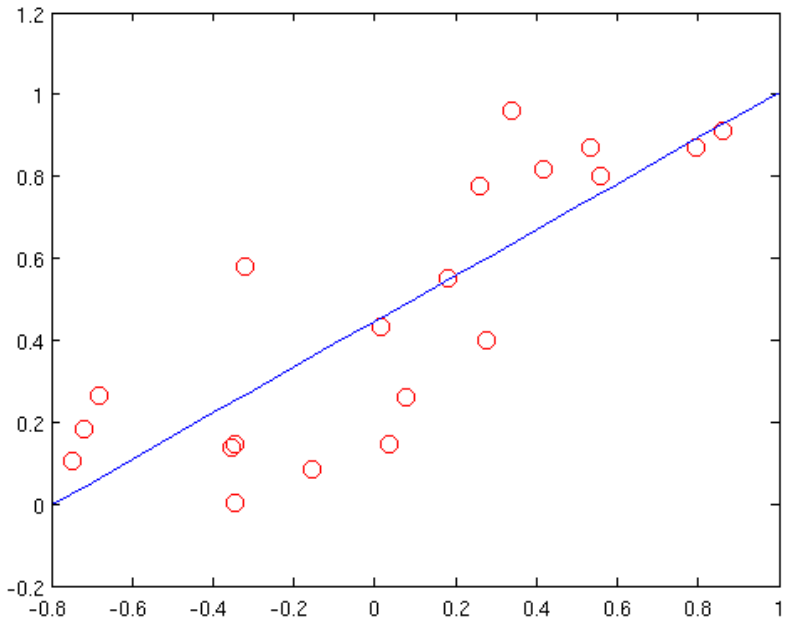
MATLAB Avançado

Melissa Weber Mendonça
melissa.mendonca@ufsc.br

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Fitting

Queremos descobrir uma função (linear, polinomial ou não-linear) que aproxime um conjunto de dados:



Regressão

Podemos calcular automaticamente um modelo de regressão (usando quadrados mínimos) através da janela de um gráfico. Exemplo:

```
>>  load census
>>  plot(cdate, pop, 'ro')
```

Na janela do gráfico, podemos selecionar

Tools $\rightarrow$ Basic Fitting

Norma dos resíduos

Podemos calcular a norma dos resíduos para um *fit* realizado através do comando

```
>>  sqrt(sum(resids.^2))
```

Podemos também extrapolar dados usando a interface gráfica do MATLAB, novamente em

Tools $\rightarrow$ Basic Fitting

Finalmente, podemos usar o comando

File $\rightarrow$ Generate Code

para criarmos uma função que reproduz o gráfico obtido.

Interpolação polinomial: `polyfit`

O comando

```
>>  p = polyfit(x,y,n)
```

encontra os coeficientes do polinômio $p(x)$ de grau n que aproxima os dados $y(i) = p(x(i))$, em um sentido de mínimos quadrados. O vetor `p` resultante contém os coeficientes do polinômio em ordem descendente de potências. O comando

```
>>  [p,S] = polyfit(x,y,n)
```

retorna os coeficientes do polinômio em `p` e uma estrutura `S` que pode ser usada com o comando `polyval`.
A estrutura `S` contém os campos `R`, `df` e `normr`.
Se os dados `y` são aleatórios, uma estimativa da covariância de `p` é

```
>>  (inv(R)*inv(R)')*normr^2/df
```

Avaliação de polinômios: `polyval`

O comando

```
>>  y = polyval(p,x)
```

retorna o valor de um polinômio de grau `n` (armazenado no vetor `p`) em `x`. O comando

```
>>  [y,delta] = polyval(p,x,S)
```

usa a estrutura `S` gerada pelo comando `polyfit` para gerar `delta`, que é uma estimativa do desvio padrão do erro obtido ao se tentar calcular `p(x)`.

Regressão por Quadrados Mínimos

Por exemplo, se quisermos fazer uma regressão linear em um conjunto de pontos (x, y), usamos o comando

```
>>  p = polyfit(x,y,1)
```

O resultado é um vetor p que contém os coeficientes da reta

$$y = p_1 x + p_2$$

Exemplo:

```
>>  x = 1:1:20;
>>  y = x + 10*rand(1,20);
>>  p = polyfit(x,y,1);
>>  plot(x,y,'ro')
>>  hold on
>>  t = 0:0.1:20;
>>  plot(t,polyval(p,t))
```

Exemplo (com resíduos)

```
>>  x = 1:1:20;
>>  y = x + 10*rand(1,20);
>>  p = polyfit(x,y,1);
>>  fitted = polyval(p,x);
>>  res = y-fitted;
>>  subplot(2,1,1), plot(x,y,'ro','markersize',8)
>>  hold on
>>  t = 0:0.1:20;
>>  subplot(2,1,1), plot(t,polyval(p,t))
>>  subplot(2,1,2), bar(x,res)
```

Scatter Plot

O comando

```
>>  scatter(X,Y)
```

faz um gráfico dos pontos com coordenadas `X` e `Y`, usando círculos como marcadores.
Se usarmos

```
>>  scatter(X,Y,S,C)
```

podemos especificar a área de cada marcador em `S`.
Outras opções:

- `scatter(...,marcador)` usa o marcador escolhido (p. ex. `'+'` ou `'*'`)
- `scatter(...,'filled')` preenche os marcadores.

scatterhist

O comando

```
>>  scatterhist(x,y)
```

cria um *scatter plot* dos dados nos vetores `x` e `y` e também um histograma em cada eixo do gráfico.

Exemplo:

```
>>  x = randn(1000,1);
>>  y = exp(.5*randn(1000,1));
>>  scatterhist(x,y)
```

`lsline`

O comando

```
>>  lsline
```

acrescenta uma reta calculada através de regressão linear (mínimos quadrados) **para cada plot/scatter plot na figura atual**.

Atenção: dados conectados com alguns tipos de reta (`'-'`, `'--'` ou `'.-'`) são **ignorados** por `lsline`.

Exemplo

```
>>  x = 1:10;
>>  y1 = x + randn(1,10);
>>  scatter(x,y1,25,'b','*')
>>  hold on
>>  y2 = 2*x + randn(1,10);
>>  plot(x,y2,'mo')
>>  y3 = 3*x + randn(1,10);
>>  plot(x,y3,'rx:')
>>  y4 = 4*x + randn(1,10);
>>  plot(x,y4,'g+--')
>>  lsline
```

refcurve

Se o vetor p contém os coeficientes de um polinômio em ordem descendente de potências, o comando

```
>>  refcurve(p)
```

adiciona uma curva de referência polinomial com coeficientes p ao gráfico atual.
Se p é um vetor com $n+1$ elementos, a curva é dada por

$$y = p(1)x^n + p(2)x^{n-1} + \ldots + p(n)x + p(n+1)$$

gline

O comando

```
>>  gline
```

permite ao usuário adicionar manualmente um segmento de reta à última figura desenhada.

A reta pode ser editada manualmente na ferramenta de edição de gráficos do MATLAB.

Ajuste polinomial

O comando

```
>>  polytool(x,y)
```

ajusta uma reta aos vetores `x` e `y` e mostra um gráfico interativo do resultado.

```
>>  polytool(x,y,n)
```

ajusta um polinômio de grau `n` aos dados.

Só disponível na Statistics Toolbox!

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Curve Fitting Toolbox

Para fazermos o ajuste de curvas de maneira interativa, podemos usar o comando

```
>>  cftool
```

Só disponível com a Curve Fitting Toolbox

Resolução de equações lineares e não-lineares em MATLAB

Comandos básicos de álgebra linear: `det`

Para calcularmos o determinante de uma matriz quadrada `A`, usamos o comando

```
>>  det(A)
```

Exemplo:

```
>>  A = [1 2 0; 3 1 4; 5 2 1]
>>  det(A)
>>  B = [1 2 3;4 5 6;7 8 9]
>>  det(B)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Comandos básicos de álgebra linear: `eig`

Para calcularmos os autovalores de `A`, usamos

```
>>  eig(A)
```

Para calcularmos também os autovetores, usamos

```
>>  [V,D] = eig(A)
```

onde `V` tem os autovetores de `A` nas colunas, e `D` é uma diagonal com os autovalores de `A`.
Exemplos:

```
>>  eig(eye(n,n))
>>  [V,D] = eig(eye(n,n))
>>  A = [1 2 3;4 5 6;7 8 9];
>>  [V,D] = eig(A)
```

Comandos básicos de álgebra linear: `inv`

Para calcularmos a inversa de uma matriz *quadrada e inversível* `A`, usamos o comando

```
>>  inv(A)
```

Exemplos:

```
>>  M = [1 4 3;2 1 0;0 0 1];
>>  inv(M)
>>  inv(M)*M
>>  A = [1 2 3;4 5 6;7 8 9];
>>  inv(A)
>>  inv(A)*A
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Resolução Sistemas Lineares no MATLAB

Aqui, vamos supor que queremos resolver um sistema linear, ou seja, um problema do tipo

Encontrar $x \in \mathbb{R}^n$ tal que

$$Ax = b$$

com $A \in \mathbb{R}^{m \times n}$ e $b \in \mathbb{R}^m$.

Sistemas quadrados: usando `inv`

Primeiramente, se a matriz `A` for quadrada e inversível, podemos encontrar

$$x = A^{-1}b.$$

usando o comando

```
>>  x = inv(A)*b
```

Sistemas quadrados: o operador \

Se, por outro lado, não for desejável encontrar a inversa da matriz `A`, podemos usar o operador \ para resolver $Ax = b$:

```
>>  x = A\b
```

ou então a função `linsolve`:

```
>>  x = linsolve(A,b)
```

resolve o sistema linear $Ax = b$ usando a fatoração LU caso a matriz seja quadrada.

- $A_{n\times n}$ inversível: solução por LU;
- $A_{n\times n}$ singular: erro.
- $A_{m\times n}$: quadrados mínimos

Sistemas quadrados: decomposição LU

Sabemos que a eliminação gaussiana leva uma matriz `A` em duas matrizes `L` e `U` tais que

$$A = LU$$

Assim,

$$Ax = b \Leftrightarrow (LU)x = b \Leftrightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$$

Para encontrarmos a decomposição LU de uma matriz A no MATLAB, usamos o comando

```
>>  [L,U] = lu(A)
```

Podemos em seguida resolver o sistema $Ax = b$ fazendo

```
>>  y = L\b
>>  x = U\y
```

Exemplo

Testar a solução com

```
>>  inv(A)*b
>>  A\b
>>  [L,U] = lu(A);
>>  U\(L\b)
>>  linsolve(A,b)
```

$$A = \begin{pmatrix} 1.0001 & 1 \\ 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

Exemplo

Testar a solução com

```
>>  inv(A)*b
>>  A\b
>>  [L,U] = lu(A);
>>  U\(L\b)
>>  linsolve(A,b)
```

$$A = \begin{pmatrix} 1.0001 & 1 \\ 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \qquad x = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

Exemplo

Testar a solução com

```
>>  inv(A)*b
>>  A\b
>>  [L,U] = lu(A);
>>  U\(L\b)
>>  linsolve(A,b)
```

$$A = \begin{pmatrix} 1.0001 & 1 \\ 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 2.0001 \\ 2 \end{pmatrix}$$

Exemplo

Testar a solução com

```
>>  inv(A)*b
>>  A\b
>>  [L,U] = lu(A);
>>  U\(L\b)
>>  linsolve(A,b)
```

$$A = \begin{pmatrix} 1.0001 & 1 \\ 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 2.0001 \\ 2 \end{pmatrix} \qquad x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Este sistema é chamado *mal-condicionado*: uma mudança pequena no lado direito muda completamente a solução.

Métodos Iterativos para Sistemas Lineares

pcg Gradiente conjugado precondicionado:

```
>>   x = pcg(A,b)
```

tenta resolver o sistema linear $n \times n$ $Ax = b$. A deve ser simétrica, definida positiva e esparsa.

Métodos Iterativos para Sistemas Lineares

bicg Gradiente bi-conjugado:

```
>>  x = bicg(A,b)
```

tenta resolver o sistema linear $n \times n$ $Ax = b$. A deve ser esparsa.

Métodos Iterativos para Sistemas Lineares

gmres Generalized minimum residual method:

```
>>  x = gmres(A,b)
```

tenta resolver o sistema linear $n \times n$ $Ax = b$. A deve ser esparsa.

Métodos Iterativos para Sistemas Lineares

lsqr LSQR (quadrados mínimos):

```
>>  x = lsqr(A,b)
```

tenta resolver o sistema $m \times n$ $Ax = b$ através do método de quadrados mínimos. A deve ser esparsa.

Resolução de equações não-lineares

Agora, queremos resolver o problema de encontrar $x \in \mathbb{R}^n$ tal que

$$F(x) = 0$$

onde $F : \mathbb{R}^n \to \mathbb{R}^m$ (onde m ou n podem ser iguais a 1).

Equação não linear a uma variável: `fzero`

Aqui, o problema que nos interessa é encontrar uma raiz da equação

$$f(x) = 0$$

onde $f : \mathbb{R} \to \mathbb{R}$.
Para isto usamos o comando

```
>>  x = fzero(fun,x0)
```

Mas: quem é `fun`?
É a *referência* (*function handle*) da função f!

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Referências a funções definidas inline

Podemos usar funções anônimas para chamar `fzero`.

Exemplo:

```
>>  quadratica = @(x) x.^2-4;
>>  x = fzero(quadratica,6)
```

ou ainda

```
>>  x = fzero(@(x) x.^2-4,6)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Referências a funções definidas em arquivo

Se a função para a qual gostaríamos de encontrar uma raiz estiver em um arquivo próprio, no formato

minhafuncao.m

```
[y] = minhafuncao(x)
    comandos
```

podemos chamar a função `fzero` a partir do ponto `x0`, escrevendo

```
>>  x = fzero(@minhafuncao,x0)
```

O algoritmo da função `fzero` usa uma combinação dos métodos da bissecção, secante e interpolação quadrática inversa.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Exemplos

- Encontrar uma das raizes de $f(x) = x^2 - 4$ a partir do ponto $x = -6$.

```
>>  quadratica = @(x) x.^2-4;
>>  fzero(quadratica,-6)
```

- Encontrar uma raiz de $f(x) = e^{2x} - 3$.

```
>>  fun = @(x) exp(2*x)-3;
>>  fzero(fun,0)
```

Raizes de um polinômio: `roots`

Para encontrar as raizes de um polinômio de grau n da forma

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

primeiramente representamos este polinômio como um vetor *linha* `p` no MATLAB, cujas componentes são os coeficientes dos termos em ordem descendente de grau, ou seja,

```
>>  p = [a_n  a_{n-1}  ···  a_2  a_1  a_0]
```

Em seguida, usamos o comando

```
>>  r = roots(p)
```

resultando em um vetor coluna `r` com as raizes deste polinômio.

Exemplo

$$p(x) = t^3 + 2t^2 - 5t - 6$$

```
>>  p = [1 2 -5 -6]
>>  roots(p)
```

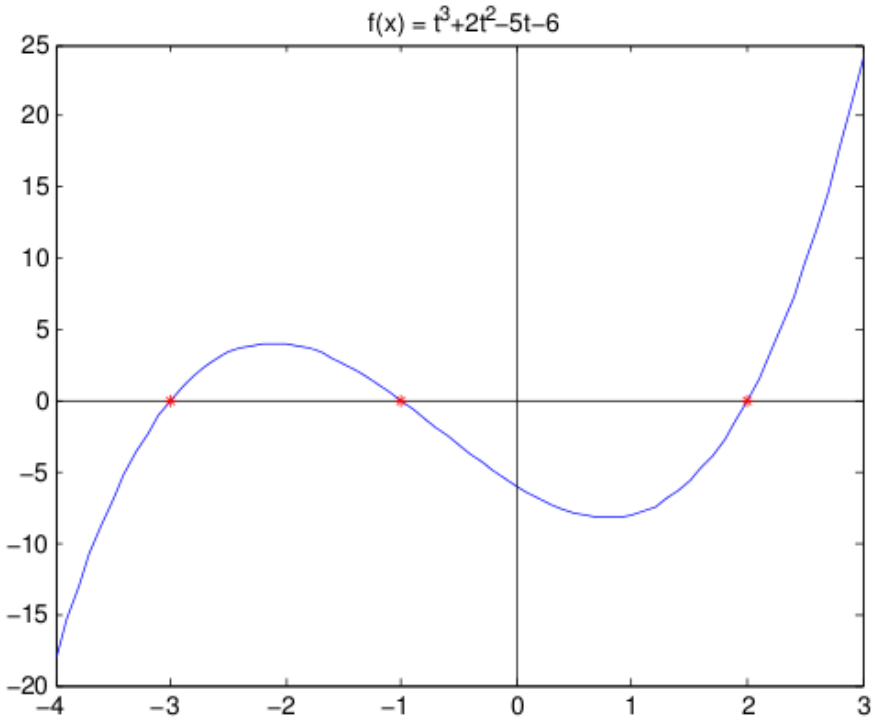


Figura : $p(x) = t^3 + 2t^2 - 5t - 6$ e suas raizes.

Sistema de equações não lineares: `fsolve`

Para encontrarmos a solução de um sistema de equações não lineares da forma

$$F(x) = 0$$

onde $F : \mathbb{R}^n \to \mathbb{R}^m$, usamos a função `fsolve`, identicamente à função `fzero`:

```
>>  fsolve(@minhafuncao,x0)
```

se utilizarmos uma função em arquivo, ou

```
>>  fsolve(fun,x0)
```

se utilizarmos uma função anônima.

Só na Optimization Toolbox!

Exemplo

Resolver o sistema de equações

$$\begin{cases} y_1 &= 3x_1^2 + 4x_2^2 - 16 \\ y_2 &= 2x_1^2 - 3x_2^2 - 5 \end{cases}$$

```
>>  fun = @(x) [3*x(1).^2+4*x(2).^2-16;
                2*x(1).^2-3*x(2).^2-5];
>>  fsolve(fun,[1;1])
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Exemplo

Encontrar a raiz de

$$F(x) = \begin{bmatrix} x_1^2 + x_2 x_3 \\ sin(x_1 + 2x_2 - 3x_3) \end{bmatrix}$$

```
>>  fun = @(x) [x(1).^2+x(2).*x(3);
                sin(x(1)+2*x(2)-3*x(3))];
>>  fsolve(fun,[1;1;1])
>>  fsolve(fun,[0;0;0])
```

Otimização: Minimização de funções

Agora, queremos resolver o problema

$$\text{minimizar } f(x).$$

Para encontrarmos o mínimo de uma função real de várias variáveis, a partir de um ponto inicial `x0`, usamos o comando

```
>>  x = fminsearch(@funcao,x0)
```

Se quisermos também saber o valor da função no ponto de mínimo, usamos a sintaxe

```
>>  [x,fval] = fminsearch(@funcao,x0)
```

Exemplo

Minimizar

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

```
>>  f = @(x) 100*(x(2)-x(1).^2).^2+(1-x(1)).^2
>>  fminsearch(f,[0;0])
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Minimização de uma função de uma variável com restrições: `fminbnd`

Para encontrarmos o mínimo de uma função de uma variável dentro de um intervalo $[a, b]$, usamos o comando

```
>>  x = fminbnd(@funcao,a,b)
```

Se quisermos também saber o valor da função no ponto de mínimo, usamos a sintaxe

```
>>  [x,fval] = fminbnd(@funcao,a,b)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Exemplos

- Minimizar $f(x) = x$ nos intervalos $[0, 1]$ e $[-10, 1]$.

```
>>  f = @(x) x;
>>  fminbnd(f,0,1)
>>  fminbnd(f,-10,1)
```

- Minimizar $f(x) = x^2 - 1$ no intervalo $[1, 3]$.

```
>>  f = @(x) x.^2;
>>  fminbnd(f,1,3)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Outros comandos úteis

Integração numérica geral: `integral`

Para calcularmos uma aproximação numérica de $\int_a^b f(x)dx$, usamos o comando

```
>>  q = integral(fun,a,b)
```

em que `fun` é uma referência a uma função.

Exemplo:
Calcular a integral imprópria de $f(x) = e^{-x^2}(\ln(x))^2$ entre 0 e ∞.

```
>>  fun = @(x) exp(-x.^2).*log(x).^2;
>>  q = integral(fun,0,Inf)
```

Integração numérica finita: `quad`

Para calcularmos uma aproximação numérica de $\int_a^b f(x)dx$ pela quadratura de Simpson (adaptativa), usamos o comando

```
>>  q = quad(fun,a,b)
```

em que `fun` é uma referência a uma função.

Integração numérica discreta: `trapz`

Se tudo o que conhecemos sobre a função é seus valores em um conjunto de pontos, podemos aproximar o valor da sua integral $\int_a^b f(x)dx$ usando o comando `trapz`. Para calcularmos uma aproximação numérica de $\int_a^b f(x)dx$, primeiramente precisamos representar a função f em um conjunto de pontos:

```
>>  x = 0:pi/100:pi;
>>  y = sin(x);
```

Agora, usamos o comando `trapz`:

```
>>  z = trapz(x,y)
```

Diferenciação Numérica: `gradient`

O *gradiente* de uma função $f : \mathbb{R}^n \to \mathbb{R}$ é dado por

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial F}{\partial x_n}(x) \right).$$

Para calcularmos o gradiente de uma função de uma variável, procedemos da seguinte maneira.

```
>>  x = a:h:b;
>>  f = funcao(x);
>>  g = gradient(f,h)
```

O comando `gradient` calcula numericamente a derivada de f em função da variável x nos pontos escolhidos.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

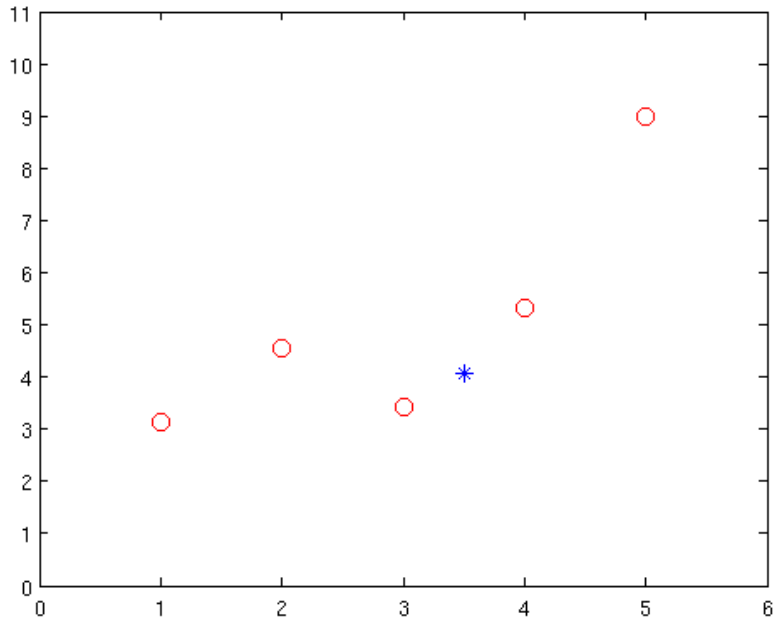
Diferenciação Numérica: gradient

Para calcularmos o gradiente de uma função de duas variáveis, o procedimento é equivalente. A diferença é que agora precisamos gerar uma malha de pontos usando o comando meshgrid.

```
>>  x = a:hx:b;
>>  y = c:hy:d;
>>  [x,y] = meshgrid(x,y);
>>  f = funcao(x,y)
>>  [gx,gy] = gradient(f,hx,hy)
```

Interpolação

Suponha que temos um conjunto de dados, e gostaríamos de encontrar uma função polinomial (ou polinomial por partes) que *interpole* estes pontos.



Interpolação 1D: `interp1`

O comando

```
>>  yi = interp1(x,Y,xi,method)
```

interpola os dados (`x`,`Y`) nos novos pontos `xi`, usando o método `method`, que pode ser:

- `'nearest'` Vizinho mais próximo
- `'linear'` Interpolação linear (default)
- `'spline'` Splines cúbicos
- `'cubic'` Interpolação por polinômios de Hermite

Exemplo

```
>>  x = 0:10;
>>  y = sin(x);
>>  xi = 0:.25:10;
>>  yi = interp1(x,y,xi);
>>  plot(x,y,'o',xi,yi);
>>  hold on;
>>  zi = interp1(x,y,xi,'nearest');
>>  plot(xi,zi,':k');
>>  wi = interp1(x,y,xi,'spline');
>>  plot(xi,wi,'m');
>>  ui = interp1(x,y,xi,'cubic');
>>  plot(xi,ui,'--g')
```

Interpolação 2D: `interp2`

O comando

```
>>  ZI = interp2(X,Y,Z,XI,YI,method)
```

interpola os dados (`X`,`Y`,`Z`) nos novos pontos (`XI`,`YI`) usando o método `method`, que pode ser

- `'nearest'` Vizinho mais próximo
- `'linear'` Interpolação linear (default)
- `'spline'` Splines cúbicos
- `'cubic'` Interpolação cúbica, se os dados forem uniformemente espaçados; senão, é o mesmo que `spline`.

Resolução de Equações Diferenciais

Uma EDO é uma equação que envolve uma ou mais derivadas de uma variável dependente y com respeito a uma única variável independente t ($y = y(t)$).
O MATLAB resolve equações diferenciais ordinárias de primeira ordem dos seguintes tipos:

- EDOs explícitas, do tipo $y' = f(t, y)$
- EDOs linearmente implícitas, do tipo $M(t, y)y' = f(t, y)$, em que $M(t, y)$ é uma matriz
- EDOs implícitas, do tipo $f(t, y, y') = 0$

Para resolvermos equações diferenciais de ordem superior, precisamos escrevê-las como um sistema de equações de primeira ordem (como fazemos no curso de cálculo).

Problemas de Valor Inicial

Geralmente, temos uma família de soluções $y(t)$ que satisfaz a EDO. Para obtermos uma solução única, exigimos que a solução satisfaça alguma condição inicial específica, de forma que $y(t_0) = y_0$ em algum valor inicial t_0.

$$\begin{cases} y' = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

Solvers

A sintaxe para resolver uma equação diferencial é

```
>>  [t,y] = solver(odefun,tspan,y0,options)
```

Os argumentos de entrada são sempre os seguintes:

- `odefun`: O *handle* para uma função que avalia o sistema de EDOs em um ponto. Esta função deve estar na forma `dydt = odefun(t,y)`, onde `t` é um escalar e `dydt` e `y` são vetores coluna.
- `tspan`: vetor especificando o intervalo de integração.
- `y0`: vetor das condições iniciais para o problema.
- `options`: Struct de parâmetros opcionais.

Os argumentos de saída são

- `t`: vetor coluna das variáveis independentes (pontos no intervalo desejado)
- `y`: vetor ou matriz contendo, em cada linha, a solução calculada no ponto contido na linha correspondente de `t`.

Solvers

Os métodos disponíveis estão divididos de acordo com o tipo de problema que resolvem:

- Problemas Não-Stiff:
  - `ode45` (Runge-Kutta, passo simples)
  - `ode23` (Runge-Kutta, passo simples)
  - `ode113` (Adams-Bashforth-Moulton, passo múltiplo)
- Problemas Stiff:
  - `ode15s` (numerical differentiation formulas (NDFs), passo múltiplo)
  - `ode23s` (Rosenbrock, passo único)
  - `ode23t` (Trapezoide)
  - `ode23tb` (Runge-Kutta)

Para equações implícitas da forma

$$f(t, y, y') = 0,$$

pode-se usar o solver `ode15i`.

Exemplo

Resolver o problema de valor inicial

$$\begin{cases} y'(t) = \cos t \\ y(0) = 0 \end{cases}$$

```
>>  odefun = @(t,y) cos(t);
>>  [T,Y] = ode45(odefun,[0 1],0)
>>  plot(T,Y)
>>  hold on
>>  plot(T,sin(T),'m')
```