Universidade Nova de Lisboa
OMNIS CIVITAS CONTRA SE DIVISA NON STABIT
Faculdade de Ciências e Tecnologia

Scripts e Funções (Ciclos)

Pedro Barahona
DI/FCT/UNL
Métodos Computacionais
1º Semestre 2016/2017

Ciclos

- A forma mais geral de "execução repetida" de uma sequência de instruções é, como foi visto anteriormente, controlado por uma instrução "enquanto".

- Repetimos abaixo o ciclo com que se obtem o valor do factorial do número n.

- Muitas vezes, como no exemplo ao lado, este ciclo tem um padrão bem definido, em que se pretende que uma variável (neste caso, i) tome todos os valores entre um valor inicial (1) e um valor final (n).

- Como este padrão é muito frequente, existe uma instrução "para" (***for***) que permite simplificar a especificação destes programas

```
entra n;
f ← 1;
i ← 1;
enquanto i <= n fazer
   f ← f * i;
   i ← i + 1;
fim enquanto;
sai f;
```

Ciclos

- Neste caso, podemos comparar as especificações alternativas (quer em pseudo-código, quer na linguagem OCTAVE.

```
f ← 1;
i ← 1;
enquanto i <= n fazer
   f ← f * i;
   i ← i + 1;
fim enquanto;
```

```
f = 1;
i = 1;
while  i <= n
   f ← f * i;
   i ← i + 1;
endwhile;
```

```
f ← 1;
para i de 1 até n (passo 1)
  f ← f * i;
fim para;
```

```
f = 1;
for i= 1:1:n
  f ← f * i;
endfor;
```

Ciclos

- Os ciclos **para** podem especificar quaisquer valores inicial, final e para o passo, podendo este ser negativo! Quando tem o valor 1 (muito frequente), pode omitir-se.

- Em cada ciclo a variável de iteração é incrementada / decrementada. A iteração realiza-se se o valor da variável for não superior /inferior ao valor final.

```
f = 1;
for i= 1:1:n
  f ← f * i;
endfor;
```

```
f = 1;
for i= 1:n
  f ← f * i;
endfor;
```

```
f = 1;
for i= n:-1:1
  f ← f * i;
endfor;
```

- Tipicamente os valores usados nos ciclos (inicial, passo, final) são inteiros, mas no OCTAVE podem ser usados números reais.

- Em particular, todos os programas acima são equivalentes, (se n for um inteiro).

- O ciclo não se realiza se o valor inicial é superior/inferior ao final. No exemplo acima, o ciclo não se realiza para n = 0 (como se pretende, pois 0! = 1)

Ciclos

- Embora menos frequente, é possível usar valores inicial e final diferentes de 1, como nos exemplos abaixo:

```
s = 0;
for i = 5:2:15
  s = s + i;
endfor;
```

S = 5+7+9+11+13+15

```
s = 0;
for i = 20:-2:10
  s = s + i;
endfor;
```

S = 20+18+16+14+12+10

Ciclos

- Quando há garantia que o ciclo é executado pelo menos uma vez, pode colocar-se a verificação da condição no fim do ciclo "repetir", mas a condição é alterada (normalmente negada, como indicada abaixo)

```
f ← 1;
i ← 1;
enquanto i <= n fazer
   f ← f * i;
   i ← i + 1;
fim enquanto;
```

```
f = 1;
i = 1;
while  i <= n
   f ← f * i;
   i ← i + 1;
endwhile;
```

```
f ← 1;
i ← 1;
repetir
   f ← f * i;
   i ← i + 1;
até i > n ;
```

```
f = 1;
i = 1;
do
   f ← f * i;
   i ← i + 1;
until i > n ;
```

Indentação

- Para melhorar a legibilidade dos programas convem indentá-los, usando as instruções se, enquanto, para e repetir como "parenteses" e escrevendo os blocos que são controlados uns caracteres para a direita.

```
sp = 0; si = 0;
for i = 1:20
  if rem(i,2) == 0
    sp = sp + i;
  else
    si = si + i;
  endif
endfor;
```

- A função predefinida `rem(i,2)` determina o resto da divisão de `i` por `2`.
- A condição `rem(i,2) == 0` determina pois se o número i é par.
- Para obter a divisão inteira dos inteiros positivos `m` e `n`, utiliza-se `floor(m/n)`.

- Deixa-se para exercício determinar os valores de **sp** e **si** no final do programa!

Programas / Scripts

- De uma forma geral, podemos definir um programa como um conjunto estruturado de instruções.
- Naturalmente não queremos repetir a escrita dessas instruções cada vez que as queremos executar, pelo que as deveremos armazenar num ficheiro.
- A forma mais “simples” de armazenar as instruções é através de um script ou guião, que precisa de ser criado e posteriormente chamado.

Programas / Scripts

- Criação de um script
  - um script Matlab é um ficheiro contendo código fonte e gravado com a extensão ".m"
- Chamar o script
  - Escrever o nome do script (sem o .m)
    - se não existe variável ou função definida, o interpretador procura o ficheiro **nome.m**
    - **tem de estar na pasta de trabalho** (ou numa pasta que o interpretador conheça)

Programas / Scripts

- Vamos exemplificar este caso com a queda de um grave, sem atrito e na vertical, que é governada pela equação do 2º grau

$$h(t)= gt^2 + v_0t +h_0$$

em que h(t) representa a altura do grave no instante t

- g é a aceleração da gravidade
- v_0 é a velocidade inicial é a velocidade inicial
- h_0 a altura inicial

- Naturalmente, para determinar o tempo de queda basta resolver a equação do 2º grau associada para o que usaremos o script com nome queda0.m.

Programas / Scripts

- O script “queda0.m” é mostrado de seguida, tal como criado na GUI do Octave.

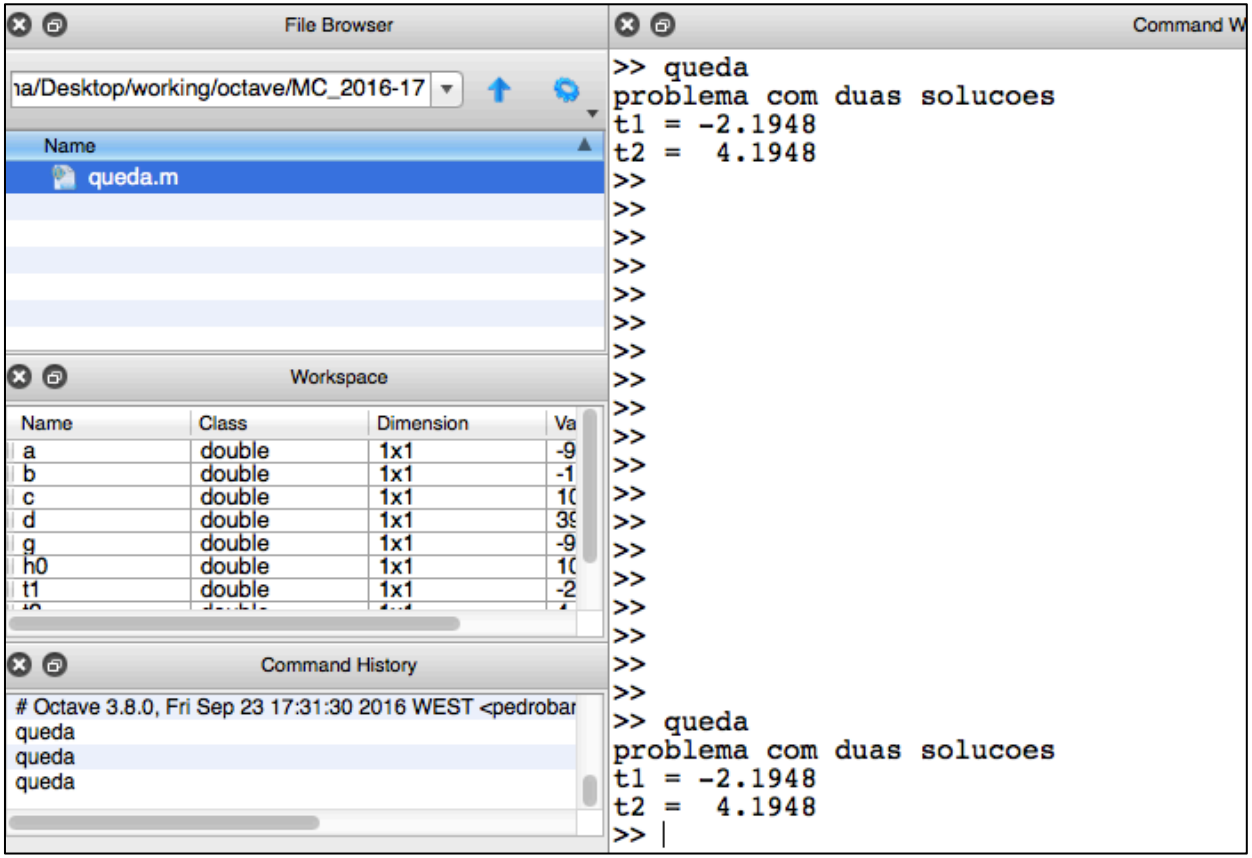
* /Users/pedrobarahona/Desktop/working/octave/MC_2016-17/queda0.m

```
1  g = -9.8;
2  v0 = -1;
3  h0 = 100;
4
5  a = g;
6  b = v0;
7  c = h0;
8
9
10 d = b^2 - 4*a*c;
11 if d < 0
12 disp("problema sem solucao")
13 elseif d == 0
14 disp("problema com solucao unica")
15 t = -b/(2*a);
16 else
17 disp("problema com duas solucoes")
18 t1 = -b+sqrt(d)/(2*a)
19 t2 = -b-sqrt(d)/(2*a)
20 end
```

- Nota: a melhorar...

Programas / Scripts

- Para chamar o script basta colocarmo-nos da directoria em que foi gravado o script e chamar o script pelo nome



Programas / Scripts

- Em resumo, na linguagem MATLAB, um ficheiro .m
  - É interpretado como texto, código fonte
- Quando o interpretador encontra um identificador verifica
  - 1º se é uma variável definida
  - 2º se é uma função já carregada
  - 3º se há um ficheiro com esse nome e extensão .m
    - Se chega à 3º opção, executa o que está no ficheiro
- **Nota**: Se o ficheiro não está na pasta de trabalho (corrente), deveremos indicar o caminho quer a partir da diretoria corrente quer a partir da raiz das diretorias.

Programas / Scripts

- Vantagens da utilização de scripts
  - Maior facilidade de execução e de edição
  - Aumentar a inteligibilidade
    - Utilização de indentação
      - Avançar o texto entre o início e o fim de um “bloco”
      - Essencial, mesmo para pequenos programas
    - Possibilidade de colocação de comentários
      - Em MATLAB, todo o texto que se segue ao caracter %
  - Debug
    - Visualizar o valor de variáveis quando da atribuição

Programas / Scripts

- Eis o script “queda.m” com comentários e indentação e visualização do discriminante.

* /Users/pedrobarahona/Desktop/working/octave/MC_2016-17/queda.m

```
1  g = -9.8;
2  v0 = -1;     % valor da velocidade inicial
3  h0 = 100;    % valor da altura inicial
4
5  a = g;
6  b = v0;
7  c = h0;
8
9  d = b^2 - 4*a*c  % valor do discriminante
10 if d < 0
11   disp("problema sem solucao")
12 elseif d == 0
13   disp("problema com solucao unica")
14   t = -b/(2*a);
15 else
16   disp("problema com duas solucoes")
17   t1 = -b+sqrt(d)/(2*a)
18   t2 = -b-sqrt(d)/(2*a)
19 end
```

Programas / Scripts

- Problemas com a utilização de scripts
  - Código fica todo junto
    - diferentes tarefas não são separadas
    - variáveis comuns a todo o programa
      - Incluindo variáveis na área de trabalho
  - Não permite estruturar os programas
- Para ultrapassar os problemas com os scripts deveremos utilizar **funções**

Programas / Funções

- Abaixo a função de cálculo do tempo de queda assumindo-se valores positivos para a altura inicial, de forma a não se ter de considerar valore imaginários.

/Users/pedrobarahona/Desktop/working/octave/MC_2016-17/quedaf.m

```
1  function t = quedaf(v0, h0);
2  % função para cálculo do tempo de queda de um grave
3  % assume-se h0 positivo e o discriminante positivo
4    g = -9.8;
5
6    d = v0^2 - 4*g*h0
7    t = -v0-sqrt(d)/(2*g)
8  end
9
```

Programação Estruturada

- Resolver problemas complexos através de problemas simples
  - um de cada vez
- Hierarquizar os problemas
  - Abstrair dos detalhes concretos
  - Generalizar soluções
  - Decompor as tarefas e implementar partes
    - **Testar cada uma**
- Juntar tudo no programa final
- Funções permitem implementar esta metodologia

Funções

- Em geral, as linguagens de programação, além de oferecerem funções pré-definidas (ex: sqrt(x), cos(x),…) permitem que o programador defina as suas próprias funções.

- A ideia é *abstrair* numa função (com nome, inputs e outputs) todos os procedimentos necessários para calcular os resultados pretendidos, isto é, calcular os outputs a partir dos inputs.

Funções

- Esta forma de proceder, tem muitas vantagens, já que permite:
  - **Estruturar** um programa em componentes básicos;
  - Reutilizar esses componentes básicos noutros programas.
- De notar que qualquer linguagem contem várias funções pré-definidas, como por exemplo as funções max/2 e sqrt/1 já utilizadas.

Funções

- Uma função simples

```
function d = dobro(a);
    d = 2*a
end
```

- Assinatura da função

Funções

- Uma função simples

```
function d = dobro(a);
   d = 2*a
end
```

- Nome da variável
  - (na função)
  - com o valor a devolver.
- Operacionalmente:
  - quando chega ao end final, o interpretador devolve uma cópia do valor guardado na variável indicada aqui.

Funções

- Uma função simples

```
function d = dobro(a);
   d = 2*a
end
```

- Nome da função
  - Para o interpretador encontrar a função o nome da função tem de ser igual ao nome do ficheiro com o código fonte:

Funções

- Uma função simples

```
function d = dobro(a);
   d = 2*a
end
```

- Parâmetros da função
  - variáveis da função que recebem os **argumentos**, cujo valor se desconhece na implementação usados na execução.
- e.g: dobro(5)
  - parâmetro: a
  - argumentos: 5

Funções

- Uma função simples

```
function d = dobro(a);
   d = 2*a
end
```

- Os parâmetros contêm cópias dos argumentos fornecidos à função.
  - O parâmetro **a** contem uma cópia do valor com que a função é chamada
- Estas variáveis **a** e **d** só existem dentro da função.

Funções

- Uma função simples

```
function d = dobro(a);
    d = 2*a
end
```

- De notar:
  - A função tem de estar guardada num ficheiro com o nome da função e com extensão .m
    - Neste caso, no ficheiro “igualdois.m”
  - Normalmente (mas não necessariamente) uma função
    - Tem parâmetros de entrada (neste caso a)
    - Retorna um valor (neste caso d)

Funções

- Uma função é chamada pelo seu nome, indicando-se no caso de haver, os parâmetros de entrada

```
function d = dobro(a);
   d = 2*a
end
```

- Geralmente a chamada da função é feita para afetar o valor de uma variável do contexto em que é chamada.

```
octave:13> x = 4
a = 4
octave:14> y = dobro(x)
y = 8
```

Funções

- Entre as palavras **function** e **end**
  - As variáveis são locais e estão isoladas
  - A variável **d** dentro da função é independente da variável **d** fora da função

```
function d = dobro(a);
   d = 2*a
end
```

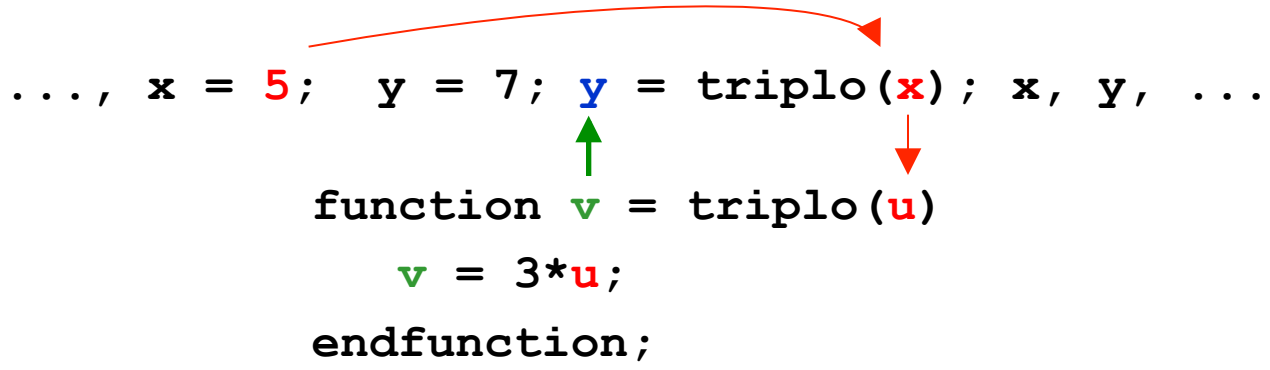
```
octave:13> d = 4
d = 4
octave:14> y = dobro(d)
y = 8
octave:13> d
d = 4
```

Funções

- Existem duas formas "clássicas" de passagem de parâmetros:
  - Por **valor**
  - Por **referência**
- Em Octave, os parâmetros são geralmente passados ***por valor*** (com uma excepção - nomes de funções).
- Podemos analizar este modo de passagem de parâmetros em casos mais complexos.

Passagem de Parâmetros por Valor

- Exemplo: A computação da função triplo pode ser assim explicada:



1. Quando começa a ser executada a função triplo, são criadas duas novas variáveis, **u** e **v**, que são **locais** à função f.
2. O valor inicial da variável (local) **u** é o valor do parâmetro de chamada. Neste caso, é o valor da variável (do programa) **x**.
3. A instrução v = 3*u apenas envolve as variáveis locais **u** e **v**.
4. Após a execução da função, a variável **v**, contém o valor a ser retornado ao programa principal.
5. No programa principal esse valor é atribuído à variável (de programa) **y**.

Passagem de Parâmetros por Valor

- **Exemplo**:

  Consideremos a função **triplo** (que, como o nome indica, calcula o "triplo" do parâmetro de entrada). A sua definição (em Octave) é feita através da especificação

```
function v = triplo(u)
   v = 3*u;
endfunction;
```

- Se chamada com o valor 5, normalmente o valor que uma determinada variável tem na altura da chamada da função, esse valor é triplicado e retornado. Por exemplo, se invocarmos a função na sequência (pode ser ao terminal)

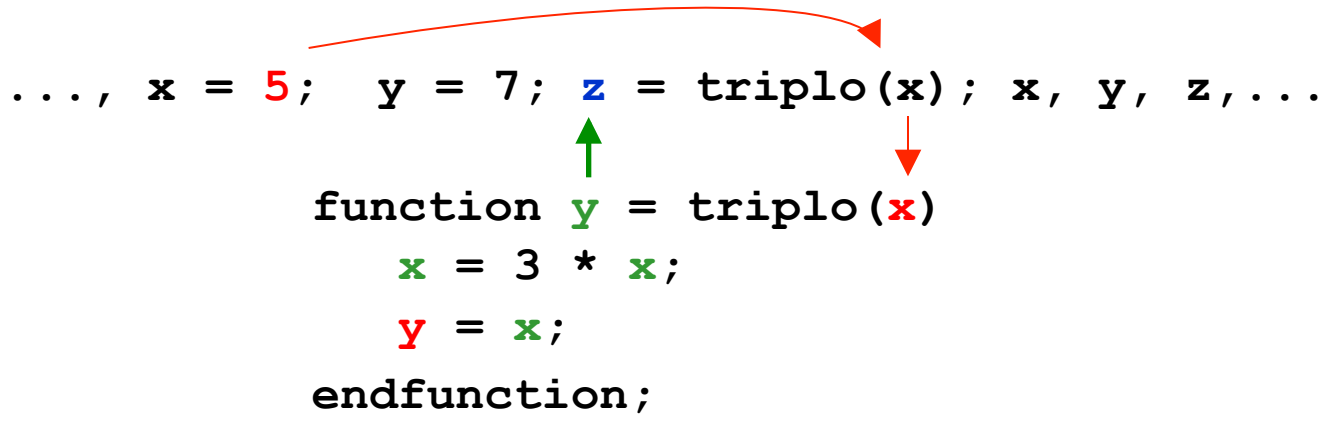
```
x = 5, y = 7, y = triplo(x); x, y,...
```

  os valores de x e y reportados no terminal são

```
x = 5
y = 7
x = 5
y = 15.
```

Passagem de Parâmetros por Valor

NOTA: As variáveis que aparecem na definição da função são **locais** a essa função.



- Num programa grande, com várias funções, é inevitável que variáveis em funções diferentes tenham o mesmo nome (embora os nomes das variáveis devam ser escolhidos para evitar essas "coincidências").

- Assim, elas podem ter o mesmo nome das variáveis de programa, que **não serão confundidas** com elas.(as variáveis **x, y** do programa principal não se confundem com as variáveis **x, y** da função)

x = 5 , y = 7 e **z = 15**.

Passagem de Parâmetros por Referência

- Outras linguagens (Pascal, C, C++, ...) permitem a passagem de parâmetros **por referência**. Neste caso, o que é passado à função é uma **referência** (apontador) à variável do programa principal, que pode ser alterada pela função.
- Por exemplo se o parâmetro x fosse passado por referência (indicado com uma notação fictícia)

```
..., x = 5; z = f(x); x, z, ...

function y = f(ref x)
   x = 2*x;
   y = x;
endfunction;
```

os valores de **x** e **z** reportados no terminal, após a chamada da função, seriam

- x = 10 (sendo passada por referência, a variável x referida na função é a mesma variável que a variável x do programa); e
- z = 10 (como anteriormente, o valor da função é atribuído à variável z do programa principal)

Funções de Funções em Octave

- Uma vez especificada, uma função pode ser utilizada na especificação de outras funções, ou nos próprios parâmetros de entrada de outras ou da mesma função.
- Por exemplo, definidas as funções dobro e triplo, nos respectivos ficheiros dobro.m e triplo.m

Ficheiro dobro.m

```
function y = dobro(x)
   y = 2*x
endfunction
```

Ficheiro triplo.m

```
function y = triplo(x)
   y = 3*x
endfunction
```

- Pode fazer-se a chamada `z = dobro(triplo(5))` ou definir a nova função sextuplo

Ficheiro sextuplo.m

```
function y = sextuplo(x)
   y = dobro(triplo(x))
endfunction
```

Funções Múltiplas em Octave

- A passagem de parâmetros por referência permite que uma função (ou procedimento) passe vários valores para o programa que a invocou. Basta passar por referência as variáveis onde esses valores devem ser "colocados".
- O Octave, **não** suporta passagem de parâmetros por referência. A computação de vários resultados numa função é conseguida pela computação de um vector de resultados.
- Por exemplo, se se pretender que a função **f**, com argumento **x**, retorne dois valores, **y1** e **y2**, especifica-se a função como

```
function [y1,y2] = f(x)
    ...
    y1 = ...;
    y2 = ...;
    ...
endfunction;
```

Funções Múltiplas em Octave

Exemplo:

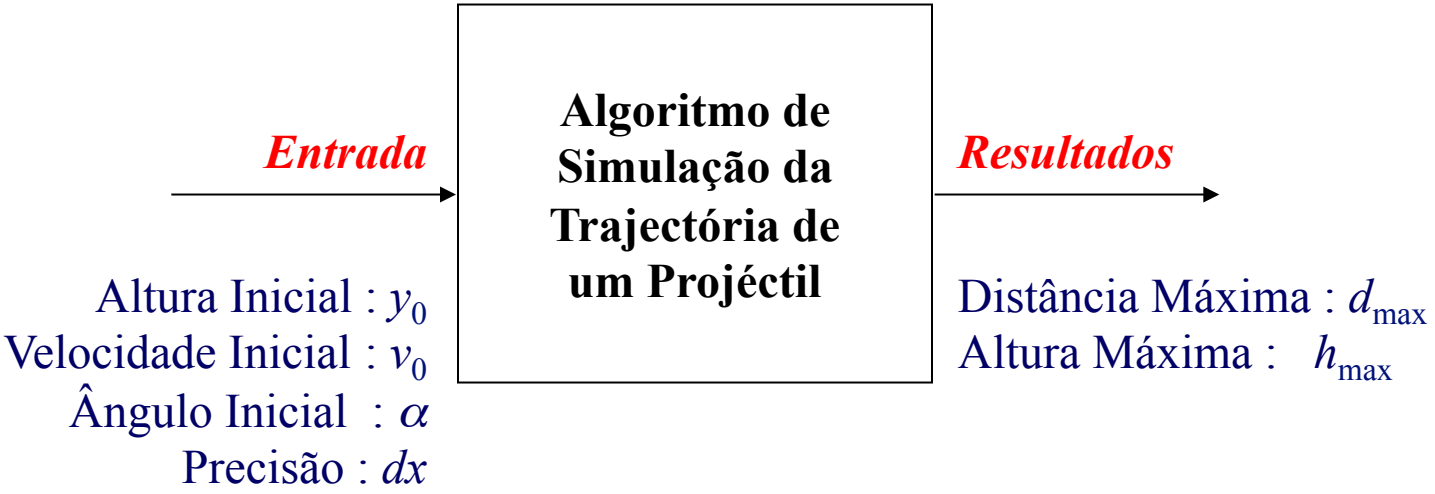
- Para um ângulo alpha (expresso em radianos) obter numa só função os valores do seu seno, coseno, tendo em atenção que

```
function [s,c,t] = trigonom(x)
   s = sin(x);
   c = cos(x);
   t = tan(x);
 endfunction;
```

- Podemos agora exemplificar a utilização de funções na resolução de um problema de engenharia relativo à trajectória de um projéctil.

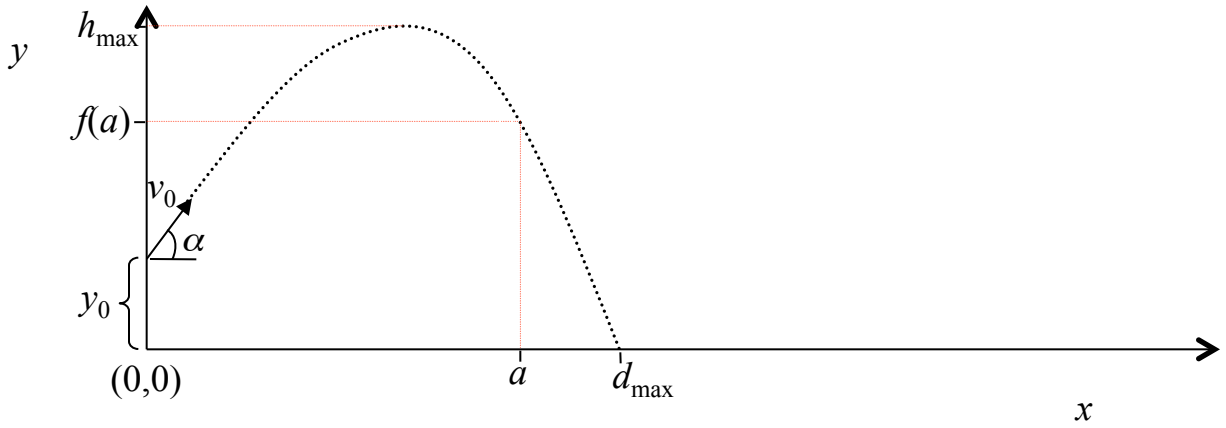
Trajectória de um Projéctil

- Dada uma altura inicial (y_0) uma velocidade inicial (v_0) e um ângulo inicial de lançamento (α), com base no modelo da trajectória apresentado e para uma dada precisão (dx), determinar a distância máxima (d_{max}) e a altura máxima (h_{max}) atingidas pelo projéctil.



Modelação do Problema

- Um projéctil é lançado de uma altura de y_0 metros, com um ângulo inicial de lançamento de α radianos e com uma velocidade inicial de v_0 metros por segundo.



- A trajectória do projéctil em coordenadas (*x*,*y*) pode ser modelada através da seguinte equação:

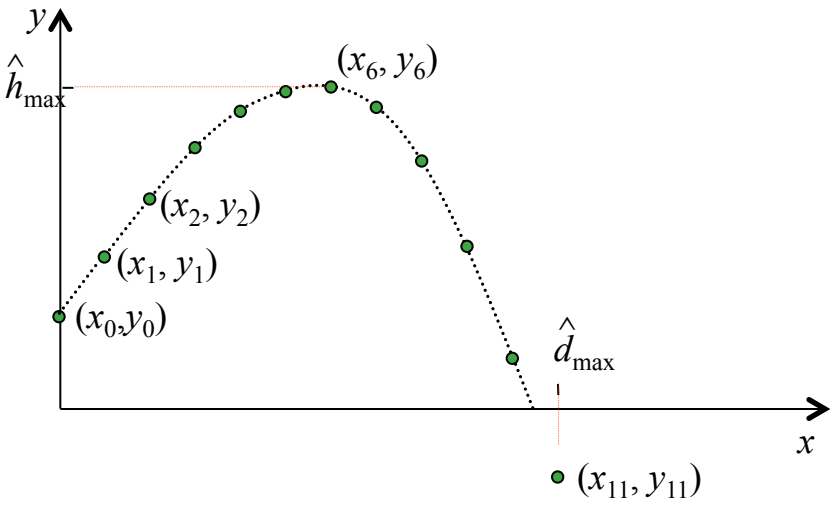
$$f(x) = y = x\tan(\alpha) - \frac{g}{2v_0^2\cos^2(\alpha)}x^2 + y_0$$

Resolução Informal

- **Problema:** determinar a distância máxima (d_{max}) e a altura máxima (h_{max}) atingidas pelo projéctil.
- Pode-se simular a trajectória do projéctil usando a função *f* para calcular o valor de *y* correspondente a cada valor de *x*.
  - Considera-se o ponto inicial da trajectória $x_0 = 0$: (x_0, y_0)
  - Considera-se uma sequência de valores de *x* (x_1, x_2,…) para os quais se calcula o respectivo valor de *y* (y_1, y_2,…) usando a fórmula indicada.
  - Termina-se o cálculo quando aparecer o primeiro ponto da trajectória com o valor de *y* negativo.
- Os valores da distância máxima e da altura máxima podem ser aproximados respectivamente pelos valores maximos de *x* e *y* obtidos nos pontos calculados da trajectória.

Resolução Informal

- A simulação da trajectória do projéctil pode ser simulada por um ciclo em que:



- O último ponto considerado é o primeiro (x_{11}) com *y* negativo: $y_{10} > 0$ e $y_{11} < 0$
- A distância máxima (d_{max}) é aproximada pelo maior valor de x: $d_{max} \approx x_{11}$
- A altura máxima (h_{max}) é aproximada pelo maior valor de y: $h_{max} \approx y_6$
- A precisão das aproximações depende dos pontos da trajectória calculados:
  - se a distância *dx* entre dois pontos consecutivos diminuir, a precisão aumenta

Problemas e Algoritmos

- Uma vez compreendida a especificação de um problema, e obtido um método informal de o resolver, há que especificar um algoritmo formal, que possa vir a ser a base do programa para resolver o problema inicial.

- Para especificar um algoritmo deveremos
  1. Definir quais as variáveis necessárias, bem como o seu significado.
     - As variáveis deverão ser sempre inicializadas antes de ser utilizadas !
  2. Decompor o algoritmo em componentes suficientemente simples para serem programadas facilmente.
     - Tipicamente num algoritmo podem definir-se as seguintes fases
       1. Inicialização de Variáveis
       2. Corpo do algoritmo (que pode ser ainda mais decomposto)
       3. Apresentação de Resultados

Simulação de Trajectórias - Variáveis

- No presente problema de trajectórias podemos identificar as seguintes variáveis, com os correspondentes significados:

Variável	*Valor Inicial*	*Significado*
g	9.8	aceleração da gravidade
yo	entrada	altura inicial
v0	entrada	velocidade inicial
alfa	entrada	ângulo inicial
dx	entrada	precisão desejada
x	0	distância do projéctil num dado instante
y	y0	altura do projéctil nesse dado instante
dmax	0	distância máxima da trajectória
hmax	0	altura máxima da trajectória

$$y = x\tan(\theta) - \frac{g}{2v_0^2\cos^2(\theta)}x^2 + y_0$$

Exemplo de Funções: Altura/4

- Como vimos, podemos determinar a altura y da trajectória de um projéctil através da expressão abaixo.

$$y = x\tan(\alpha) - \frac{g}{2v_0^2\cos^2(\alpha)}x^2 + y_0$$

- Essa determinação pode ser especificada na função altura, definida abaixo e guardada num ficheiro de nome "altura.m"

Ficheiro altura.m

```
function y = altura(x, y0, v0, alfa)
   g = 9.8;
   y = x*tan(alfa)-(g*x^2)/(2*v0^2*cos(alfa)^2)+y0;
endfunction
```

- Uma vez definida, esta função pode ser utilizada para definição doutras funções.

Algoritmo de Simulação

- O corpo do algoritmo corresponde a um ciclo em que vão sendo obtidos valores de x e de y até se obter um valor de y negativo.
- Em simultâneo vão sendo considerados os valores máximos de x e de y, a que corresponderão os valores finais de dmax e hmax

```
enquanto y > 0 fazer
  x ← x + dx;
  y ← altura(x, y0, v0, alfa)
  se hmax < y
        hmax ← y
  fim se;
fim enquanto
dmax ← x % dmax é obtido pelo valor final de x
```

- A apresentação dos resultados neste caso corresponde a escrever os valores de dmax e hmax (no terminal).

```
sai(dmax);
sai(hmax);
```

$$y = x\tan(\theta) - \frac{g}{2v_0^2\cos^2(\theta)}x^2 + y_0$$

Programa Octave

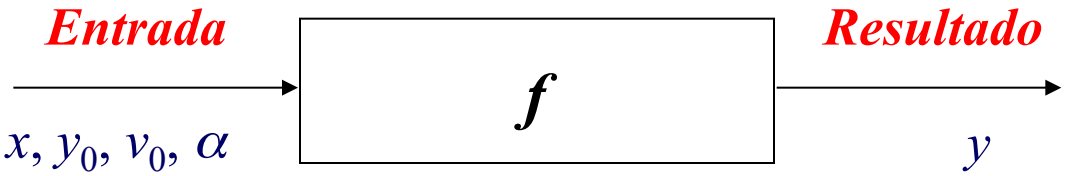
```
% Inicialização de Variáveis
    g    = 9.8;  % aceleração da gravidade
   y0    = input(" Qual a altura inicial (m)? ");
   v0    = input(" Qual a velocidade inicial (m/s)? ");
   alfa  = input(" Qual o angulo inicial (rad)? ");
   dx    = input(" Qual a precisao (m)? ");
   x = 0 ; dmax =  0; % distância máxima da trajectória
   y = y0; hmax = y0; % altura máxima da trajectória

% Ciclo de Simulação
   while y > 0
      x = x + dx;
      y = altura(x, y0, v0, alfa);
      hmax = max(y, hmax);
   endwhile
   dmax = x;

% Apresentação de Resultados
  disp("Distância maxima da trajectoria (m):"); disp(dmax);
  disp("Altura maxima da trajectoria (m):"); disp(hmax);
```

Utilização de Funções

- No exemplo da trajectória de um projéctil, o cálculo do valor de *y* correspondente a cada valor de *x* pode ser abstraído numa função *f* responsável por executar esses cálculos correctamente.



- Mais interessantemente, todo o cálculo da altura e distância máximas de uma trajectória pode ser abstraído numa função *maximos,* que por sua vez utiliza a função *f*.

Exemplo de Funções Múltiplas: Máximos/4

- Um exemplo de função múltipla é a função **maximos**, que determina a distância e a altura máxima da trajectória de um projectil nas condições anteriores, e que é apresentada abaixo

Ficheiro maximos.m

```
function [dmax, hmax] = maximos(y0, v0, alfa, dx)
    x = 0 ; dmax = 0;
    y = y0; hmax = y0;
    while y > 0
       x = x + dx;
       y = altura(x, y0, v0, alfa);
       hmax = max(y, hmax);
    endwhile
    dmax = max(x, dmax);
endfunction
```

Exemplo de Funções Múltiplas em Octave

- Esta função pode agora ser chamada da interface, com vários valores de y0, v0, alfa e dx.

Ficheiro maximos.m

```
function [dmax, hmax] = maximos(y0, v0, alfa, dx)
    ....
endfunction
```

- Por exemplo,

```
>> [d,h] = maximos(0,10,pi/2,0.1)
d = 10.300
h = 2.5510
```

- Outro exemplo

```
>> [d,h] = maximos(10,10,pi/2,0.01)
d = 16.420
h = 12.551
```