nova janela. A forma como o gráfico é criado pode ser alterada ao bel-prazer do usuário.

Ao executar o Gnuplot aparecerá uma linha de

comando, que chamaremos de shell. O Gnuplot possui uma linguagem e comandos próprios. Já que temos um shell, por que não brincamos com ele?

```
gnuplot> f(x) = sin(x)
gnuplot> plot f(x)
gnuplot>
```

Alakazam! O programa abre uma janela parecida com a da **figura 1**. É a função trigonométrica seno, que desenhamos com apenas duas instruções. Interessante, não? O comando `plot` possui muitos parâmetros que podem alterar seu comportamento. Podemos desenhar, por exemplo, várias funções simultaneamente, usar pequenos pontos quadrados ou redondos para diferenciá-las, interpolar pontos no gráfico e assim por diante. O Gnuplot permite também realizar muitos tipos de gráficos: de duas e três dimensões, histogramas, pizzas...

Uma das características mais úteis do programa é a facilidade com que podemos exportar o gráfico para imagens em formatos padronizados, como *gif* ou *png*. Podemos fazer todos os testes usando, como saída, uma janela na tela e, quando estivermos seguros dos resultados (e dos parâmetros que temos que ajustar no Gnuplot), podemos gerar um arquivo de imagem. ➡

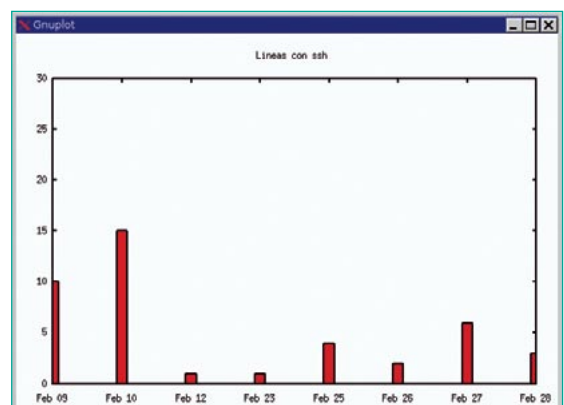


Figura 2: O gráfico de nosso aplicativo exemplo.

O programa Gnuplot

O Gnuplot foi desenvolvido tendo em mente a automatização: podemos fazer scripts usando os comandos de seu shell. A idéia é poder usar o Gnuplot na montagem dos famosos “encanamentos” que os usuários de UNIX tanto adoraram. Os “geeks” mais avançados criam aplicativos inteiros usando *pipes* (em português, cano) – e não poderíamos ficar de fora dessa moda.

O Gnuplot necessita de um terminal para funcionar. Esse terminal atua como driver de impressão e pode ser um aparelho (como uma impressora), um arquivo de imagem (um *JPEG*, por exemplo) ou diretamente uma janela em seu ambiente de trabalho. Aproveitando-nos dessa característica podemos ir lapidando a configuração de um gráfico diretamente no shell do Gnuplot – basta visualizá-lo em uma janela. Quando tudo estiver perfeito, a única coisa que precisaremos fazer é substituir o terminal.

Podemos consultar o estado das variáveis usando o comando `show`:

```
gnuplot> show terminal
terminal type is x11
gnuplot>
```

Com o comando `help`, podemos conhecer os vários terminais suportados. Para o nosso programa, só nos interessam os terminais *x11* e *JPEG*. Para trocar o terminal, temos que alterar o conteúdo de uma variável. Para isso, temos o comando `set`:

```
gnuplot> set terminal pstricks
Terminal type set to 'pstricks'
gnuplot> set terminal x11
Terminal type set to 'x11'
Options are '0'
gnuplot>
```

Uma vez definido o terminal, temos que configurar os parâmetros do gráfico. No Gnuplot, inúmeros parâmetros afetam a forma como o gráfico será desenhado. Por exemplo, se não especificarmos nada e informarmos uma série de pontos, o programa pinçará os pontos no plano cartesiano mas não traçará o gráfico que os une. Outros parâmetros ajustam a precisão dos intervalos usados.

As variáveis são configuradas com o comando `set`. O comando `help set` mostra todas as variáveis com as quais podemos bulir. Por exemplo, para dar título ao gráfico usamos:

```
gnuplot> set title "Nosso lindo
gráfico"
```

Depois que as variáveis tiverem sido corretamente configuradas, basta gerar o gráfico com o comando `plot`. O `plot` também possui inúmeras opções e permite que vários gráficos sejam traçados simultaneamente. Cada um dos gráficos possui seu próprio conjunto de opções. Por exemplo:

```
gnuplot> plot sin(x) using impulses
```

Isso desenhará uma função seno usando linhas verticais em vez de uma linha ligando os pontos.

Gnuplot + Python = Diversão

Para poder utilizar o Gnuplot dentro do Python necessitamos baixar e instalar a biblioteca *py-gnuplot*, a partir do site oficial [4] ou por meio de um pacote próprio de sua distribuição de Linux. Obviamente, também será necessário instalar o programa Gnuplot. Para o nosso sisteminha, usaremos um método anônimo que nos permite mandar comandos diretamente ao Gnuplot:

```
>>> g('set title \"Nosso Lindo
Gráfico\"')
```

Uma vez que tenhamos tudo configurado, invocaremos o método `plot()`. A biblioteca *py-gnuplot* foi criada para poupar trabalho na hora de acessar o Gnuplot e permitir o uso de funções numéricas (*Numeric* [5]) como fonte de dados. Podemos criar gráficos a partir de:

- ⇒ Arquivos
- ⇒ Listas do Python
- ⇒ Funções

Em nosso programinha, desenharemos o gráfico a partir de uma lista gerada pela busca com expressões regulares. Uma peculiaridade marcante do *py-gnuplot* é a sua grande rapidez ao desenhar um gráfico. Se emitirmos o comando `plot()` agora, não veremos nada! Para parar o programa durante o traçado do gráfico devemos empregar o comando `raw_input()` entre `plot()` e `reset()`.

```
>>> a = [[1,2], [2,2], [4,5]]
>>> g.plot(a)
>>>
```

Isso desenhará um gráfico ligando os pontos (1,2), (2,2) e (4,5).

Nosso programa

O código de nosso programa está na [listagem 1](#). Temos uma classe chamada `Grafico` com três métodos. O primeiro é o obrigatório `__init__()` e tudo o que faz é preparar o ambiente para o Gnuplot: define e inicializa algumas variáveis e declara uma a que chamamos de *hash*. Depois, guarda em outra variável de instância a cadeia que usamos como padrão.

O segundo método é `carregaDados()`; ele abre um arquivo em modo de leitura, compila a expressão regular e, para cada linha do arquivo, compro-

va se a linha contém a expressão regular que buscamos. Se a linha combinar com a expressão, usamos o caracter de espaço (" ") para dividi-la. Tomam-se as duas primeiras palavras da linha, que representam a data, para usar como chave para um dicionário que chamamos de hash. Se essa data não estiver ainda no dicionário, ela deve ser introduzida lá com o valor 1. Se a data já estiver no dicionário, somamos

1 ao valor lá guardado. Dessa forma, o dicionário terá registro do número de ocorrências de cada padrão classificadas por data.

O terceiro método é `desenha()`. O método atribui identificadores aos pontos tanto nos eixos cartesianos quanto no gráfico – para isso, usa a variável `xtics`. Esses identificadores serão as datas que usamos como chaves do dicionário hash. Depois, preparamos uma

lista contendo as coordenadas X e Y dos pontos a traçar, definimos o alcance (comprimento) dos eixos X e Y e, por fim, traçamos o gráfico.

O programa é dividido em três partes: a criação do objeto `Grafico`, a invocação de seu método `carregaDados()` com o nome do arquivo apropriado e a chamada ao método `desenha()`. Como resultado final teremos um gráfico parecido com o mostrado na **figura 2**. ■

SOBRE OS AUTORES

José María Ruíz está terminando seu Projeto de Conclusão de Curso na Faculdade de Engenharia Técnica em Informática e Sistemas. Já há sete anos usa e desenvolve software livre, desde os velhos tempos da telinha preta do *DOS* até o moderno *FreeBSD*. Atualmente, trabalha na *Animatika*, uma empresa de Málaga que se dedica ao software livre.

Pedro Orantes está cursando o 3º ano de Engenharia Técnica em Informática e Sistemas e, simultaneamente, o 3º ano de Engenharia Técnica em Gestão de Informática em Madri. Usa o Linux há seis anos no computador de trabalho, tanto para trabalhos de escritório como para desenvolvimento. Está desenvolvendo seu Projeto de Conclusão de Curso com *Jython*.

INFORMAÇÕES

- [1] Gnuplot: www.gnuplot.info
- [2] Regular Expressions HOWTO: www.amk.ca/python/howto/regex/
- [3] Aurélio explica expressões regulares: www.aurelio.net/er/
- [4] Py-Gnuplot: gnuplot-py.sourceforge.net
- [5] Numeric: numeric.scipy.org/



GO-Global[®]

Suas aplicações na Web em minutos.

Web Enabling

O GO-Global[®] publica na Web suas aplicações, legadas ou não (Windows[®], Unix[®] e Linux[®]), sem a necessidade de reescrever uma linha de código sequer, mantendo todas as funções originais e preservando os investimentos anteriores em software e hardware.

Virtual Office

O GO-GLOBAL[®] é 100% seguro! Agora sim seus colaboradores poderão acessar seus aplicativos corporativos de qualquer lugar do planeta, a partir de qualquer computador, como se estivessem conectados localmente.

Server Based Computing

Com o protocolo exclusivo Rapid X, o GO-GLOBAL[®] dispensa a aquisição de licenças adicionais (Terminal Server) exigidas pelas demais soluções Server Based Computing.

A solução Rápida, Simples e Econômica.



Faça o download gratuito
por 30 dias
www.go-global.com.br



Telefone: (11) 4153-5850
comercial@arcsystem.com.br