# $4^{a}$ Aula - Funções de Intervalo (I).

# Programação
## Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

# Funções de Intervalo (I)

Consideremos a seguinte sequência:

$$x_o = 0.7 \ \rightarrow \ x_1 = x_o^2 = 0.7^2 \ \rightarrow \ x_2 = x_1^2 = x_o^4 = 0.7^4 \ \ldots$$

ou seja, podemos considerar o **termo geral**:

$$x_{n+1} = x_n^2$$

Note-se que esta **relação** satisfaz as seguintes **propriedades**:

1. Se $x_o \in [0, 1] \Rightarrow x_n \in [0, 1]$;
2. Para os valores iniciais '**0**' e '**1**', o resultado é uma repetição daqueles números, então, eles designam-se por **pontos fixos**;
3. Se **começarmos por um valor qualquer** no intervalo $]0, 1[$ as sucessivas iterações irão ser "**atraídas**" para o ponto '**0**'.

# Funções de Intervalo (II)

- Se escrevermos agora uma função ligeiramente diferente:

  $x_o = 0.7 \ \rightarrow \ x_1 = \mu x_o \ \rightarrow \ x_2 = \mu x_1 = \mu^2 x_o \ \ldots \quad \Leftrightarrow \quad \mathbf{x_{n+1} = \mu x_n}$

  irá ser **sobre** o parâmetro $\mu$ que a **repetição** se irá dar. Assim:

  - Se $\mu > 1$, para todo o valor de $\mathbf{x_o} > 0$, a função irá **divergir**;
  - Se $\mu \in [0, 1]$ e se $\mathbf{x_o} \in [0, 1]$ então a função permanecerá sempre no intervalo $[0, 1]$.

- Esta **função** descreve o comportamento de uma **população com crescimento linear** (**Lei de Malthus** - 1798).

**Definição:** **Função de intervalo** é uma aplicação **f** de um intervalo real $[a, b]$ sobre si próprio

$$f \, : [a, b] \rightarrow [a, b]$$

definida pela forma recursiva:

$$x_{n+1} = f(x_n) \qquad \forall \ x_n \in [a, b], \ \ n \in \mathbb{N}_o$$

# Funções de intervalo (III)

- Exemplos de **funções de Intervalo**:
  - $x_{n+1} = \sqrt{\mu\, x_n}$ $\qquad ([0,1] \to [0,1])$
  - $x_{n+1} = r\, x_n^2$ $\qquad ([0,1] \to [0,1])$
  - $x_{n+1} = \cos(\alpha\, x_n)$ $\qquad ([-1,1] \to [-1,1])$

  com **r** e $\boldsymbol{\mu}$ definidos no intervalo [0,1] e $|\boldsymbol{\alpha}| < \infty$.
- Como se viu, no exemplo anterior, os **resultados** dependem crucialmente dos **valores dos parâmetros**;
- Na grande maioria dos casos, o estudo destas **funções** exige a utilização de **cálculo numérico**;
- Deve-se a **Mitchell Feigenbaum** (1944 - ) uma parte significativa dos primeiros trabalhos sobre **funções de intervalo**. Muitos desses cálculos foram feitos com uma simples **máquina de calcular**!

# Funções de intervalo - Lei de Malthus ('Prog03_01a3.c')

- Construamos então o programa, com $\mu = 0.1$, $x_o = 0.7$ e até $x_{20}$:

```
#include <stdio.h>
int main ()
{
 float x ;
 int i ;
 i = 0;
 x = 0.7;
 while (i <= 20)
  {
   printf ("Iter %d: x = %f\n",i,x);
   x = 0.1 * x;
   i = i + 1;
  }
  return 0;
 }
```

- A expressão geral é

$$\mathbf{x_{n+1} = \mu\, x_n},$$

- Declarar a variável **x** e inicializá-la;
- Introduzir um ciclo até $x_{20}$, um contador (**i**) e a condição do loop se realizar, declará-lo, iniciá-lo e incrementá-lo;
- Mostrar os resultados
- Como se disse '**main**' é uma **função** que tem por valor um '**int**' e retornar um valor para a **shell** (linha de comandos).

# Notas sobre C ('Prog04_01.c')

- **Incremento de variáveis** de uma unidade:

  $i = i + 1; \quad \Leftrightarrow \quad$ **++i** ou **i++**

  Exemplos se $\mathbf{i = 20}$:

  printf ("%d\n", **++i**); $\Rightarrow$ **21**
  printf ("%d\n", **i++**); $\Rightarrow$ **20**

  Resultados idênticos se tem para a subtracção com '**- -i**' ou '**i- -**'.

- Para **somar**, **subtrair**, **multiplicar** ou **dividir** uma quantidade, **qt**, a uma variável, **var**, pode escrever-se:

  **var 'operação'**= **qt**;

  Exemplos com $\mathbf{i = 20}$:

  printf ("%d\n", **i += 1**); $\Rightarrow$ **21** $\Leftrightarrow$ **i = i + 1**
  printf ("%d\n", **i −= 5**); $\Rightarrow$ **15** $\Leftrightarrow$ **i = i − 5**
  printf ("%d\n", **i *= 4**); $\Rightarrow$ **80** $\Leftrightarrow$ **i = i * 4**
  printf ("%d\n", **i /= 4**); $\Rightarrow$ **5** $\Leftrightarrow$ **i = i / 4**

# Dinâmica de Populações - Função Logística

- Seja $\mathbf{x}$ a **fracção dos indivíduos** de uma dada espécie em relação ao **número total de indivíduos** num dado ambiente:

  $$\mathbf{x} \in [0, 1]$$

- Se pretendermos calcular a sua **evolução no tempo**, a **dependência mais simples** que podemos considerar é que essa fracção irá ser uma função do valor **fracção** no ano anterior:

  $$\mathbf{x_{n+1} = f(x_n)}$$

- Já vimos que a Lei de Malthus, $\mathbf{x_{n+1} = \mu\, x_n}$, é insuficiente, pois, ou vai para '$\mathbf{0}$' ou vai para '$\infty$';
- Ora sabemos que a fracção nunca poderá ser superior a '$\mathbf{1}$'. Assim, quando $\mathbf{x}$ se aproxima desse valor '$\mu$', deve diminuí-lo:

  $$\mathbf{\mu(x) = r\,(1 - x)} \qquad \Rightarrow \qquad \mathbf{x_{n+1} = r\, x_n\,(1 - x_n)}$$

  a que se dá o nome de **função logística**.
- Para $\mathbf{r} \in [\mathbf{0}, \mathbf{4}]$, a fracção $\mathbf{x_n}$ permanece no intervalo $[\mathbf{0}, \mathbf{1}]$.

# Função Logística ('Prog05_01.c')

- Façamos então um **programa** para calcular a **função logística**:

```
#include <stdio.h>
int main ()
{
 float x, r ;
 int i = 0;
 r = .4 ;
 x = 0.75;
 while (i <= 20)
  {
   printf ("%d: x = %f\n",i++,x);
   x = r * x * (1 - x);
  }
 return 0;
}
```

- Programa é idêntico ao anterior com o ajuste da função;
- Igualmente poderiamos integrar o incremento da variável **i** no comando do **printf**;
- Note-se os sinais **++** foram colocados depois o **i** para que a variável só seja incrementada depois de ser passada para a função **printf**.
- Note-se que a **inicialização** duma variável pode ser simultânea à sua **declaração**;

# Leitura de Dados a Partir da *Shell*

- Vamos alterar o nosso programa (**Prog05_01.c**) para permitir novas leituras (ver **Prog05_02a4.c**). Para permitir a introdução do valor do parâmetro $\mu$, em tempo real, **a partir do teclado**.
- Assim, a linha de inicialização da variável **r**:

  r = 0.4;

  poderá ser substituída por

  **printf**("Escreva, por favor, o valor do parametro 'r': ");
  **scanf** ("**%f**", **&r**);
- A título de **verificação** podemos acrescentar mais uma linha para imprimir no ecran a leitura efectuada:

  **printf** ("Leu o valor %f\n", r);
- Podemos igualmente pedir a para ler a condição inicial $x_o$:

  **printf** ("Escreva, por favor, a condicao inicial de 'x': ");
  **scanf** ("**%f**", **&x**);
- Podemos igualmente juntar as duas leituras.

# Função Logística - Gráficos

Gráficos da função logística para diferentes valores de $\mu$ com $x_o = 0.7$

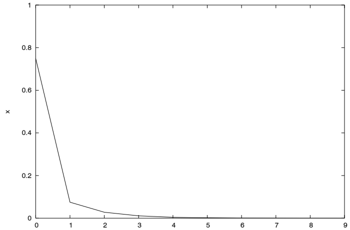


Fig.1 - $\mu = 0.4$

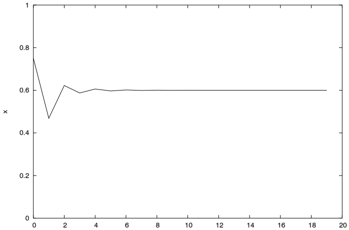


Fig.2 - $\mu = 2.5$

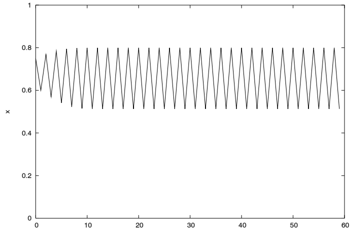


Fig.3 - $\mu = 3.2$

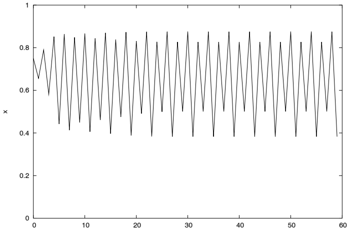


Fig.4 - $\mu = 3.5$

# Função Logística

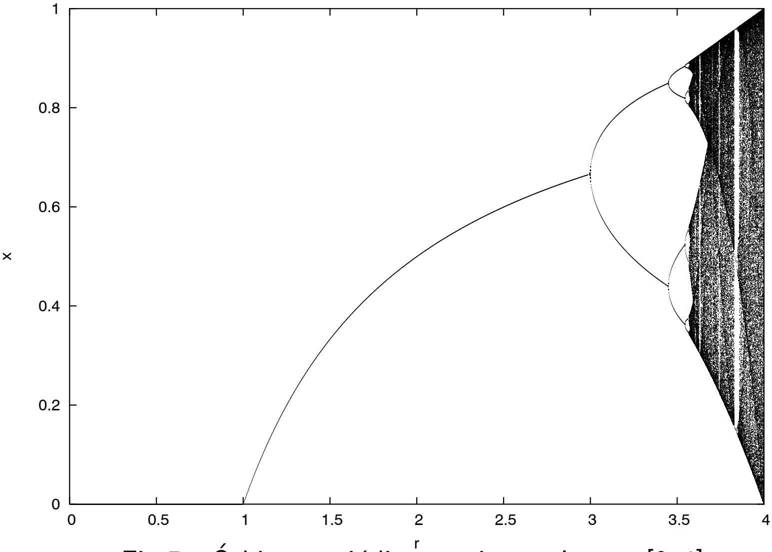


Fig.5 - Órbitas periódicas no intervalo $\mu \in [0, 4]$

# Função Logística

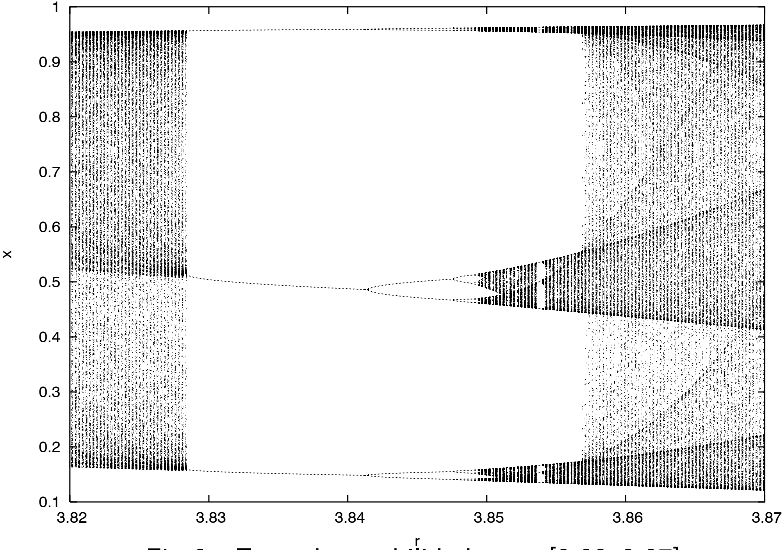


Fig.6 - Zona de estabilidade $\mu \in [3.82, 3.87]$

# Função Logística – Maxima

A visualização dos gráficos anteriores pode ser feita usando o **sistema de computação algébrica** **maxima** (xmaxima). Podem usar-se as seguintes intruções:

```
> xmaxima
(%i1) load ("dynamics");
(%i2) evolution (0.4*x*(1-x), 0.7, 200, [style, lines], [y, 0, 1]);
(%i3) evolution (1.0*x*(1-x), 0.7, 200, [style, lines], [y, 0, 1]);
(%i4) evolution (2.5*x*(1-x), 0.7, 200, [style, dots], [y, 0, 1]);
(%i5) evolution (3.2*x*(1-x), 0.7, 100, [style, dots]);
(%i6) evolution (3.2*x*(1-x), 0.7, 100, [style, lines], [y, 0, 1]);
(%i7) evolution (3.2*x*(1-x), 0.7, 200, [style, points], [y, 0, 1]);
(%i8) evolution (3.5*x*(1-x), 0.7, 2000, [style, dots]);
(%i9) orbits(x*r*(1-x), 0.7, 50, 200, [r, 0, 4], [style, dots]);
(%i10) orbits(x*r*(1-x), 0.7, 50, 200, [r, 3.82, 3.87], [style, dots]);
```