Programando em `gnuplot`

Praciano-Pereira, T.

Departamento de Matemática
Universidade Estadual Vale do Acaraú
5 de abril de 2008
tarcisio@member.ams.org
pré-prints do Curso de Matemática de Sobral
no. 2008.1
Editor Tarcisio Praciano-Pereira
tarcisio@member.ams.org

Resumo

Neste artigo estou mostrando como é possível escrever programas em `gnuplot` apresentando alguns exemplos e fazendo comparações dos mesmos com programas escritos em `C`. Sugiro que o leitor abra uma sessão de `gnuplot` ao acompanhar a leitura deste artigo, e execute os exemplos que artigo contém. Na bibliografia, ao final, se encontra de onde baixar `gnuplot` que é um programa distribuido livremente.

`gnuplot` é muito frágil como ambiente de programação, entretanto representa uma opção interessante porque é um ambiente disponível com grande universalidade, para várias plataformas, gratuito, e sabe fazer gráficos de modo que é possível automatizar algumas experiências criando uma dinâmica interessante no ensino de vários tópicos do Cálculo.

Começo, na primeira seção, com uma breve descrição do pacote `gnuplot`, na segunda seção discuto alguns exemplos simples, um programa para calcular a função de Fibonacci, outro para calcular a *delta de Dirac* e um terceiro para calcular a função característica de um intervalo. Finalizo, na terceira e última seção, apresentando um exemplo mais complicado, cálculo de alguns coeficientes de Fourier junto com o gráfico da "aproximação" trigonométrica de uma função, este programa está disponível, pode ser baixado livremente, em local citado na bibliografia.

O conteúdo deste artigo pode ser facilmente modificado para ser usado com os vários capítulos do Cálculo Diferencial e Integral.

palavras chave: programação com gnuplot, funções recursivas, função de Fibonacci, polinômio trigonométrico.

1 gnuplot, um ambiente para fazer gráficos

`gnuplot`[1] é um programa voltado para fazer gráficos, é um pacote computacional que pode ser usado com um ambiente gráfico para apresentaçoes de Matemática ou pode ser chamado iterativamente de dentro de um programa para construir gráficos com uma matriz de pontos produzidos pelo programa.

Eis um exemplo de uma sessão com o programa na qual se pode ver a página onde o programa pode ser encontrado e baixado. O resultado, ao se dar `enter` ao final da última linha é o gráfico da função

$$f(x) = x^2 \tag{1}$$

no intervalo padrão $[-10, 10]$ que pode ser alterado com o comando

```
set xrange [-30:35]
```

no exemplo acima o intervalo foi alterado para $[-30, 35]$.

Uma pequena dificuldade para usar o programa é que ele exige a comunicação usando a sintaxe de uma linguagem de programação, na verdade a sintaxe da linguagem `C`.

```
G N U P L O T
Version 4.0 patchlevel 0
last modified Thu Apr 15 14:44:22 CEST 2004
System: Linux 2.6.18-6-k7

Copyright (C) 1986 - 1993, 1998, 2004
Thomas Williams, Colin Kelley and many others

This is gnuplot version 4.0.  Please refer to the documentation
for command syntax changes.  The old syntax will be accepted
throughout the 4.0 series, but all save files use the new syntax.

Type ‘help‘ to access the on-line reference manual.
The gnuplot FAQ is available from
http://www.gnuplot.info/faq/

Send comments and requests for help to
<gnuplot-info@lists.sourceforge.net>
Send bugs, suggestions and mods to
<gnuplot-bugs@lists.sourceforge.net>


Terminal type set to 'x11'
gnuplot> f(x) = x**2 + 3*x + 4
gnuplot> plot f(x), 0
```

[1] Os autores discutem o nome do programa e dizem que é `gnuplot` e não `Gnuplot`

O programa tem uma ajuda muito rica e há vários textos em português mostrando como usar o programa, por exemplo [2], e o próprio programa traz uma ajuda que pode ser obtida usando o comando `help`. Esta ajuda *on line* do programa é um pouco sintética e serve apenas para quem já tenha alguma experiência com o programa.

Vou partir do pre-suposto que o leitor conhece `gnuplot` e sabe usar os comandos básicos:

plot, print

e sabe definir uma função como no exemplo acima. Isto é suficiente para tudo que eu fizer neste artigo com mais dois outres comandos, que vou apresentar no local certo, dando exemplos de como funcionam.

Como é possível definir funções em `gnuplot` e como o pacote aceita funções recursivas, e tem o comando `if/else`, então é possível programar em `gnuplot` é o que estarei demonstrando nas próximas duas seções.

2 Funções recursivas no `gnuplot`

As funções recursivas são bem conhecidas nas aulas de Matemática, aqui quero me referir às *sucessões recursivas* e a *função de Fibonacci* é dos exemplos mais conhecidos.

Aqui está a definição usual da Fibonacci

Definição 1 *Sucessão de Fibonacci*

$$F(1) = 1 \tag{2}$$
$$F(2) = 1 \tag{3}$$
$$n \leq 3 \rightarrow F(n) = F(n-2) + F(n-1) \tag{4}$$

As funções recursivas são um exemplo de *engenharia* lógica das mais primorosas e todo programador coloca em nível muito alto o uso de funções recursivas em programas com o desafio de evitar que o programa se quebre por falta de memória. As funções recursivas são o substituto natural dos controles lógicos para fazer laços como `while, for`, e portanto na ausência destes, como é o caso do `gnuplot`, as *funções recursivas* junto com o condicional `if/else` preenchem esta lacuna, e em `gnuplot` existe `if/else`.

Vou logo mostrar a sua implementação no `gnuplot` para permitir que o leitor acompanhe o artigo efetuando algumas experiências.

```
gnuplot> F(n) = (n==1)?1:(n==2)?1:F(n-2)+F(n-1)
```

O primeiro passo agora é deslindar o segredo místico desta expressão.

Como foi dito na introdução `gnuplot` espera que a comunicação seja feita usando a sintaxe da linguagem `C` que é semelhante à do `Pascal`, `Fortran` ou `BASIC` com algumas exceções, duas das quais temos acima.

1.) "`==`" é o operador lógico que testa igualdade, em `C`, copiado pelo `Java`, `C++`, `Python`.

2.) "`if/else`" aqui temos um `if/else` extremamente compacto formado de dois operadores "`?, :`. É fácil de absorver esta estrutura lógica, o símbolo de interrogação está perguntando se a condição entre parentesis é verdadeira:

   (n==1)?1

   pode ser lido:

   "*Se n igual a 1* for verdadeira ? então 1"

   e o operador "dois-pontos" é a parte "senão" (`else`). Desta forma podemos ler

   (n==0)?1:0

   como:

   "*n igual a zero é verdadeira ? então 1 senão 0*"

Com esta forma de escrever `if/else` temos uma definição muito compacta da função de Dirac, que ficaria em `gnuplot` assim

```
Dirac(n) = (n==0)?1:0
gnuplot> print Dirac(10)
0
gnuplot> print Dirac(0)
1
gnuplot> print Dirac(-1)
0
gnuplot> Dirac(3)
         ^
         invalid command
```

Na última linha do exemplo eu me esqueci de usar o comando `print` e deixei ficar o erro porque nem sempre ele é fácil de ser entendido: `gnuplot` ainda não aprendeu a escrever o valor de uma função, é necessário o comando `print`.

Outro exemplo uma função-onda quadrada, a função característica de um intervalo

```
chi(x) = (x<=-1)?0:(x<=1)?1:0
```

que define $\chi_{[-1,1]}$.

Vou terminar esta seção mostrando como calcular uma soma de Riemann com gnuplot, definindo-a, também, como uma função recursiva.

O que desejo, no cálculo aproximado da integral, é que sejam acumulados (somados) na variável "`soma`" os valores de uma função f em cada um dos nós de uma partição. Antes de dar início ao processo, devo limpar a memória impondo

```
soma = 0
```

Vou usar uma partição uniforme porque simplifica o algoritmo e não quero provar a existência da integral mas sim calcular o seu valor aproximadamente

nos casos em que a existência da integral já esteja demonstrada. O processo de acumulação dos valores se dá com o comando:

```
soma = soma + f(x)
```

Aqui vem o poder das funções recursivas, elas tem um ponto de parada controlado por um `if()` que não sendo `verdadeiro` deixam executar uma condição `else` onde novamente aparece a equação da função:

- `gnuplot> Riemann(a,b,soma,delta)=\`
- `(a>b)?soma*delta:Riemann(a+delta,b,soma+f(a),delta)`

A contrabarra no meio de uma linha elimina, para `gnuplot`, sinal de `fim de linha` possibilitando-nos a escrita de um comando ao longo de várias linhas.

É o *sinal* de `fim de linha` que indentifica para `gnuplot` que o comando foi encerrado. O corpo da função esta na segunda linha, eu vou explicar como funciona.

- "`a>b`" é a condição de parada desta função, é controlado por um `if` aqui representando pela interrogação colocada depois da condição. Quando ela se tornar verdadeira o programa para devolvendo o valor da soma multiplicado pelo *delta*, observe que foi aplicada, aqui, a propriedade distribuitiva da multiplicação de *delta* pela soma das alturas de f em cada ponto inicial dos sub-intervalos da partição uniforme.
- "`Senão`" for verdadeira, é o `else`, aqui representado pelos dois pontos, ao final do produto `soma*delta`, a função é novamente chamada com seus parâmetros atulizados:
  - Um novo valor para o parâmetro `a`, substituido-o por `a + delta`, que substitui, num *laço* convencional, o comando "`a = a + delta`".
  - Um novo cíclo da função será executado agora com o parâmetro `soma` atualizado com uma parcela $f(a)$. Também aqui este novo valor passado à função substitui o comando "`soma = soma + f(x)`" que apareceria num *laço* convencional.
- `Parâmetros invariantes` são o segundo , `b`, que contém o extremo final do intervalo de integração, e o último, `delta`, que contém a precisão escolhida para o cálculo aproximado da integral.
- `Para rodar` chamo `Riemann(0,1,0,0.001)` em que
  - Atribuo ao parãmetro `a` o valor inicial do intervalo de integração, neste exemplo estou calculando a integral (aproximada) de f no intervalo $[0, 1]$.
  - O parâmetro `b` recebe o valor 1, a extremidade final do intervalo de integração.

  - Ao parâmetro soma é atribuido o valor inicial 0, limpando a memória em que vão ser guardados (acumulados), os valores de f em cada um dos nós da partição (exceto o último).
  - Atribuo ao parâmetro delta, a precisão escolhida para o cálculo, o valor 0.001.

- executando

```
gnuplot>f(x) = x**2
gnuplot> print Riemann(0,1,0,0.001)
```

Se você esquecer o comando "print" vai der um erro ao final da execução, porque um "comando", na lógica do gnuplot, deve começar com uma das palavras chave do programa, seguido de parâmetros. Riemann(0,1,0,0.001) é um parâmetro do comando print.

Observe que é possivel definir a função f depois que se tiver definido a função Riemann(), isto torna possível calcular integrais de diversas funções, apenas trocando a equação de f depois que você tiver definido o programa Riemann().

Veja o resultado, que você pode facilmente reproduzir porque gnuplot está comummente instalado em qualquer micro, e não estando é fácil baixá-lo e instalá-lo, na biliografia, [3], está o site oficial do programa.

```
gnuplot> print Riemann(0,1,0,0.001)
```

0.3328335 $\approx \frac{1}{3}$.

A figura (1) lhe mostra um script pronto para rodar.

```
f(x) = x**2

Riemann(a,b,soma,delta)=\
   (a>b)?soma*delta:Riemann(a+delta,b,soma+f(a),delta)

print Riemann(0,1,0,0.001)
```

Figura 1: Script gnuplot para cálculo de soma de Riemann

Para comparação, apresento um programa recursivo, escrito em C para o cálculo de somas de Riemann, na figura (2), com duas observações:

- o objetivo não é mostrar que gnuplot é melhor do que C como linguagem de programação, seria inverídico, mas é um bom apoio para fazer pequenos programas, ao vivo, durante uma aula de Matemática computacional oferecendo a possibilidade, para estudantes o testem na hora,

```
/* programa Riemann;  */

# include "/ambiente.h"

float f(float x)
{
   return(x*x);
};

float  Riemann(float a, float b, float soma, float delta)
{
   if (a > b )  return( soma*delta);
   else return( Riemann(a+delta, b, soma+f(a),delta));
};

int main(void)
{
   float a=0, b=1, delta = 0.01;
   a     = entrada_float("Inicio do intervalo de integraçao  a= ", a);
   b     = entrada_float("Fim  do intervalo de integraçao b = ",b);
   delta = entrada_float("Precisão no cálculo da integral delta = ", delta);
   printf("o valor da integral, com precisão %2.3f é %2.3f \n",delta,
         Riemann(a,b,0,delta));
   return(0);
}
```

Figura 2: Soma de Riemann em C

- a bilioteca `ambiente.h` é necessária aos meus programas em C. Nela se encontram definidas funções de comunicação com o usuário, como

  `entrada_float()`

  e ela se encontra disponível, para uso público, sem restrições, distribuida sob GPL em [6].

3 Polinômios trigonométricos

Todos os resultado de que vou fazer uso aqui, podem ser encontrados em vários textos de Cálculo Avançado, por exemplo no livro de Djairo, [1], neles você encontrará as demonstrações de todas as afirmações que eu fizer aqui.

Vou desenvolver nesta seção um exemplo completo apresentando os coeficientes e o gráfico de uma função assim como do seu polinômio trigonométrico

de uma certa ordem. O resultado é um *programa em* `gnuplot` que o leitor poderá facilmente rodar. O programa não está apresentado no artigo, para não alongá-lo inutilmente, você poderá baixá-lo livremente daqui, [5].

Os polinômios trigonométricos são uma combinação linear dos vetores

$$sen(nx), cos(mx)$$

definidos num múltiplo do intervalo $[-\pi, \pi]$. Estes vetores são elementos do espaço das funções contínuas definidas neste intervalo. Desta forma podemos calcular a aproximação de uma função contínua definida em $[-\pi, \pi]$ com um certo erro, usando polinômios trigonométricos. Esta é a forma de apresentar *polinômios trigonométricos* que vou usar aqui.

Deixe-me, primeiro, fazer uma definição que vai simplificar a comunicação no resto do artigo.

Definição 2 (ondas básicas) sen_k, cos_k

Vou estabelecer a notação

$$sen_k(x) = sen(kx) \ ; \ cos_k(x) = cos(kx)$$

para todo $k \in \mathbf{N}$.

Como na Álgebra Linear, calculamos as projeções de um vetor relativamente aos elementos de uma determinada base, usando um *produto escalar*. Quando isto for possível, se tem "*um espaço com produto escalar*", é o caso do conjunto das funções contínuas definidas em um intervalo da reta.

Uma outra utilidade do produto escalar é a determinação das projeções de um vetor na direção de outro vetor, em geral este segundo vetor é o vetor unitário da direção que interessa.

Neste artigo quero calcular as projeções de uma função na direção das *ondas* básicas sen_k, cos_k. A expressão seguinte é muito usada na Álgebra Linear,

$$R \int_{-\pi}^{\pi} f(x)g(x)dx \tag{5}$$

como um exemplo de produto escalar, é bilinear e positiva definida, para qualquer número real $R > 0$ escolhido. Vamos escolher um número real adequado de modo que as ondas básicas tenham módulo 1. Começando com sen_1

$$\int_{-\pi}^{\pi} sin^2(x)dx = ||sen_1||^2 = \pi \tag{6}$$

vemos que se $R = \frac{1}{\pi}$ o módulo do sen_1 será 1, e o resultado imediatamente se aplica para cos_1.

Podemos usar agora um resultado do Cálculo, mudança de variáveis numa integral, para conseguir o resultado geral que precisamos:

$$\int_{-\frac{\pi}{n}}^{\frac{\pi}{n}} sin^2(nx)dx = \frac{1}{n}\int_{-\pi}^{\pi} sin^2(x)dx \tag{7}$$

Como no intervalo $[-\pi, \pi]$ tem n copias iguais de $sin^2(nx)$ concluímos que

$$\int_{-\pi}^{\pi} sin^2(nx)dx = \int_{-\pi}^{\pi} sin^2(x)dx = \pi \tag{8}$$

A lógica é a mesma para mostrar que o módulo de todos os vetores sen_k, cos_k é π e a conlusão é que podemos corrigir o produto escalar escrevendo $R = \frac{1}{\pi}$ fazendo com que todas as ondas básicas tenham módulo 1.

As integrais de quaisquer duas ondas básicas diferentes é zero, em outras palavras são vetores ortogonais:

$$< sen_k, sen_j >= 0 \text{ se } k \neq j \tag{9}$$

$$< sen_k, cos_j >= 0 \text{ para todos os } j, k \in \mathbf{N} \tag{10}$$

Nestas ondas básicas o parâmetro representa as frequências básicas da teoria das ondas com a frequência zero representando uma onda constante, com intensidade 1, que corresponde a $cos(0x)$, ou seja cos_0. Claro que também podemos definir sen_0, mas sem grande vantagem. Isto vai criar um problema que deixaremos para resolver posteriormente.

Temos um produto escalar relativamente ao qual as ondas básicas são vetores unitários o que nos coloca no ambiente comum da Álgebra Linear apenas com uma situação especial: temos uma infinidade de vetores unitários e ortogonais, um espaço que é de dimensão não finita, o espaço de todas as ondas, todas as combinações lineares das ondas básicas, esta é basicamente a teoria de Fourier das funções 2π-periódicas.

Vamos aplicar esta teoria usando uma função contínua no intervalo $[-\pi, \pi]$ mostrando um uso didático importante para o `gnuplot`, que é o nosso objetivo aqui. A teoria de Fourier representa apenas a motivação.

Dada uma função contínua f, definida no intervalo $[-\pi, \pi]$ definimos as projeções

$$a_k =< f, cos_k >= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)cos(kx)dx \tag{11}$$

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)dx \tag{12}$$

$$b_k =< f, sen_k >= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)sen(kx)dx \tag{13}$$

$$b_0 = 0 \tag{14}$$

A expressão de a_0, equação (12), está errada, mas o erro é consequência da definição e vejamos como ela pode ser corrigida de maneira bem natural. É interessante, e instrutivo, deixar ficar o erro nas experiências com os alunos para que eles vejam que há algo errado para posteriormente mostrar como corrigir, e porque.

No caso do a_0 temos um vetor que não normal, tem módulo 2π, a onda de frequência zero, e um vetor que não seja de módulo 1 vai introduzir lixo na projeção dos vetores tomados em sua direção. A solução para isto (é o problema já mencionado) consistem em normalizar este vetor o que significaria definir

$$cos_0(x) = \frac{1}{2\pi} \tag{15}$$

em vez disto se preferiu redefinir b_0

$$\tfrac{1}{2\pi} b_0 = \tfrac{1}{\pi} \int_{-\pi}^{\pi} f(x)dx \tag{16}$$

$$b_0 = \tfrac{1}{2} \int_{-\pi}^{\pi} f(x)dx \tag{17}$$

Agora definimos duas integrais, são denominações históricas, a "integral do coseno" e a "integral do seno", com as expressões para calcular os coeficientes de Fourier, em `gnuplot`, na figura (3).

```
gnuplot> four1(x,fim,n,soma)=\
   (x>=fim)?soma*delta:four1(x+delta,fim,n,soma+f(x)*cos(n*x))
gnuplot> four2(x,fim,n,soma)=\
   (x>=fim)?soma*delta:four2(x+delta,fim,n,soma+f(x)*sin(n*x))
```

Figura 3: integral do seno e do coseno

O programa, [5, `fourier.gnuplot`], calcula os coeficientes até um valor escolhido dentro do programa e usando a função f também definida no programa e finalmente faz os gráficos conjuntos de f e do polinômio trigonométrico

$$P_n(x) = \sum_{k=0}^{n} a_k cos(kx) + b_k sen(kx) \tag{18}$$

Para todo k os coeficientes a_k serão calculados dentro do programa com os comandos:

```
ak = four1(inicio,fim,k,0);
bk = four2(inicio,fim,k,0);
```

em que primeiro foi definida a função f, guardados os valores aproximados de $\pm\pi$, nas variáveis

```
inicio, fim
```

e, no programa, k varia desde zero até um valor escolhido para n.

O programa é distribuido livremente sob a `GPL` na página indicada. O autor ficará grato se melhorias feitas no programa lhe forem comunicadas assim como cópias destas melhorias lhe sejam enviadas para que sejam igualmente publicadas na mesma página.

Referências

[1] Djairo Guedes de Figueiredo

Análise

Projeto Euclides - SBM

[2] Maurício Galo

Uma introdução ao gnuplot

- UNESP

http://www.4shared.com/file/27829205/4e0d05e0/gnuplot_introducao.html

[3] Gnuplot

`gnuplot` *um programa para fazer gráficos e alguns cálculos*

http://www.gnuplot.info

[4] Praciano-Pereira, T *Programas para Cálculo Numérico* - 2007

http://www.4shared.com/dir/2041165/e14cc331/programas.html

[5] Praciano-Pereira, T.

O programa `fourier.gnuplot`

http://www.edo-metodos.sobralmatematica.org/programas

[6] Praciano-Pereira, T.

A biblioteca `ambiente.h`

http://www.4shared.com/file/12302715/4a6cfaac/ambiente.html

[7] *Wikipedia, a enciclopédia livre na Internet*

http://www.widipedia.org