

# 5ª Aula - Funções de Intervalo (II).

## Ciclo **for**. Leitura e Escrita em Ficheiros. (I)

### Programação Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério  
sme@tecnico.ulisboa.pt

Departamento de Física  
Instituto Superior Técnico  
Universidade de Lisboa

# Condições (I)

- Como se viu (por exemplo, **Prog05\_02.c**), para certos valores de  $\mu$ , as iterações vão para um '**valor fixo**' muito depressa, nesses casos podia parar-se o cálculo mais cedo.
- Para poder decidir, é necessário usar uma instrução condicional:  
**if (Condição) { ... } else { ... }**
- Os operadores lógicos, usados em **C**, são os seguintes:
  - A **igualdade** faz-se com dois sinais de igual: '**==**';
  - A condição **diferente** ( $\neq$ ) faz-se com: '**!=**';
  - A condição **e** ( $\wedge$ ) faz-se com: '**&&**';
  - A condição **ou** ( $\vee$ ) é feita com: '**||**';
  - As desigualdades fazem-se com: '**<**', '**>**', '**>=**', '**<=**';
  - A **negação** é feita por um ponto de exclamação '**!**';
- Para além das **operações lógicas** é possível ter no lugar da **condição** uma **função**, uma **variável** ou mesmo uma constante.
- O operador **if** interpreta como **falso** o '**0**'. **Tudo o resto é verdadeiro.**

## Condições (II) (Prog05\_05.c)

- Para se poder testar se uma **iteração** é **igual** à **anterior**, é necessário usar uma variável (**x1**) que guarde o valor anterior<sup>1</sup>.

Seja então o ciclo:

```
x1 = x;
while (i <= 20)
{
    printf ("%d: %f\n", i, x);
    x = r * x * (1. - x);
    if (x1 == x)
        break;
    x1 = x;
    ++i;
}
```

- Antes de iniciar o **loop** atribuíamos à variável **x1** o valor inicial de **x** e, a cada iteração, fazemos a sua **atualização**;
- Depois de actualizar a variável **x**, podemos introduzir a expressão condicional que testa a igualdade;
- Se a igualdade for **satisfeita**, queremos parar o **loop**. Tal pode ser feito usando a instrução **break**.

O **break**, ao ser executado, quebra a execução do **loop** e continua a execução do programa na instrução **seguinte** ao final do ciclo.

<sup>1</sup>Note-se que isto só é válido quando os valores são todos iguais.

## Condições (III) (Prog05\_06.c)

- A condição de **igualdade**, vista no exemplo anterior, **só muito lentamente** é obtida, pois, a tendência assintótica pode demorar até atingir a igualdade.
- Por isso, é conveniente fazer uma ligeira alteração à **condição de igualdade** e transformá-la numa *relação de proximidade*;
- Podemos exigir, por exemplo, que a distância entre dois valores seja inferior a  $10^{-6}$ :

$$-10^{-6} < (x_{n+1} - x_n) < 10^{-6}$$

- Podemos então substituir a condição **(x1 == x)** por:  
**((x1 - x) > -1.e-6) && ((x1 - x) < 1.e-6)**

- Mas isto é uma condição sobre **módulo da diferença**:  
**(fabs (x1 - x) < 1.e-6)**

A função de **C**, módulo de um real, é **fabs** e encontra-se definida em **math.h**, assim devemos fazer o include:

```
#include <math.h>
```

## Ciclo com **for** (Prog05\_07.c)

- Até aqui fizemos os ciclos usando a instrução **while**. Como vimos, a sintaxe do **while** é:

```
while (Condição) {Corpo}
```

- Por exemplo, no caso da condição ser sobre um **int** (**int i1**) e só se efectuar enquanto **i1 <= 20**, tem-se

```
i1 = 0; while (i1 <= 20) {... ++i1;
```

- O ciclo do **for** é análogo e tem a seguinte sintaxe:

```
for ( 'Início' ; Condição ; 'Incremento' ) {Corpo}
```

- Assim, o exemplo anterior com a instrução **while** pode ser feito com a instrução **for** do seguinte modo:

```
for ( i1 = 0 ; i1 <= 20 ; ++i1 ) {Corpo}
```

# Função Logística - Órbitas (I)

- Vimos atrás os **gráficos das órbitas** em função do parâmetro **r**.
- Podemos agora fazer um **programa** para calcular essas **órbitas** num certo intervalo  $r \in [r_1, r_2]$ , em que se incrementa o **parâmetro r** de uma quantidade **dr**.
- Para tal temos de fazer um ciclo em que se efectuam, para cada **r**, as seguintes tarefas:
  - 1 Deixar a função **estabilizar**, isto é, calcular previamente um certo número de iterações;
  - 2 Guardar **o valor final** e imprimir os valores seguintes até ele se repetir.
- Para escrever este programa podemos usar ciclos **for** e usar **precisão dupla** (**double**), isto é, 8 bytes em vez dos 4 usados pela **precisão simples** (**float**).

## Função Logística - Órbitas (II) (Prog05\_08.c)

```
for (r = r1 ; r <= r2 ; r += dr)
{
  x = x0;
  for (i1 = 0 ; i1 <= i0 ; ++i1)
    x = r * x * (1. - x);
  x_ref = x;
  for (i1 = 0 ; i1 < imax ; ++i1)
  {
    x = r * x * (1. - x);
    printf ("%f %f\n", r, x);
    if (fabs (x - x_ref) < delta)
      break;
  }
}
```

- Calculam-se as **órbitas periódicas** com  $r \in [r_1, r_2]$  incrementados de **dr**;
- Atribui-se a '**x**' o valor inicial '**x0**';
- Executam-se as '**i0**' iteradas (para estabilizar os valores);
- Toma-se para referência a última iterada '**x\_ref**';
- Para testar a repetição, calculam-se novas iteradas até ao máximo **imax**;
- Imprimem-se o valor de '**r**' e de '**x**';
- Testa-se a diferença entre valores consecutivos com um erro de '**delta**' e, em caso **afirmativo**, pára-se o ciclo com a instrução **break**;

## Função Logística - Órbitas (III) (Prog05\_08.c)

- Se quisermos **visualizar os pontos** obtidos a partir do programa **Prog05\_08.c** num gráfico, em **unix**<sup>2</sup>, podemos guardá-los num ficheiro, utilizando o **redireccionamento** do output:

```
./Prog05_08 > Prog05_08.txt
```

- Depois pode usar-se o programa de gráficos '**gnuplot**' a visualizar o gráfico:

```
gnuplot
```

```
gnuplot> plot "Prog05_08.txt" using 1:2 with dots lt 3
```

Experimentar substituir depois do '**with**' por: points lt 1 pt 9 lw 5

- Para mais indicações ver na página da cadeira:

```
'HowTo - > gnuplot'.
```

---

<sup>2</sup>Este procedimento pode igualmente ser feito em **Microsoft Windows** fazendo a execução do programa directamente na janelas de comandos.

# Como Guardar os Resultados de um Programa?

- Como se viu no programa anterior é escrita no ecrã muita informação que, se não for **devidamente guardada**, se perde.
- Podemos então usar uma facilidade da **shell** de **Unix** que consiste em **redireccionar** o **output** do ecrã para um ficheiro ('>'):  

```
./Prog05_08 > 'ficheiro_onde_escrever.txt'
```
- Note-se que também se pode **redireccionar** o **input** de um programa para um ficheiro através do sinal inverso '<'
- Do ponto de vista do **C**, '**escrever em**' significa abrir um **canal** para o qual é enviada a informação (o mesmo se passa na leitura).
- Quer a **escrita**, quer a **leitura** de **ficheiros**, do **ecrã** ou de **outros locais** corresponde simplesmente a estabelecer um **canal** entre o **programa** que estamos a usar e o **dispositivo** para a qual queremos enviar (ou receber) os **dados**.