Introdução ao Matlab

Olga Sato e Paulo Polito *
Departamento de Oceanografia Física, Química e Geológica
Instituto Oceanográfico - Universidade de São Paulo
São Paulo, SP

18 de fevereiro de 2019

Sumário

*olga.sato@usp.br, polito@usp.br

1 Introdução

Este mini-curso é direcionado para quem nunca mexeu no Matlab, portanto se você já tem alguma experiência saiba que pode ser muito básico. O curso é direcionado para Matlab. Como alternativa sugerimos o Octave e o Scilab que são "Open Source", gratuitos e aceitam quase a mesma sintaxe. Há versões destes pacotes gratuitos também para MS-Windows, Linux e Mac OSX.

O curso completo pode ser acessado online no site:

```
http://los.io.usp.br/matcurso/.
```

Materiais complementares ao curso, como a versão PDF deste manual, bem como rotinas prontas e arquivos de dados aqui utilizados, podem ser acessados no seguinte link do site do IO:

```
http://www.io.usp.br/tiki-list_file_gallery.php?galleryId=115,
ou
http://www.io.usp.br/tiki-index.php?page=QuickMatlab.
```

Os comandos do Matlab podem ser digitados no terminal ou escritos num programa através de um editor de programas de sua preferência e executados numa sessão de Matlab. Vamos começar primeiramente digitando alguns comandos no terminal, ou como dizemos, na linha de comando.

Antes de começar com o Matlab, vamos fazer alguns passos preliminares:

- Depois que você estiver logado no Linux, abra um terminal clicando na parte superior esquerda da tela em Aplicações/Acessórios/Terminal. (Este curso será baseado no Linux, porém como dito anteriormente, você poderá utilizar um outro sistema operacional) ;
- Crie um diretório de trabalho. Para manter um mínimo de organização, todos os programas e arquivos de dados serão criados ou copiados para esse diretório. Para isso digite: **mkdir nome** (onde nome é o seu mesmo). Use letras minúsculas, sem acentos e uma palavra só.
- Entre nesse diretório digitando: **cd nome**.

1.1 Como faço para usar o Matlab?

No Windows basta clicar no ícone do Matlab e abre-se a interface gráfica do usuário, conhecido também como Graphical User Interface (GUI). No Linux, também dá para clicar no ícone, mas prefiro usar o terminal.

Para começar e terminar uma sessão de Matlab:

- Para começar o matlab na linha de comando do terminal basta digitar:

  matlab

  Trabalhar com a interface gráfica é absolutamente dispensável, o essencial é apenas a janela de comandos. No Linux, para se livrar da GUI inicie o Matlab assim (é muito mais rápido):

  matlab -nodesktop -nosplash
- Para terminar uma sessão do Matlab digite:

  quit (ou exit)

1.2 Alguns comandos auxiliares

Antes de começar a programar em Matlab propriamente, vamos aprender alguns comandos que podem ser úteis para saber o que está acontecendo com a sua sessão.

Experimente alguns deles na sessão do Matlab e veja o que acontece:

whos	mostra que variáveis estão carregadas na sua sessão.
pwd	comando emprestado do Unix para saber em que diretório você está.
ls ou dir	lista os arquivos do diretório que o Matlab está aberto.
cd diretório	trocar para um outro diretório.
clc	limpar a janela.
version	versão do Matlab.
date	data.
clear	limpar a memória da sessão do Matlab.
help comando	mostra o que algum comando significa. Tente usar com alguns dos comandos acima.

1.3 Utilidades do Matlab

O Matlab é uma ferramenta muito poderosa de programação. Algumas das formas de sua utilização:

1. Fazer contas. O Matlab é ótimo para ser usado como calculadora.
2. Fazer gráficos, incluindo desde simples gráficos xy lineares, como também gráfico em base logarítmica, mapas, de barras, erros, histograma, etc.
3. Pode-se fazer programas interativos onde o usuário digita alguma informação e o programa processa, como também dá para fazer o programa ler arquivos de dados do próprio disco.
4. Dá para programar modelos numéricos complexos. Só vai depender da capacidade de memória e de processamento do seu equipamento.

1.4 Justificativa

Matlab é uma linguagem interpretada ao invés de compilada. Outras linguagens interpretadas são: Perl, IDL, Python, Basic, Shell e é claro, Octave. Algumas compiladas: C, C++, COBOL, Fortran, Pascal e para os dinossauros, RPGII.

O uso de linguagens interpretadas oferece vantagens:

- Sintaxe simples, desestruturada
- Programação na linha de comando
- Uso despreocupado da memória
- Tratamento, visualização e armazenamento de dados

e desvantagens:

- Lentidão, ineficiência
- Exige muita memória
- Encoraja hábitos suínos de programação
- Volatilidade e formatos proprietários

2 Operando o Matlab

O Matlab é muito prático para realizar operações matemáticas simples. Faça algumas contas e veja como funciona. Quer uma ajuda? Vamos lá:

```
 >> 1+1 [Enter]
ans =
     2
```

O Matlab vai sempre colocar qualquer resposta que você não tenha definido com algum nome específico na variável *ans*. Para ver o que já tem na memória, digite:

```
>> whos
  Name      Size    Bytes   Class    Attributes
  ans       1x1         8   double
```

O que isso significa? Que na memória temos uma variável chamada ans, que tem dimensão 1x1 e é um número real de 8 bytes.

Vamos continuar fazendo contas, lembrando que no Matlab todas aquelas propriedades de operações matemáticas são válidas:

- `>> 50*80+7`

- `>> 50*(80+7)`
- `>> (468+990)/(238-37)`
- `>> 245^45`
- `>> 1e10/1e-8`

Os **operadores matemáticos** são:

+	adição
-	subtração
*	multiplicação
/	divisão
$\wedge$	potenciação

Algumas funções já estão pré–definidas no Matlab. Abaixo temos uma lista de algumas dessas funções:

- Trigonométricas: cos, sin, tan, etc., todos em radianos.
- Logarítmicas: log, log10, etc.
- Raiz quadrada: sqrt.
- Exponencial: exp.

Para usar essas funções, tente no terminal alguns exemplos:

- `>> cos(0)`
- `>> sin(60)`
- `>> sin(60*pi/180)`

  Observe que a função trigonométrica deve ser dada em radianos.
- `>> sqrt(144)`
- `>> exp(1)`

2.1 Definindo variáveis

Ao invés de fazer as operações matemáticas diretamente com os números, quando for fazer um programa, é conveniente definir variáveis:

```
>> a=1000;
>> b=2;
>> c=a+b;
```

Para ver o valor de c, digite–o simplesmente. Dê o comando whos para ver quais são as variáveis na sessão. Para saber o valor de qualquer variável, simplesmente digite–o.

O Matlab foi criado para operar com matrizes. Um número é uma matriz de uma linha e uma coluna. Um vetor é uma matriz de uma linha e várias colunas. Uma matriz 3D é uma matriz de várias linhas, colunas e fatias. Estas matrizes podem conter números inteiros, reais, binários, caracteres, etc.. Digite o seguinte:

```
>> x=[1 1.4e-1];y=[1,3.14];
>> y(1,2)

    ans =

    3.1400
>> clear ans
>> whos

     Name Size Bytes Class

     x 1x2 16 double array

     y 1x2 16 double array
```

O que há de relevante nesses exemplos tão simples?

- Podemos encadear comandos usando ponto-e-vírgula.
- O resultado de comandos terminados em ponto-e-vírgula não é mostrado.
- Nos referimos a um elemento da matriz y como y(linha,coluna).
- Se não houver uma variável associada, o valor mostrado é associado à variável ans.
- Podemos criar matrizes separando as colunas com espaços ou vírgulas e separando as linhas com ponto-e-vírgula.
- Espaços em branco e "tabs"não são interpretados.
- Limpamos a variável ans da memória com o comando **clear ans**.
- O comando **whos** descreve as variáveis da memória.

Vamos ver adiante que podemos definir uma variável não somente com números. Faça o exercício a seguir:

```
» clear all
» num=3.1415;
» nome='3.1415';
» meunome='Olga Sato';
» whos
```

- O comando **clear all** limpa tudo.
- A variável **num** contém um número, 3.1415, que serve para fazermos contas.
- A variável **nome** contém os seis caracteres 3.1415 que não servem para fazermos contas.
- De forma similar **meunome** contém quinze caracteres que evidentemente não servem para fazermos contas.
- Observe com o comando **whos** que algumas variáveis são números reais, porém outros são caracteres.

2.2 Contas

Ao fazermos operações matemáticas e lógicas devemos sempre atentar para o fato que estamos lidando com matrizes. Vejamos onde esse fato faz diferença no resultado final:

```
>> a=[1 2 3]
     a =     1     2     3

>> b=[2; 3; 4]
     b =  2
          3
          4

>> c=a+b
     ??? Error using ==> +
     Matrix dimensions must agree.

>> b=[2 3 4]
     b =  2     3     4

>> c=a+b  c =  3     5     7
```

- No primeiro caso criamos as matrizes a e b com três elementos cada, mas a é 1x3 e b é 3x1.
- O comando que criou a tem espaços onde b tem pontos-e-vírgulas).
- Como a soma de matrizes funciona elemento a elemento, o Matlab reclama.
- Na segunda tentativa a e b foram criadas do mesmo modo, as duas são 1x3 e tudo funciona como devia.

Vamos ver como fica quando tentamos fazer a multiplicação de variáveis:

```
>> a=[1 2 3]
     a =     1     2     3

>> b=[2; 3; 4]
     b =  2
          3
          4

>> c=a*b
     c =  20

>> d=a.*b
     ??? Error using ==> .*
     Matrix dimensions must agree.

>> b=[2 3 4]
     b =     2     3     4

>> d=a.*b
     d =     2     6    12
```

- Como no exemplo anterior criamos as matrizes a e b com três elementos cada, uma em pé e a outra deitada.
- A multiplicação das matrizes, "*", resulta em 20 (que pode se visto como matriz unitária ou escalar) pois 1x2 + 2x3 + 3x4 =20.
- No caso seguinte colocamos um ponto antes do asterisco, **.***, indicando para o Matlab que queremos fazer um produto escalar, ou seja, multiplicar elemento por elemento.
- Não dá certo porque as matrizes são de tamanhos diferentes. Recriando b no mesmo formato de a e fazendo o produto escalar obtemos d com o mesmo tamanho de a e b.

Muitas vezes precisaremos transformar matrizes-linha em matrizes-coluna e vice-versa. Isto se chama transposição e se indica com o apóstrofe ' colocado após a matriz que se deseja transpor.

No exemplo abaixo, veja como transformar uma matriz-linha em uma matriz-coluna:

```
>> a=[1 2 3], b=[2; 3; 4]
     a =  1     2     3
     b =  2
          3
          4

>> b=b'
     b =  2     3     4
```

Note que matriz transposta não é matriz inversa.

```
>> m=magic(3)
     m = 8   1   6
         3   5   7
         4   9   2

>> inv(m)
     ans = 0.1472   -0.1444    0.0639
          -0.0611    0.0222    0.1056
          -0.0194    0.1889   -0.1028

>> m'
     ans = 8   3   4
           1   5   9
           6   7   2
```

- O Matlab tem uma série de matrizes especiais pré-fabricadas, uma delas se chama magic e é usada para ilustrar esta diferença.
- A matriz inversa é obtida através da função inv(m) e não tem os mesmos números que a matriz original.
- A matriz transposta é, no caso de uma matriz quadrada, do mesmo tamanho da original, mas os elementos trocam de lugar em relação à diagonal.

Não esqueça: Matlab opera com matrizes. Portanto há vários tipos de contas que dependem do tamanho ou formato das matrizes (arrays) com as quais podemos nos confundir. É muito mais comum fazermos multiplicação, divisão e potenciação elemento-a-elemento, como fazemos em C, FORTRAN etc..

Um vetor é uma matriz unidimensional, ou seja, uma matriz-coluna contendo uma só coluna, ou uma matriz-linha com uma linha só. O elemento de uma matriz é dado por um par de índices que designa sua posição relativa. Por exemplo, o primeiro elemento de uma matriz a é o elemento da primeira linha e da primeira coluna, ou seja, a(1, 1); o segundo elemento da primeira linha, a(1, 2); o primeiro da segunda linha, a(2, 1); e assim por diante.

Defina uma matriz qualquer de dimensões 3×3. Por exemplo:

```
>> a = [1 2 3; 4 5 6; 7 8 9]
     a = 1   2   3
         4   5   6
         7   8   9

>> a(:, 1)

>> a(2, :)
```

- Podemos tomar os elementos que estão em qualquer linha ou coluna usando o símbolo : na posição correta.

Exercícios: Faça os exercícios sugeridos e tire as suas próprias conclusões.

```
>> a=[1 2 3 4 5 6];
```

```
>> b=[9 8 7 6 5 4];
>> a(1)
>> b(3)
>> c=a+b
>> a+10
>> d=b-a
>> e=(a+b)/10
>> e(3)=0
>> a1 = [a a]
>> a2 = [a;a]
```

O que achou? Dá para fazer muitas manipulações com matrizes. Abaixo estão mais algumas funções que podem ser úteis na hora de manipular uma matriz. Lembre, em Matlab, tudo é matriz.

Algumas **funções estatísticas** aplicadas à uma função A:

mean(A)	média dos termos da matriz
sum(A)	soma dos elementos da matriz
std(A)	desvio padrão
cumsum(A)	soma cumulativa dos elementos de A
median(A)	mediana
min(A)	o valor mínimo de A
max(A)	valor máximo de A
var(A)	variância

3 O primeiro programa

Até agora utilizamos o terminal para fazer algumas operações simples. Mas a ideia de se utilizar um programa é que podemos colocar vários comandos juntos, ordenados de uma forma lógica para se executar uma tarefa. Ao invés de repetirmos longas e tediosas operações lógico-matemáticas podemos escrever programas, sub-programas e funções, aproveitando melhor o nosso limitado tempo de vida.

No escopo deste curso básico vamos ver apenas os tipos mais comuns de programas do Matlab, os chamados **m-files**. Um m-file é como qualquer outro código fonte, um arquivo texto contendo uma sequência de comandos. M-files são salvos surpreendentemente com a extensão ".m".

Há centenas de funções no Matlab e explicar cada uma delas seria uma colossal perda de tempo. Para saber que funções existem na sua máquina tente **help** na linha de comando e se informe. Para saber rapidamente o que uma a função faz, basta pedir um "help". Por exemplo, para saber o que função "repmat"faz, digite **help repmat**. Além de inglês aprende-se muita álgebra lendo esses manuais.

As linguagens de programação mais comuns tem os seguintes elementos básicos em comum:

- Condicionais, algo do tipo "se isto então aquilo"(if then else) ou "caso isto faça aquilo"(case switch)
- Loops, algo do tipo "Para n de tanto a tanto faça isso"(for, do) ou "enquanto n for assim repita isso"(while)
- Leitura e gravação de arquivos, algo do tipo "abrir ler/gravar fechar"(open read/write close)

3.1 Como escrever e rodar um m-file

Vamos escrever um programa em Matlab bem simples, como no exemplo abaixo. Primeiramente abra um editor de texto de sua preferência. Utilize um editor de texto que tenha a possibilidade de salvar o arquivo sem formatação nenhuma, ou seja um arquivo texto ou simplesmente ASCII. No linux você tem as seguintes opções: gedit, emacs e o vi. Escolha um deles e edite as linhas abaixo. Salve com um nome sugestivo como: ex01.m.

```
% Este programa é parte do curso de Introdução à Matlab:
% Observe que os comentários são precedidos pelo símbolo
% Fazer um loop de 1 a 20 de 1 em 1 e imprimir esse valor no terminal.
for i=1:1:20
disp(i)
end
```

No terminal do Matlab digite: ex01. Pronto, aparecerão os números que você mandou escrever. Faça imprimir os valores de 1 até 50, de 5 em 5.

3.2 Como salvar seus resultados e recuperá-los

Variáveis podem ser carregadas numa sessão do Matlab como também criadas e salvas para serem utilizadas posteriormente. O comandos para fazer essa tarefa são:

load nomearquivo	carregar um arquivo de dados
save nomearquivo	salvar um arquivo com todos os dados que estão no espaço de trabalho em formato Matlab: .mat
save nomearquivo A B	salvar um arquivo de dados contendo as variáveis A e B
save nomearquivo -ascii	salvar um arquivo de dados em formato ascii

No exemplo abaixo podemos ver como se salva um arquivo:

```
>> x=-pi:.1:pi;

>> y=-5:.1:5;

>> [xx,yy]=meshgrid(x,y);

>> zz=sin(xx).*cos(yy);

>> save arquivox.mat zz
```

- O procedimento cria dois vetores, x e y. Baseado neles o comando meshgrid cria uma grade (ou um campo) formada por duas matrizes, xx e yy, a partir das quais se calcula zz.
- O comando **save** cria o arquivo arquivox.mat contendo a variável zz.
- O arquivo arquivox.mat está em formato ".mat"que é proprietário, ou seja, é difícil de ler com outros programas. Com a opção -ascii criamos um arquivo em formato ASCII, que pode ser lido por outros programas.

3.3 Carregar arquivo e definir as variáveis

Os arquivos para serem carregados no Matlab devem estar organizados em colunas, todos os valores devem estar preenchidos e devem ter somente caracteres numéricos.

- Um arquivo chamado **sta_data.dat** contendo os dados de CTD de uma estação hidrográfica foi preparado para este exercício.
- Baixem o arquivo do site diretamente para o diretório que você está, ou seja, naquele que você criou no começo deste curso.
- Abra o arquivo de dados usando o editor de texto e veja como os dados estão organizados.
- Vamos criar um m-file que lerá esse arquivo de dados. Comece um novo arquivo no editor de textos com as seguintes linhas de comando:

```
load sta_data.dat
z=sta_data(:,1);
tem_a=sta_data(:,2);
sal_a=sta_data(:,3);
tem_b=sta_data(:,4);
sal_b=sta_data(:,5);
save sta_data
```

- Salve o programa como **ler_dados.m** ou algum outro nome sugestivo sobre o que o programa faz.
- O que faltou fazer nesse programa? Comentários. Edite o programa novamente e inclua comentários explicando o que cada linha ou bloco de linhas faz. Salve–o.
- Execute o programa. Na sessão do Matlab digite **ler_dados**. Não precisa colocar o ".m".

Agora você tem um arquivo chamado **sta_data.mat** que tem os dados da estação. Vamos trabalhar esses dados. A primeira coisa que gostamos de fazer é ver os dados. Para isso, vamos plotá–los.

4 Fazendo gráficos com o Matlab

Fazer gráficos no Matlab é muito simples. Defina primeiro a variável da abcissa e a da ordenada, faça o gráfico xy usando o comando **plot(x,y)** e complete com os devidos textos para explicar o que está plotando. Siga o roteiro abaixo para fazer uma sequência de quatro gráficos exemplificando o uso do comando plot. A maneira mais fácil de fazer esse exercício é copiar as linhas de comando num m-file (ou usar o m-file plot_demo.m que já está no site) e executá–lo no Matlab.

4.1 Gráficos xy

O primeiro exemplo ensina a fazer um gráfico de uma função senoidal.

```
%---------------------------------
close all;clear all
%---------------------------------
% cria eixo das abcissas
x0=1:.1:10;
% cria y0=f(x0)
y0=sin(x0+pi/4);
% faz um grafico simples
plot(x0,y0);
% titulo e identificacao dos eixos
title('Um seno senoidal')
xlabel('nome da abcissa')
ylabel('nome da ordenada')
pause
```

- Nesta primeira figura, o comando plot gera um gráfico "x-y"bem simples, no qual adicionamos um título para o gráfico e um para cada eixo.
- Sempre é bom manter o controle do espaço de trabalho, então utilize o comando **clear ou clear all** para limpar a memória e **close ou close all** para fechar todas as janelas de figuras.
- O comando **pause** causa uma parada no programa por até que a tecla "enter"seja pressionada.

O segundo gráfico mostra como sobrepor duas curvas e definir diferentes tipos de linhas.

```
%---------------------------------
% cria mais uma funcao senoidal
y1=sin(x0+pi/2);
figure(2)
% faz um grafico com duas curvas
plot(x0,y0,'.r',x0,y1,'--b');
% titulo e identificacao dos eixos
title('Dois senos senoidais')
xlabel('Grana (R$)')
ylabel('Tempo (anos)')
% Texto explicativo
text(5.4,-0.2,'x Aqui');
pause
```

- Na segunda figura, o comando **plot** aceita vários conjuntos de argumentos para gerar várias linhas com um só comando. Os "strings"'.r' e '–b' controlam as cores e os símbolos de cada curva;
- **Aviso**: O comando **plot** demanda que as matrizes a plotar tenham o mesmo número de colunas.

Terceiro, sobrepor curvas:

```
%---------------------------------
% ilustra o uso do hold on/off
figure(3)
% faz um grafico com duas curvas
plot(x0,y0,'r');
hold on
plot(x0(25:75),y1(25:75),'+b');
% titulo e identificacao dos eixos
pause
```

- Podemos combinar matrizes de comprimentos diferentes usando **hold on e hold off**.
- Um texto foi adicionado ao gráfico através do comando **text**. Note que alguns símbolos do LaTeX (e.g. ^2, _i, alpha, beta) podem ser utilizados no Matlab.

Quarto: quando tem mais que uma linhas é melhor usar legenda.

```
%---------------------------------
% cria mais tres funcoes senoidais
% em uma unica matriz
y3=[sin(x0);sin(x0+pi/16);...
sin(x0+pi/8)];
figure(4)
% faz um grafico com duas curvas
plot(x0,y3)
% titulo e identificacao dos eixos
title('Tres senos senoidais')
xlabel('Grana (R$)')
ylabel('Tempo (anos)')
legend('uno','ni','trois')
pause
```

- Ao combinar várias curvas no gráfico da quarta figura, geramos certa confusão. Podemos adicionar uma legenda explicativa com o comando legend.
- Os comandos loglog,semilogx e semilogy funcionam como o plot, mas geram eixos logarítmicos em vez de lineares.
- Como exercício crie gráficos de y0.^2, y1.^2 e y3.^2 com eixos logarítmicos (x,y ou os dois).

Colocar todos os gráficos numa página só:

```
%---------------------------------
close all
%---------------------------------
% Estas quatro figuras podem ser
% combinadas numa só
% através do comando subplot:
subplot(2,2,1),loglog(x0,y0.^2);
title(' Aha'', Log-Log!')
subplot(2,2,2),
plot(x0,y0,'.r',x0,y1,'--b');
subplot(2,2,3),plot(x0,y0,'r');
hold on
plot(x0(25:75),y1(25:75),'+b');
hold off
subplot(2,2,4),plot(x0,y3)
legend('uno','ni','trois')
```

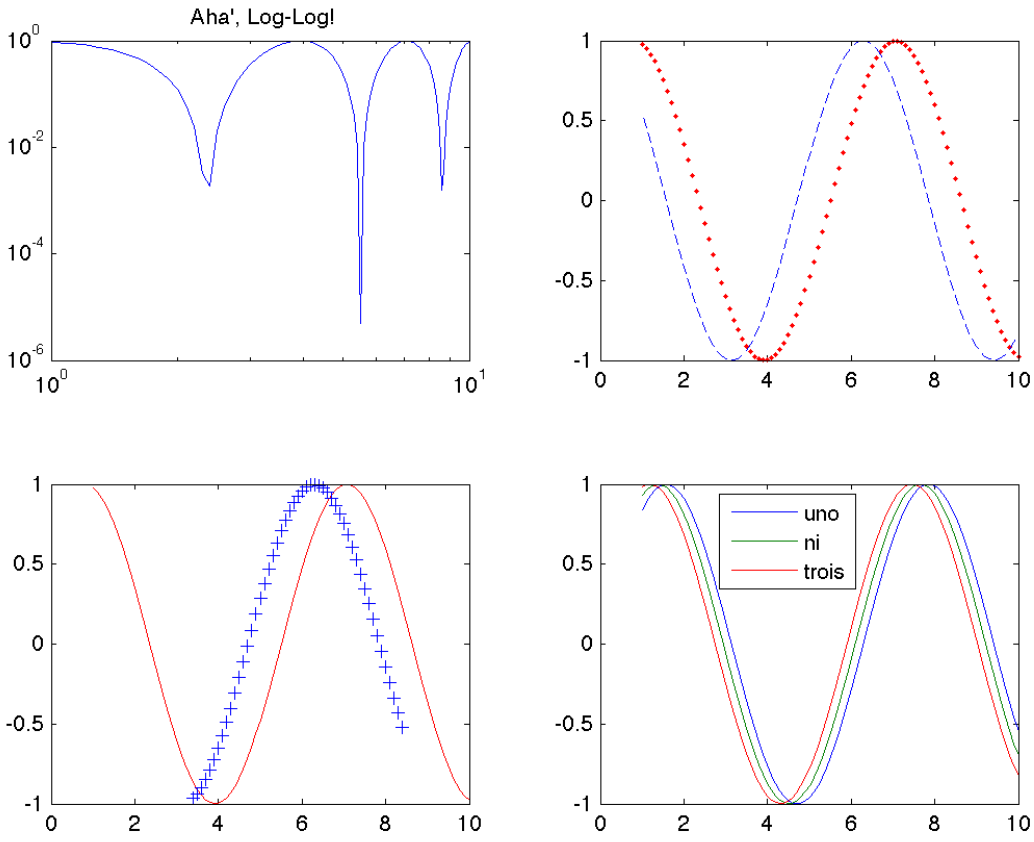

Figura 1: Demonstração de como gerar vários gráficos numa só página.

4.2 Perfil de variáveis oceanográficas

O Matlab permite que você faça simulações com dados sintéticos como no exercício anterior. Mas em oceanografia, o que queremos é usá–lo para estudar os processos oceânicos. Então vamos ver como podemos utilizá–lo na prática.

Veja o que tem no seu espaço de trabalho no Matlab. O comando é **whos**. Lembra? Vamos limpá–lo para a próxima tarefa: **clear all; close all**.

Para esse exemplo, vamos utilizar o arquivo **sta_data.mat** que contem os dados dos perfis de temperatura e salinidade das estações A e B.

- Carregue o arquivo de dados:

```
>> load sta_data
```

- Plote o perfil de temperatura da estação A. No oceano, as profundidades são negativas pois o eixo-z aponta para cima:

```
>> plot(temp_a,-z)
```

- Plote com pontos ao invés de linha contínua:

```
>> plot(temp_a,-z,'.')
```

- Se quiser linha contínua com pontos:

```
>> plot(temp_a,-z,temp_a,-z,'.')
```

- Um gráfico útil tem que ter legendas:

```
>> xlabel('Temperatura (^oC)')
>> ylabel('Profundidade (m)')
>> title('Estação A')
```

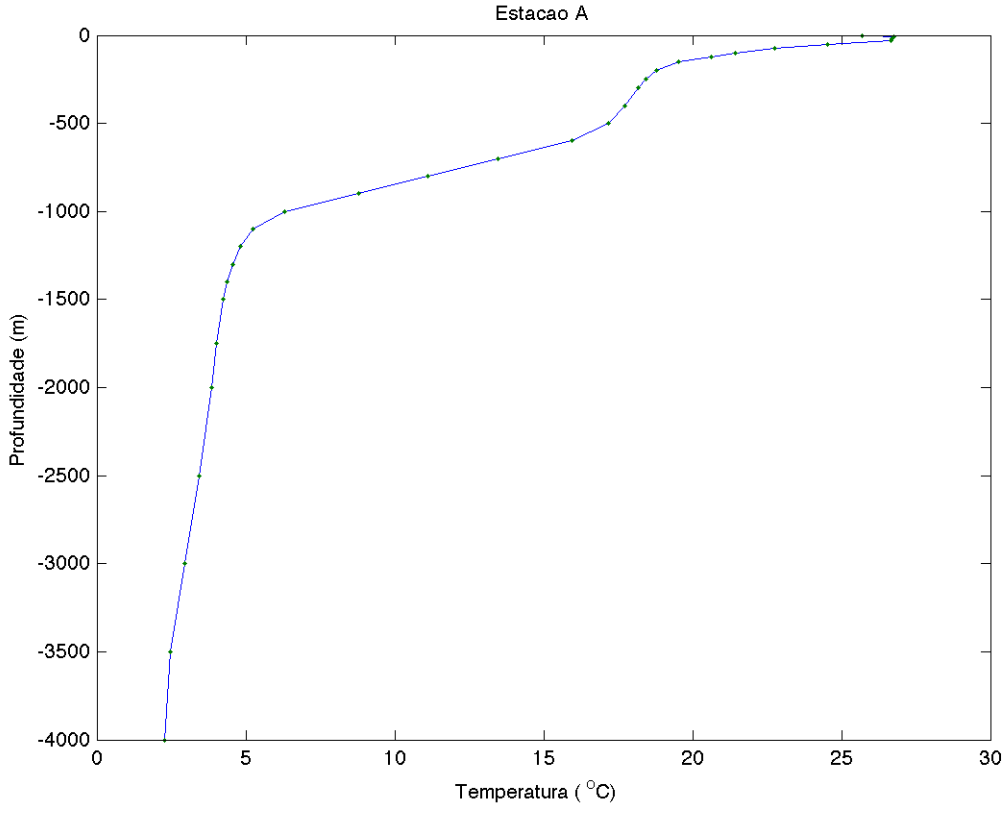

Figura 2: Perfil de temperatura (°C) da estação A.

Aqui estão alguns comandos do Matlab que podem ser utilizados para fazer gráficos xy.

plot(x,y)	gráficos em coordenadas cartesianas.
xlabel('texto')	adiciona texto no eixo-x.
ylabel('texto')	adiciona texto no eixo-y.
title('texto')	adiciona texto acima do gráfico.
hold on/off	liga/desliga a sobreposição de gráficos.
subplot(m,n,p)	cria m×n gráficos numa só página, e o presente fica na posição p.
figure	abre uma nova janela para gráfico.
close ou close all	fecha uma ou todas as janelas de gráficos.
legend	faz legenda das curvas plotadas.
axis	controla a escala e a aparência do eixo. Tem várias opções.
grid	desenha a grade do gráfico.

O comando **plot** permite utilizar vários tipos de símbolos e linhas. Digite **help plot** e veja as possibilidades:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	–	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

O comando **subplot(m,n,p)** permite colocar vários gráficos numa só página. Por exemplo, para fazer quatro gráficos, com dois gráficos na parte superior e dois na inferior, devemos definir m=2 e n=2.

```
>> subplot(2,2,1);plot(x,y)

>> subplot(2,2,2);plot(x,y)
```

- Divide a página em 4 partes e prepara para receber um gráfico na parte superior esquerda,
- Prepara para receber um gráfico na parte superior direita, e assim por diante.

Utilizem todos os comandos aprendidos até agora e escrevam um programa que consiga produzir o seguinte gráfico com os dados do arquivo sta_data.dat:

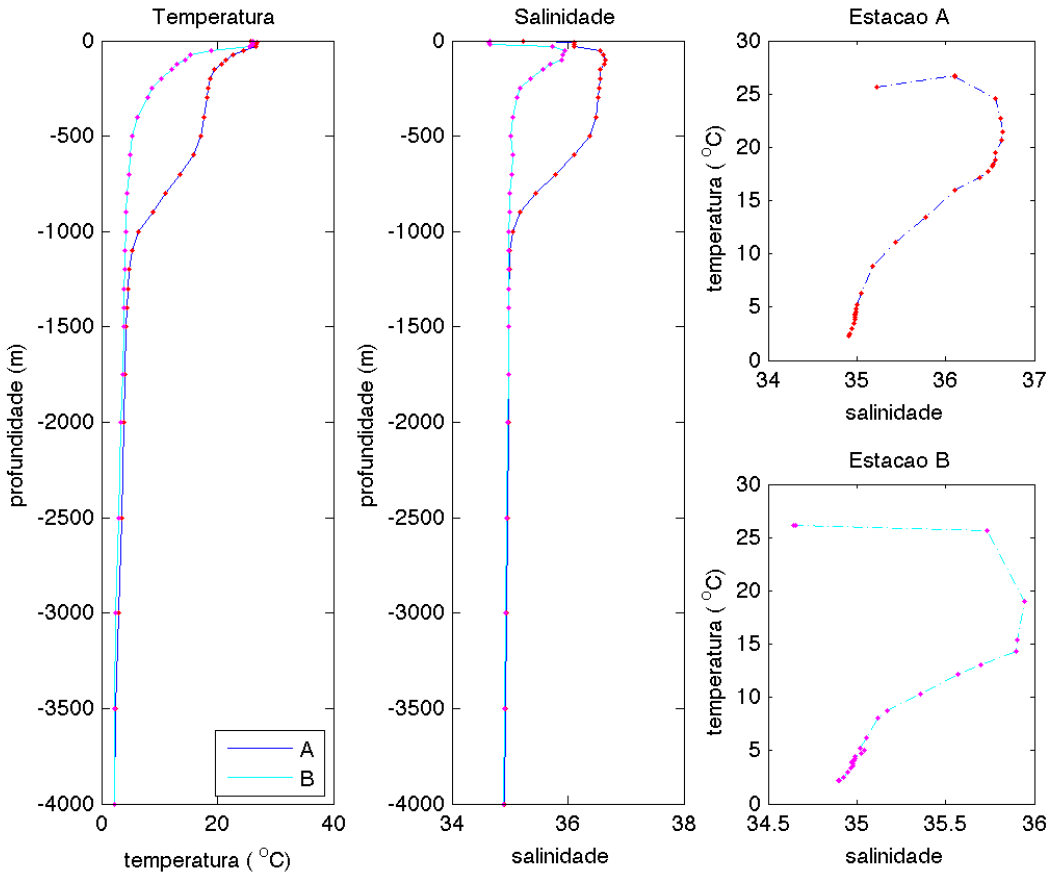

Figura 3: Perfis de temperatura (°C) e salinidade e os diagramas TS das estações A e B.

4.3 Séries temporais

Comumente dados medidos num determinado ponto no espaço por um certo período geram uma série temporal. A variação da altura do nível médio do mar medido por marégrafos é um exemplo típico. Considere no exercício abaixo a série maregráfica obtida em Fortaleza. O programa plot_mare.m e o arquivo de dados maré_fortz.dat devem ser baixados do site.

```
% Programa plot_mare.m
% ---------------------------------
% Exemplo de análise harmônica de marés
clear all; close all
% Este programa ira carregar os dados de mare, calcular a
% data juliana e plotar a altura da mare em funcao do tempo.
% ---------------------------------
% Este exemplo sera baseado nos dados de mare de Fortaleza.
```

```
% Fonte: goosbrasil.org, no link do GLOSS-Brasil.
load mare_fortz.dat
data=mare_fortz;
yy=data(:,1);
mm=data(:,2);
dd=data(:,3);
hh=data(:,4);
z=data(:,5);
% ---------------------------------
% Calcular a data juliana continua
t=datenum(yy,mm,dd,hh,0,0);
% ---------------------------------
% fazer o grafico da altura em funcao do tempo
subplot(2,1,1); plot(t,z)
grid;axis('tight')
ylim([0 3500])
datetick('x',28,'keepticks')
xlabel('tempo (dias)')
ylabel('Altura (mm)')
title('Dados de Mare em Fortaleza')
disp(['variancia do sinal original (z) = ',num2str(var(z),'%7.2f')])
```

Uma dificuldade em se plotar uma série temporal é que vetor tempo deve ser contínuo. Os vetores que representam o tempo são o ano, mês e o dia. Se for fazer um gráfico com esses valores, o eixo do tempo fica confuso pois o valor do ano muda, mas os meses e os dias são sempre os mesmos. Devemos então tentar construir um vetor contínuo de tempo com esses valores.

No Matlab tem uma função que faz isso: **datenum**. No programa acima ela é usada para determinar o vetor t. Digite t no terminal e veja o que aparece. Esses valores representam a data serial desde uma data arbitrariamente escolhida: 1 de Janeiro do ano 0. Se fizer o gráfico em função de t, ninguém vai entender essa data, é um número enorme. Para converter essa data para um valor que os humanos possam entender, pelo menos os da cultura ocidental, após plotar o gráfico usar o comando **datetick**. Dê um **help datetick** para saber quais são as opções para expressar o tempo.

Continuando o nosso exercício, vamos remover a média da maré para determinarmos a anomalia da altura.

```
% ----------------------------------------
% Programa remove_media.m
% Estimar a altura media:
mz=mean(z);
subplot(2,1,2); plot(t,z-mz,'g')
grid;axis('tight')
ylim([-1600 1700])
datetick('x',28,'keepticks')
xlabel('tempo (dias)')
ylabel('Altura (mm)')
title('Anomalia da altura da mare em Fortaleza')
% ----------------------------------------
% Remover a media do sinal da mare
az=z-mz;
disp(['variancia da anomalia (az) = ',num2str(var(z),'%7.2f')])
```

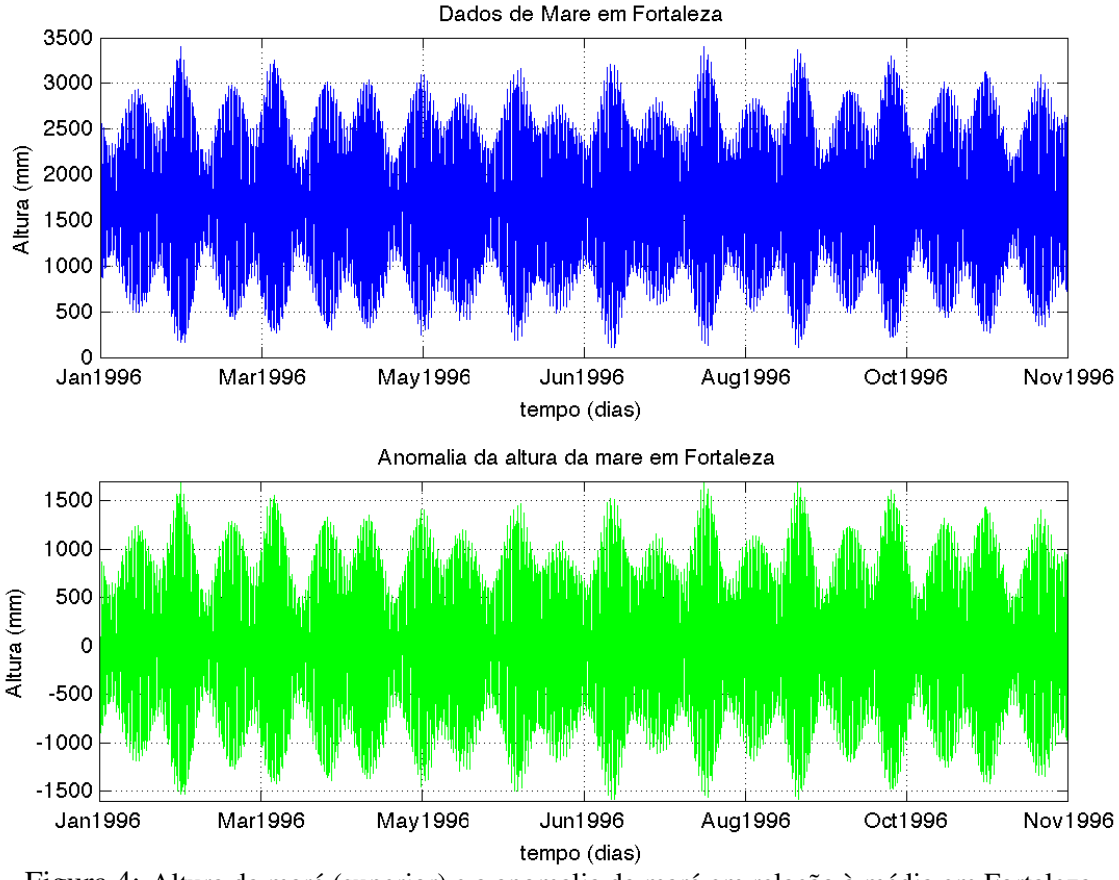

Figura 4: Altura da maré (superior) e a anomalia da maré em relação à média em Fortaleza.

4.4 Superfícies bidimensionais

Gráficos bidimensionais são utilizados para fazer mapas de alguma variável em função de sua posição no espaço, ou seja, z=f(x,y). Essa variável pode ser, por exemplo, alguma propriedade medida através de dados de satélites, produzida por modelos numéricos ou medidas feitas *in situ*. Vamos utilizar dados de concentração de clorofila medida pelo satélite MODIS. Esses dados são distribuídos pelo site: `{http://oceancolor.gsfc.nasa.gov/}`. (No site do curso rápido de Matlab temos um exemplo de como ler dados de altímetro. Experimente.)

O procedimento geral para produzir um mapa utilizando o Matlab é:

- Carregar os dados no espaço de trabalho;
- Reformatar os dados devidamente;
- Gerar o gráfico.

Aqui estão mais alguns comandos para fazer gráficos:

imagesc	gera uma imagem.
contour	gera mapa com contornos.
pcolor	mapa bidimensional.
shading	forma de preencher as cores.
colorbar	barra de cores com os valores do gráfico.
gtext	colocar um texto sobre o gráfico usando o mouse.
ginput	obter os valores da posição de um gráfico usando o mouse.
print	salvar a figura como uma imagem em diversos formatos (jpg, png, eps, etc.)

4.4.1 Mapa de superfície

Os dados de concentração de clorofila, em $mg\ m^{-3}$, estão no arquivo clorofila.asc, que deve ser baixado diretamente para o seu diretório de trabalho. Crie um programa m-file utilizando o editor de texto com o seguinte conteudo e salve como ler_modis.m. (Para facilitar, o programa pronto pode ser baixado do site do curso.)

```
clear all; close all
% ----------------------------
% Carregar e ler as variáveis
load clorofila.asc
lo=clorofila(:,1);
la=clorofila(:,2);
clo=clorofila(:,3);
% ----------------------------
% Matriz com 420 pontos de latitude
% e 420 pontos de longitude.
lo=reshape(lo,420,420);
la=reshape(la,420,420);
clo=reshape(clo,420,420);
% ----------------------------
% Para plotar em função da latitude e
% longitude, definir como vetores;
la=la(1,:)';
lo=lo(:,1)';
% ----------------------------
% Transposta da matriz clorofila.
clo=clo';
% ----------------------------
% Fazer o mapa da clorofila
imagesc(lo,la,log10(clo));axis('xy');
colorbar;
gtext('log_{10}(mg/m^3)')
xlabel('longitude')
ylabel('latitude')
print -dpng cloro
```

- De antemão devemos saber o tamanho dos mapas e a resolução, por isso podemos colocar os vetores x e y no tamanho desejado.
- Utilizando o comando **load** na forma de função, transferimos o conteúdo do arquivo para as variáveis lo, lá e clo. clo é unidimensional (vetor), para fazermos dele um mapa precisamos torná-lo bidimensional (matriz) com o comando **reshape**;
- É muito comum termos os dados guardados em ordem diversa daquela adotada pelo Matlab. Se você plotar clo logo após o reshape, notará que os continentes estão "um pouco estranhos". O operador de transposição **'** resolve esse probleminha.
- Por fim, o comando contour cria o gráfico de superficie com o comando **imagesc**. Observe que para a clorofila devemos plotar $\log_{10}$(clo) pois seus valores variam muito entre as diversas regioes no oceano.
- O imagesc inverte a figura, por isso devemos usar o **axis('xy')** para consertar isso.
- O comando **gtext** permite colocar qualquer texto sobre a figura. Use–a para colocar as unidades na barra de cores.
- Nunca esqueça de colocar o que está sendo representado nos eixos x e y.

Execute o programa ler_modis.m e veja o resultado:

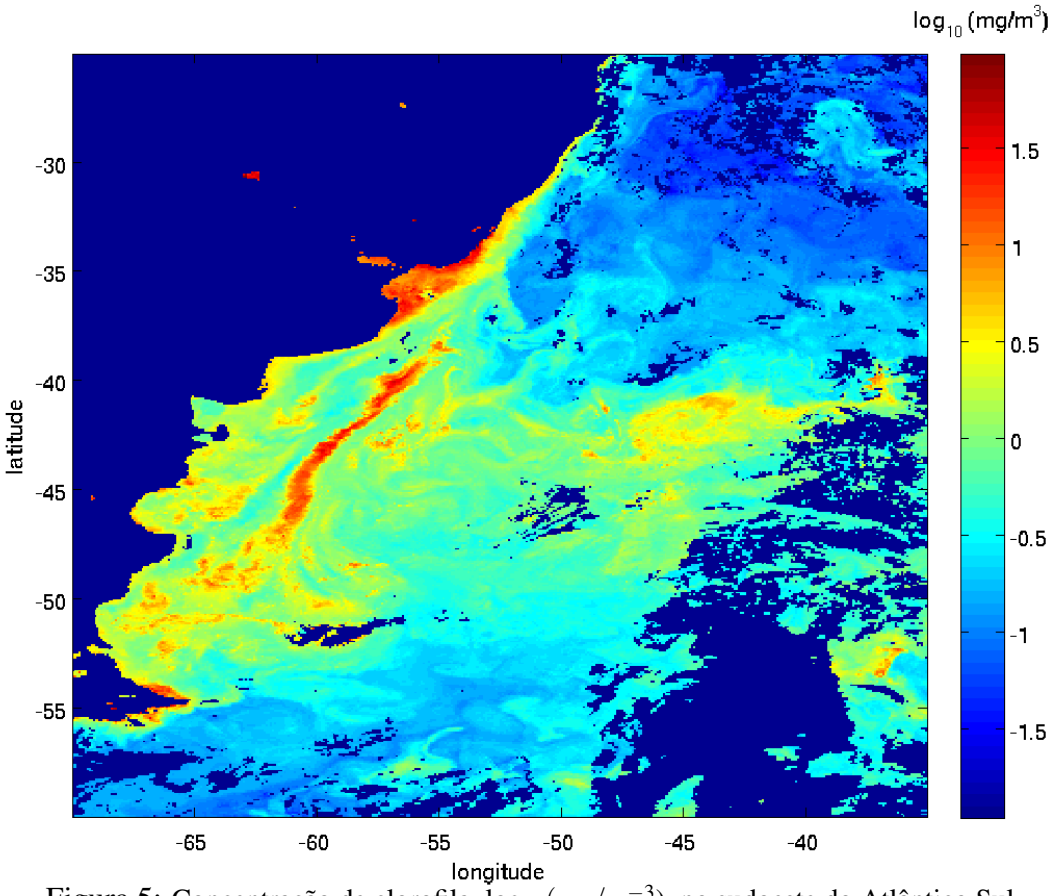

Figura 5: Concentração de clorofila, $\log_{10}(mg/m^{-3})$, no sudoeste do Atlântico Sul.

4.4.2 Seção vertical

Uma outra forma de gráfico bidimensional é fazer uma seção vertical de alguma propriedade da água do mar, por exemplo, nitrato. A média anual de nitrato na longitude de 30°W for obtida a partir da climatologia do World Ocean Database. Essa seção é obtida a partir da média de muitos dados coletados ao longo de décadas de missões oceanográficas. Esses dados são tratados rigorosamente e interpolados para gerar a média em resolucao anual, sazonal e mensal.

Os dados de nitrato estão no arquivo nit_sec30S.mat. Para facilitar, os dados já estão no format .mat.

```
% Fazer a seção vertical de nitrato
load nit_30S
pcolor(lat,-z,nit_30S)
shading interp
c=colorbar;
h=get(c,'title');
set(h,'string','\mu mol')
xlabel('latitude')
ylabel('Profundidade (m)')
title('Nitrato')
print -dpng sec_nit30S
```

- O comando **pcolor** permite também fazer uma imagem colorida. Apresenda três formas de fazer o sombreamento. O **shading interp** proporciona cores contínuas.
- As unidades na barra de cores foi feito utilizando–se conceito de **handles** onde se permite mudar diversos parâmetros dentro de um gráfico.

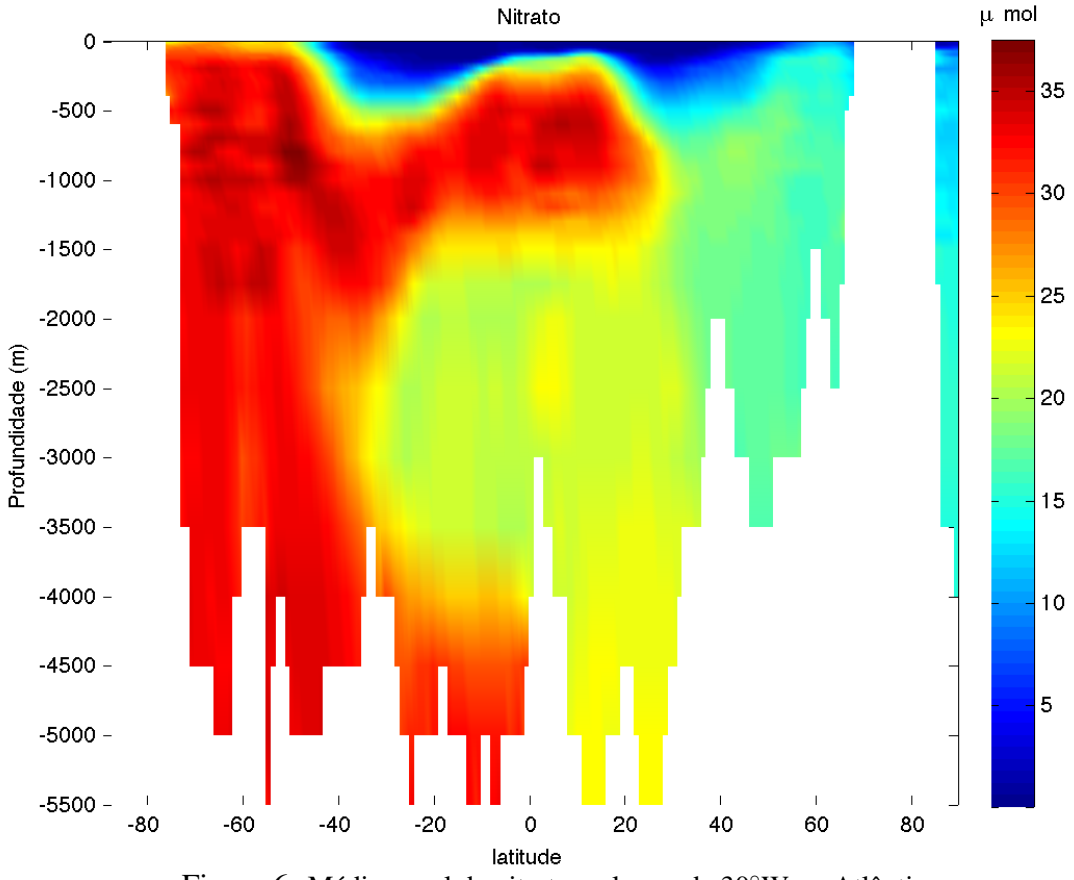

Figura 6: Média anual de nitrato ao longo de 30°W no Atlântico.

5 Funções, interações e interpolações

À medida que já criamos alguns m-files podemos avançar mais um pouco com as técnicas de programação. A idéia principal desta seção é montar um programa com um loop e parar a execução

dentro do loop para verificar o conteúdo de algumas variáveis, interagindo de várias formas com o programa. Numa segunda instância vamos criar funções para simplificar a leitura e a execução do programa.

Vamos precisar também de alguns arquivos de dados: ARGO_*.m.

5.1 Interagindo com o programa

Digite o código abaixo no editor de textos e salve para executá-lo no Matlab.

```
% Este programa voces comentam
clear all;close all
for i=43:47
si=num2str(i,'%3.3i');
file=['ARGO_' si '.m'];
eval(['run('' ' file(1:8) ' '')';]);
if isempty(P),
display(['MSG: ' file ' vazio'])
end
disp('Para avancar digite return');
keyboard
sn = input('Plota TS (s/n)?','s')
if sn=='s',
plot(S,T,'.');
end
sn = input('Cria fig jpeg (s/n)?','s')
if sn=='s',
figu=[file(1:9) 'jpg'];
eval(['print -djpeg ' figu])
end
end
```

- Ao executar o m-file ao lado ele emite uma mensagem e para. O comando que o faz parar é o **keyboard** que nos permite fazer quaisquer operações enquanto o programa fica parado até que digitemos return (o "enter"está subentendido).
- Portanto aproveite e verifique o conteúdo das variáveis para entender o que este programa fez até aqui. Verifique também o conteúdo dos arquivos ARGO_0*.m usando o comando **type**. Agora que você entendeu tudo, insira comentários antes de cada comando explicando o que eles fizeram e para quê.
- No comando **num2str** transformamos os números 43,44,... nos strings '043','044',... que usamos logo abaixo para montar o nome dos arquivos que por sua vez entram no comando eval.
- O comando **eval**('bla bla bla') é super útil pois executa o conteúdo de uma variável. É como se você estivesse digitando o 'bla bla bla' que vem entre os parênteses. Isso nos permite montar comandos contendo nomes sequenciais de arquivos e variáveis dentro de uma variável string para depois executá-los. Isto é crucial para a automação de processos.

Muito bem, agora vamos desmembrar o programa acima em um programa principal e uma função (funz.m). Este é o principal, chame–o de multi_TS.m:

```
clear all;close all
for i=43:47
[P,S,T,file]=funz(i);
sn=input('Plota curva TS (s/n)?','s')
if sn=='s',
plot(S,T,'.');
end
sn=input('Cria figura jpeg (s/n)? ','s')
if sn=='s',
figu=[file(1:9) 'jpg'];
eval(['print -djpeg ' figu])
end
end
```

Esta é a função que deve ser salvo como funz.m:

```
function [P,S,T,arq]=funz(k);
% A funcao [P,S,T,arq]=funz(k)
% Executa arquivos do tipo ARGO_[i].m
% e retorna a Pressão, Salinidade,
% Temperatura e o nome do arquivo
% correspondentes a um único perfil
  si=num2str(k,'%3.3i');
  arq=['ARGO_' si '.m'];
  disp(arq);
  eval(['run(''' arq(1:8) ''')';]);
  if isempty(P),
    display(['MSG765: ' arq ' vazio']);
  end
  return
```

Estes dois m-files combinados tem o mesmo efeito que o anterior. Salve o de cima como multi_TS.m e o de baixo como funz.m. Na linha de comando digite 'multi_TS'. Funciona igualzinho antes, exceto pelas paradas via keyboard certo?

Em relação à subrotina, note que:

- O nome das variáveis do programa principal nada tem a ver com o nome delas dentro da subrotina. No caso, o 'i' do programa é o 'k' da subrotina, o 'file' do programa é o 'arq' da subrotina.
- A quantidade, a ordem e o tipo de variáveis passadas para e recebidas da subrotina são importantes.
- Os arquivos ARGO_*.m são na realidade subprogramas que executam uma tarefa bem bobinha, apenas carregam as variáveis P, S, e T. Às vezes, para clarificar um programa muito complexo, pode ser conveniente dividir o programa principal em módulos ou subprogramas e administrá-los separadamente.
- Subprogramas são m-file como quaisquer outros e as variáveis são comuns ao programa principal e aos subprogramas.
- Os comentários incluídos no cabeçalho da subrotina funz são aqueles que aparecem quando é dado o comando help funz e devem mostrar a utilização da mesma e o significado de cada uma das variáveis de entrada e de saída. Verifique!
- Desta forma o programa principal fica mais claro, mais fácil de ler e a função funz pode ser usada por outros programas, desde que esteja no seu "path".
- O path é o conjunto de diretórios que o Matlab usa para procurar por programas e dados além do diretório corrente. O comando cd muda de diretório e o arquivo startup.m pode ser usado para definir o seu path pessoal.

No final você deve ter obtido algo do tipo:

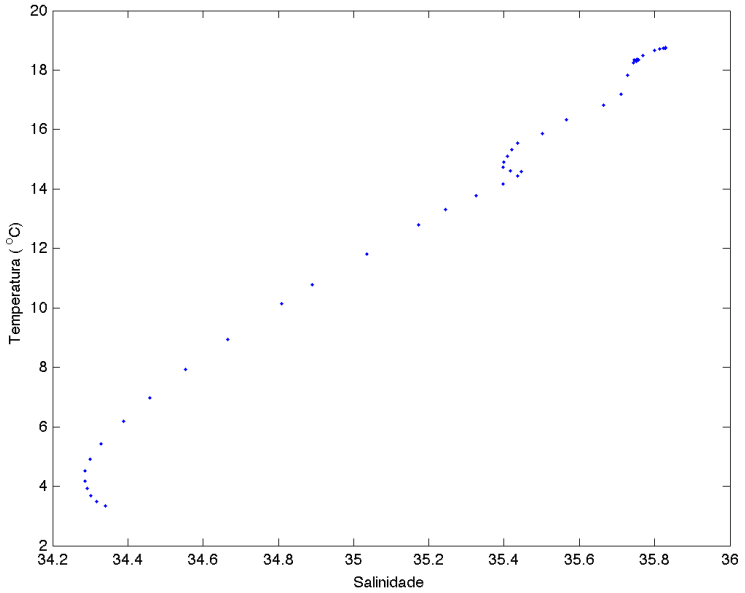

Figura 7: Diagrama TS da estação Argo 46.

5.2 Interpolação de dados

Uma necessidade comumente recorrente quando estamos trabalhando com dados, especialmente os coletados *in situ*, é a da interpolação. Precisamos interpolar os dados pois as vezes ocorre uma falha na coleta, a resolução que o dado foi coletado não foi suficiente ou necessita de dados em outros pontos. Se procurar na literatura, você vai encontrar inúmeros métodos de interpolação. Neste curso vamos abordar a interpolação linear em uma dimensão.

Tomemos como exemplo o perfil coletado pelo Argo 47 do exercício anterior. Execute o programa ARGO_047.m e plote a temperatura ou salinidade em função da pressão. Observe que há uma falha por volta da pressão de 450dbar. Vamos fazer uma interpolação linear e determinar o ponto que está faltando. Faça os seguintes comandos na própria linha de comando.

```
>> ARGO_047
>> plot(T,-P,'.')
>> pp=450;
>> ti=interp1(P,T,pp)
>> plot(T,-P,'.b',ti,-pp,'.r')
>> si=interp1(P,S,pp)
```

- O comando **interp1** pode ser utilizado para fazer a interpolação do dado que está faltando. A sintaxe do comando é **interp1(x,y,xi,método)**, onde x e y são os vetores unidimensionais com os dados originais e xi é o ponto em x que se quer interpolar.
- Seis métodos podem ser utilizados para interpolar. Se não especificar nenhum método, o Matlab utiliza o método linear. Para alguns casos em oceanografia, especificamente o de interpolação de T e S, o método linear é o melhor pois interfe menos na relação TS.

Observe que as medidas dos perfis TS não estão disponíveis em profundidades conhecidas em oceanografia como **profundidades padrão**. Como o próprio nome diz, são profundidades adotadas como padrão. Sua adoção facilita na hora de tratar várias estações ao mesmo tempo, por exemplo quando queremos determinar a velocidade geostrófica. No próximo exercício, vamos definir as profundidades padrão e interpolar o perfil TS obtido pelo Argo 47. Baixe o programa interp_TS.m e verifique o que cada linha faz. Coloque os comentários.

```
clear all; close all;
% Carregar os dados do Argo 47
ARGO_047
pp=[0:10:30 50:25:150 200:50:300 400:100:1500 1750]';
ti=interp1(P,T,pp);
subplot(1,2,1);plot(T,-P,'.b',ti,-pp,'.r')
xlabel('Temperatura (^oC)')
ylabel('Pressão (dbar)')
si=interp1(P,S,pp);
subplot(1,3,2);plot(T,-P,'.b',ti,-pp,'.r')
xlabel('Salinidade')
ylabel('Pressão (dbar)')
figure
plot(S,T,'.',si,ti,'r.')
xlabel('Salinidade')
ylabel('Temperatura (^oC)')
legend('original','interpolado')
```

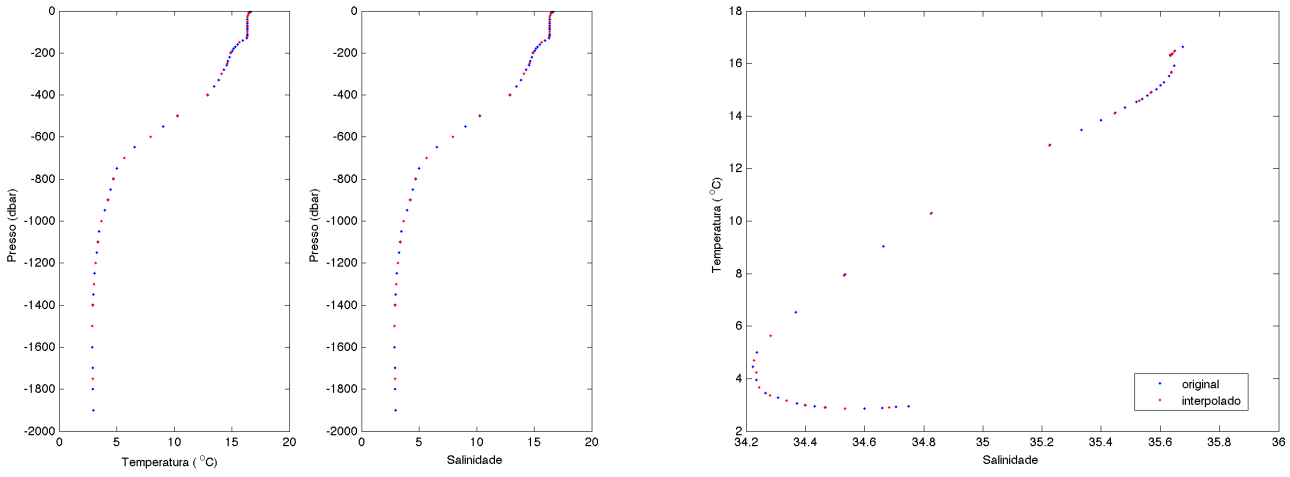

Figura 8: Perfis de temperatura e salinidade e diagrama TS da estação Argo 47. Pontos azuis representam os valores medidos e vermelhos, os interpolados.

O programa **interp2** permite a interpolação em matrizes bidimensionais e sua forma de utilização é análoga à interp1.

Aqui está um resumo dos comandos que podem ser úteis na programação:

num2str	converte número em string.
eval	permite executar uma string.
keyboard	permite paradas na execução do programa.
input	recebe informações pelo terminal.
display	mostra na tela o valor de uma variável.
interp1	interpolação unidimensional.
interp2	interpolação bidimensional.

5.3 Ajuste polinomial por regressão linear

Muitas vezes quando trabalhamos com um conjunto de pontos, estamos interessados em fazer algum tipo de ajuste polinomial. Esse ajuste pode ser um polinômio de primeiro grau ou de grau n que é determinado a partir de regressão linear. No Matlab isso pode ser feito utilizando–se a função **polyfit**.

Baixe o arquivo trend.mat e carregue–o no seu espaço de trabalho. Plote o gráfico **plot(x,y,'.')**. Teremos algo assim:

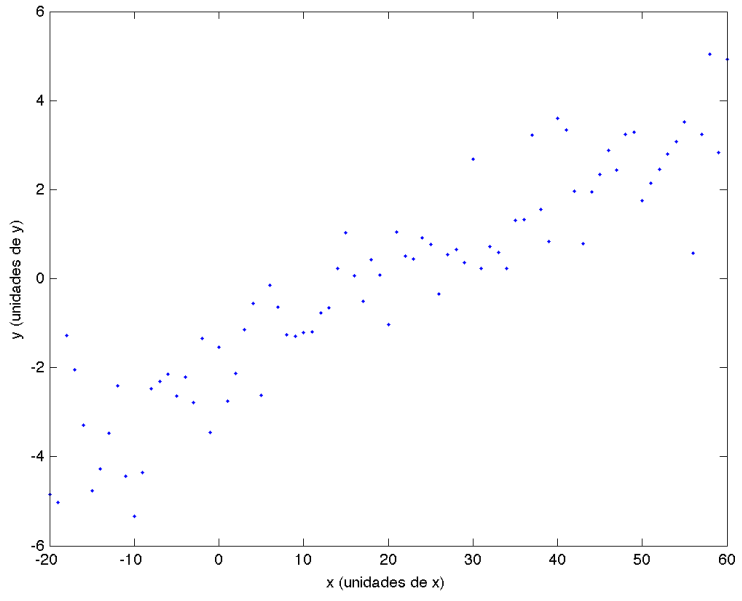

Figura 9: Exemplo de regressão linear.

O ajuste de uma reta através de regressão linear nos permitirá determinar a tendência que é observada nesses dados. Para tanto, utilizamos o ajuste polinomial.

```
load trend
plot(x,y,'.')
c=polyfit(x,y,1);
yn=polyval(c,x);
plot(x,y,'.',x,yn)
```

- A função **polyfit** irá determinar os coeficientes do ajuste polinomial, c. No exemplo, o grau do polinômio é 1, mas pode ser de qualquer grau n.
- A partir dos coeficientes, um novo vetor deve ser definido, agora calculado a partir dos coeficientes. Se a curva ajustada for de ordem 1, teremos uma reta, de ordem 2, uma parábola, etc. Usar a função **polyval** com os coeficientes.

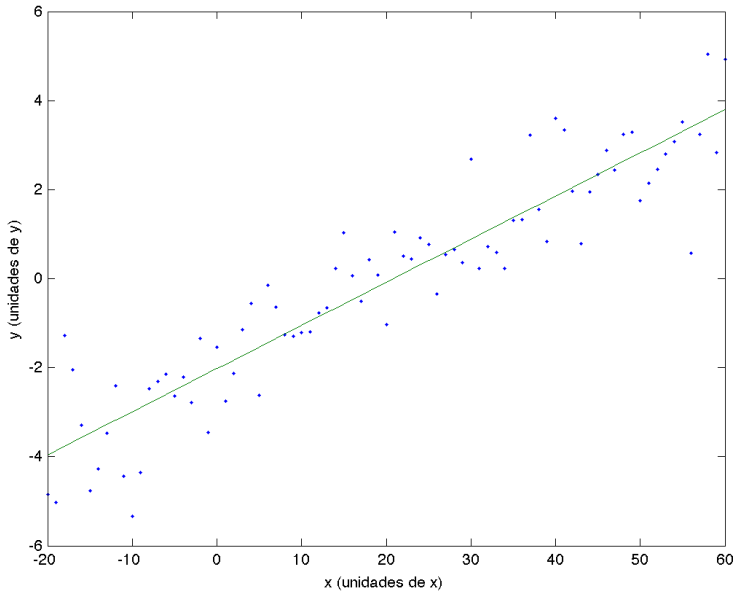

Figura 10: Exemplo de regressão linear.

6 Finalmente...

Parabéns! Você chegou no final deste curso super introdutório. Espero que tenha sido proveitoso e que este curso rápido tenha dado uma ideia sobre a vasta capacidade de processamento e versatilidade do Matlab.

Para avançar mais nos conceitos, primeiramente consulte o curso rápido online mencionado anteriormente. Outra fonte de informações é a própria Mathworks: `www.mathworks.com`.

Para ser um bom programador em Matlab, utilize–o sempre que puder, principalmente nas disciplinas desde Sistema Oceano até o curso de Ondas.

Boa sorte!