Universidade Federal do Paraná
Programa de Pós-Graduação em Geologia

GEOL7048: Tópicos Especiais em Geologia Exploratória II

Métodos semiquantitativos

Saulo P. Oliveira

Departamento de Matemática, Universidade Federal do Paraná

Aula 4

- Funções
- Funções de vetores/matrizes

Funções

Podemos definir funções em arquivos com extensão .m, inserindo no início do arquivo a linha

```
function [S1,S2,...]  = NOME(E1,E2,...)
```

Funções

Podemos definir funções em arquivos com extensão .m, inserindo no início do arquivo a linha

```
function [S1,S2,...]  = NOME(E1,E2,...)
```

- `S1,S2,...`: variáveis de saída
- `NOME`: nome da função (e do arquivo)
- `E1,E2,...`: variáveis de entrada

Funções

Podemos definir funções em arquivos com extensão .m, inserindo no início do arquivo a linha

```
function [S1,S2,...]  = NOME(E1,E2,...)
```

- `S1,S2,...`: variáveis de saída
- `NOME`: nome da função (e do arquivo)
- `E1,E2,...`: variáveis de entrada

Exemplo: função `sen` em vez de `sin`:

```
function y = sen(x)
  y = sin(x);
```

`sen.m`

Funções

Podemos definir funções em arquivos com extensão .m, inserindo no início do arquivo a linha

```
function [S1,S2,...]  = NOME(E1,E2,...)
```

- `S1,S2,...`: variáveis de saída
- `NOME`: nome da função (e do arquivo)
- `E1,E2,...`: variáveis de entrada

Exemplo: função `sen` em vez de `sin`:

```
function y = sen(x)
  y = sin(x);
```

`sen.m`

```
>> sen(pi/2)
   1
```

Funções

As variáveis de saída e de entrada são opcionais, ou seja, podemos ter funções do tipo

- `function [S1,S2,...] = NOME(E1,E2,...)`
- `function [S1,S2,...] = NOME`
- `function NOME(E1,E2,...)`
- `function NOME`

Funções

As variáveis de saída e de entrada são opcionais, ou seja, podemos ter funções do tipo

- `function [S1,S2,...] = NOME(E1,E2,...)`
- `function [S1,S2,...] = NOME`
- `function NOME(E1,E2,...)`
- `function NOME`

Nos dois últimos casos, os comandos são executados sem que nenhuma variável da memória seja alterada:

```
n=1
```

script_n.m

```
function function_x
  n=1
```

function_n.m

Funções

As variáveis de saída e de entrada são opcionais, ou seja, podemos ter funções do tipo

- `function [S1,S2,...] = NOME(E1,E2,...)`
- `function [S1,S2,...] = NOME`
- `function NOME(E1,E2,...)`
- `function NOME`

Nos dois últimos casos, os comandos são executados sem que nenhuma variável da memória seja alterada:

```
n=1
```

script_n.m

```
function function_x
  n=1
```

function_n.m

Funções

Na execução, pode-se: 1) mudar o nome das variáveis de entrada ou inserir dados; 2) mudar o nome das variáveis de saída ou omití-las:

```
function v = vet(x)
  if (x>=1)
    for k = 1:x
      v(k) = k;
    end
  end
```

vet.m

Funções

Na execução, pode-se: 1) mudar o nome das variáveis de entrada ou inserir dados; 2) mudar o nome das variáveis de saída ou omití-las:

```
function v = vet(x)
  if (x>=1)
    for k = 1:x
      v(k) = k;
    end
  end
```

vet.m

```
>> y=8.3;
>> v = vet(y)
```

Funções

Na execução, pode-se: 1) mudar o nome das variáveis de entrada ou inserir dados; 2) mudar o nome das variáveis de saída ou omití-las:

```
function v = vet(x)
  if (x>=1)
    for k = 1:x
      v(k) = k;
    end
  end
```

vet.m

```
>> y=8.3;
>> v = vet(y)
>> w = vet(4)
```

Funções

Na execução, pode-se: 1) mudar o nome das variáveis de entrada ou inserir dados; 2) mudar o nome das variáveis de saída ou omití-las:

```
function v = vet(x)
  if (x>=1)
    for k = 1:x
      v(k) = k;
    end
  end
```

vet.m

```
>> y=8.3;
>> v = vet(y)
>> w = vet(4)
>> vet(4)
```

Funções

Na execução, pode-se: 1) mudar o nome das variáveis de entrada ou inserir dados; 2) mudar o nome das variáveis de saída ou omití-las:

```
function v = vet(x)
  if (x>=1)
    for k = 1:x
      v(k) = k;
    end
  end
```

vet.m

```
>> y=8.3;
>> v = vet(y)
>> w = vet(4)
>> vet(4)
>> k
```

Funções

Na execução, pode-se: 1) mudar o nome das variáveis de entrada ou inserir dados; 2) mudar o nome das variáveis de saída ou omití-las:

```
function v = vet(x)
  if (x>=1)
    for k = 1:x
      v(k) = k;
    end
  end
```

vet.m

```
>> y=8.3;
>> v = vet(y)
>> w = vet(4)
>> vet(4)
>> k
```

OBS: as variáveis de saída precisam ser preenchidas na função.

Funções de vetores/matrizes

Consideremos agora funções cujas variáveis de entrada são vetores de tamanho 2:

```
function d = dist(u,v)
  dx = u(1)-v(1); dy = u(2)-v(2);
  d = sqrt( dx^2 + dy^2 );
```

dist.m

Funções de vetores/matrizes

Consideremos agora funções cujas variáveis de entrada são vetores de tamanho 2:

```
function d = dist(u,v)
  dx = u(1)-v(1); dy = u(2)-v(2);
  d = sqrt( dx^2 + dy^2 );
```

dist.m

```
>> u=[1,1];
>> v=[2,3];
>> dist(u,v)
```

Funções de vetores/matrizes

Consideremos agora funções cujas variáveis de entrada são vetores de tamanho 2:

```
function d = dist(u,v)
  dx = u(1)-v(1); dy = u(2)-v(2);
  d = sqrt( dx^2 + dy^2 );
```

dist.m

```
>> u=[1,1];
>> v=[2,3];
>> dist(u,v)
>> w = [4,4];
>> d = dist(u,w);
```

Funções de vetores/matrizes

Consideremos agora funções cujas variáveis de entrada são vetores de tamanho 2:

```
function d = dist(u,v)
  dx = u(1)-v(1); dy = u(2)-v(2);
  d = sqrt( dx^2 + dy^2 );
```

dist.m

```
>> u=[1,1];
>> v=[2,3];
>> dist(u,v)
>> w = [4,4];
>> d = dist(u,w);
>> d
```

Funções de vetores/matrizes

Consideremos agora funções cujas variáveis de entrada são vetores de tamanho 2:

```
function d = dist(u,v)
  dx = u(1)-v(1); dy = u(2)-v(2);
  d = sqrt( dx^2 + dy^2 );
```

dist.m

```
>> u=[1,1];
>> v=[2,3];
>> dist(u,v)
>> w = [4,4];
>> d = dist(u,w);
>> d
>> dx
```

Funções de vetores/matrizes

Consideremos agora funções cujas variáveis de entrada são vetores de tamanho 2:

```
function d = dist(u,v)
  dx = u(1)-v(1); dy = u(2)-v(2);
  d = sqrt( dx^2 + dy^2 );
```

dist.m

```
>> u=[1,1];
>> v=[2,3];
>> dist(u,v)
>> w = [4,4];
>> d = dist(u,w);
>> d
>> dx
>> d2 = dist([1,0],[4,4])
```

Funções de vetores/matrizes

Na primeira aula, vimos a criação de matrizes pelos comandos `zeros` e `ones` :

```
>> A = zeros(3,4)
>> B = ones(2,2)
```

Funções de vetores/matrizes

Na primeira aula, vimos a criação de matrizes pelos comandos `zeros` e `ones` :

```
>> A = zeros(3,4)
>> B = ones(2,2)
>> B = ones(3)
```

Funções de vetores/matrizes

Na primeira aula, vimos a criação de matrizes pelos comandos `zeros` e `ones` :

```
>> A = zeros(3,4)
>> B = ones(2,2)
>> B = ones(3)
```

Em geral `zeros(m,n)` e `ones(m,n)` geram matrizes de tamanho `m`$\times$`n` com 0s (1s). Se o argumento `n` não for informado, temos uma matriz quadrada `m`$\times$`m`.

Funções de vetores/matrizes

Na primeira aula, vimos a criação de matrizes pelos comandos `zeros` e `ones` :

```
>> A = zeros(3,4)
>> B = ones(2,2)
>> B = ones(3)
```

Em geral `zeros(m,n)` e `ones(m,n)` geram matrizes de tamanho `m`×`n` com 0s (1s). Se o argumento `n` não for informado, temos uma matriz quadrada `m`×`m`.

Exercício: defina a função `twos.m` que gera matrizes `m`×`n` preenchidas com 2.

Primeira linha:

Funções de vetores/matrizes

Na primeira aula, vimos a criação de matrizes pelos comandos `zeros` e `ones` :

```
>> A = zeros(3,4)
>> B = ones(2,2)
>> B = ones(3)
```

Em geral `zeros(m,n)` e `ones(m,n)` geram matrizes de tamanho `m`×`n` com 0s (1s). Se o argumento `n` não for informado, temos uma matriz quadrada `m`×`m`.

Exercício: defina a função `twos.m` que gera matrizes `m`×`n` preenchidas com 2.

Primeira linha: `function A = twos(m,n)`

Funções de vetores/matrizes

`A=twos(1,2)` retorna `A=[2,2]`

`A=twos(3,2)` retorna `A=[2,2;2,2;2,2]`

Funções de vetores/matrizes

Possíveis comandos:

```
A=twos(1,2): A(1,1)=2;A(1,2)=2;
A=twos(3,2): A(1,1)=2;...;A(3,1)=2;A(3,2)=2;
```

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2): A(1,1)=2;A(1,2)=2;`

`A=twos(3,2): A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
A(1,1)=2;
A(1,2)=2;
...
A(1,30)=2;
```

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
```

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
```

Se fosse `A=twos(2,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
for j = 1:30
  A(2,j) = 2;
end
```

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
```

Se fosse `A=twos(2,30)`:

```
for i = 1:2
  for j = 1:30
    A(i,j) = 2;
  end
end
```

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
```

Em Geral, se `A=twos(m,n)`:

```
for i = 1:m
  for j = 1:n
    A(i,j) = 2;
  end
end
```

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
```

```
function A = twos(m,n)

for i = 1:m
  for j = 1:n
    A(i,j) = 2;
  end
end
```

twos.m

Funções de vetores/matrizes

Possíveis comandos:

`A=twos(1,2)`: `A(1,1)=2;A(1,2)=2;`

`A=twos(3,2)`: `A(1,1)=2;...;A(3,1)=2;A(3,2)=2;`

repetição! (`for`)

Suponha que tivéssemos que construir `A=twos(1,30)`:

```
for j = 1:30
  A(1,j) = 2;
end
```

```
function A = twos(m,n)

for i = 1:m
  for j = 1:n
    A(i,j) = 2;
  end
end
```

twos.m