17ª Aula - Métodos Numéricos e Optimização (II)

Programação

Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Queda dos Corpos à Superfície da Terra

- Como se disse, um número muito significativo das **equações diferenciais**, em Física, são de **2ª ordem**.
- O **movimento**, a **uma dimensão** (1-dim), **sem atrito**, **de um corpo**, à **superfície da Terra**, é descrito pela **lei de Newton**:

$$F = -m\,g \quad \Leftrightarrow \quad m\,\frac{d^2x}{dt^2} = -m\,g \quad \Leftrightarrow \quad \frac{d^2x}{dt^2} = -g$$

- Usando as **definições** de **velocidade** e **aceleração**, obtemos

$$\frac{dv}{dt} = -g \qquad\qquad \frac{dx}{dt} = v(t)$$

ou seja, **transformámos** uma equação diferencial de **2ª ordem** num **sistema** de **duas** equações diferenciais de **1ª ordem**, que se podem resolver, como vimos anteriormente.

- Assim, a partir da **primeira** (com $\mathbf{t_o = 0}$):

$$dv = -g\,dt \quad \Leftrightarrow \quad \int_{v_o}^{v} dv = -g\int_{t_o=0}^{t} dt \quad \Leftrightarrow \quad v(t) = -g\,t + v_o$$

- **Substituindo** este resultado na **segunda** e **integrando**:

$$dx = v\,dt \quad \Leftrightarrow \quad \int_{x_o}^{x} dx = \int_0^t v\,dt \quad \Leftrightarrow \quad x(t) = -\frac{1}{2}\,g\,t^2 + v_o\,t + x_o$$

Queda dos Corpos à Superfície da Terra com Atrito

- Infelizmente, a **maioria das equações** não são assim tão **simples de resolver** como o caso anterior.
- Se considerarmos agora uma **força de atrito** proporcional ao **quadrado da velocidade** a 3-dim e a 1-dim:

$$\vec{F}_a = -k\, v(t)^2 \, \frac{\vec{v}}{|\vec{v}|} \qquad ; \qquad F_a = -k\, v(t)^2 \, sgn(v)$$

  em que $sgn(v)$ é a **função sinal**.
- **Equação do movimento** da **queda dos corpos com atrito**:

$$m\, \frac{d^2x}{dt^2} = -m\, g - k\, v(t)^2 \, sgn(v)$$

- Fazendo a sua **decomposição** num **sistema de duas equações de 1ª ordem** e **discretizando** com vista à **resolução numérica**:

$$\frac{\delta v}{\delta t} = -g - \frac{k}{m}\, v^2 \, sgn(v) \qquad\qquad v = \frac{\delta x}{\delta t}$$

- ou seja, usando o **método de Euler**:

$$v(t+\delta t) = v(t) - g\, \delta t - \frac{k}{m}\, v(t)^2 \, sgn(v(t))\, \delta t$$
$$x(t+\delta t) = x(t) + \mathbf{v(t)}\, \delta t$$

Método de Euler e Método de Euler-Cromer ('Prog31_01.c')

- Como se viu, a **discretização** do problema anterior conduziu a:

$$v(t+\delta t) = v(t) - g\,\delta t - \frac{k}{m}\,v(t)^2\,sgn(v(t))\,\delta t$$
$$x(t+\delta t) = x(t) + \mathbf{v}\,\delta t$$

- Uma vez obtido o valor de $\mathbf{v(t+\delta t)}$, a partir da **primeira equação**, temos **dois valores** possíveis da **velocidade** para **substituir** na segunda equação:
  - **Método de Euler**: o **valor da velocidade** calculado no instante '**t**': '$\mathbf{v(t)}$'.
  - **Método de Euler-Cromer**: o **valor da velocidade** calculado no instante '$\mathbf{t+\delta t}$': '$\mathbf{v(t+\delta t)}$'.
- Assim, o sistema anterior, usando o **Método de Euler-Cromer**, toma a forma:

$$v(t+\delta t) = v(t) - g\,\delta t - \frac{k}{m}\,v(t)^2\,sgn(v(t))\,\delta t$$
$$x(t+\delta t) = x(t) + \mathbf{v(t+\delta t)}\,\delta t$$

Método de Euler e Método de Euler-Cromer

- Podemos **sistematizar** os **dois métodos** aqui descritos e analisar as suas **diferenças**. Seja pois a equação diferencial:

  $$\frac{d^2x}{dt^2} = f(x(t), t)$$

- O **método de Euler** conduz ao sistema:

  $$v(t+\delta t) = v(t) + f(x(t), t)\,\delta t$$
  $$x(t+\delta t) = x(t) + v(t)\,\delta t$$

- Enquanto o **método de Euler-Cromer** conduz a:

  $$v(t+\delta t) = v(t) + f(x(t), t)\,\delta t$$
  $$x(t+\delta t) = x(t) + v(t+\delta t)\,\delta t$$

- Se substituirmos $\mathbf{v(t+\delta t)}$ na equação de $\mathbf{x(x+\delta t)}$, obtemos:

  $$x(t+\delta t) = x(t) + v(t)\,\delta t + f(x(t), t)\,\delta t^2$$

- Como se pode ver, esta expressão é igual à do **método de Euler mais** o termo $\delta\mathbf{t^2}$, ou seja, o **método de Euler-Cromer** acrescentar um termo **correctivo quadrático**.

Método Euler, Erros e Problemas **('Prog33_01.c')**

- Os métodos de **Euler** ou de **Euler-Cromer** baseam-se na hipótese da derivada ser **aproximadamente constante** no intervalo $\delta\mathbf{t}$.
- No entanto, em muitas situações, tal aproximação **não é razoavelmente satisfeita** e há **desvios** em relação à solução.
- E os **erros** vão-se **acumulando**...
- Para evidenciar as **diferenças** que surgem entre as **soluções exactas e aproximadas**, podemos analisar a equação:

$$\frac{dx}{dt} = x \qquad \Rightarrow \qquad x(t) = C\, e^t$$

- Pode ver-se no programa **('Prog33_01.c')** a **diferença** entre a solução exacta e a solução aproximada. O **desvio** torna-se **bastante evidente** à medida que se **aumenta** o passo '**dt**'.

Método de Runge-Kutta de 2ª Ordem

- Já vimos que para **melhorar a convergência** das **soluções numéricas** é conveniente procurar **soluções não lineares**.
- Seja a **equação diferencial** e o termo **n** do **método de Euler**:

$$\frac{dx}{dt} = f(x,t) \qquad\qquad x_{n+1} = x_n + f(x_n, t_n)\,\delta t$$

- Procuremos então obter **informação** sobre a **derivada** no **interior do intervalo** para melhorar a **estimativa** do valor da **função**. Seja '**h**' o acréscimo da variável.
- Designemos por $\mathbf{k_1}$ o **termo de Euler**:

$$k_1 = h\,f(x_n, t_n)$$

- Vamos agora fazer uma **estimativa no meio do intervalo**:

$$k_2 = h\,f(x_n + \tfrac{1}{2}\,k_1, t_n + \tfrac{1}{2}\,h)$$

- Podemos então calcular o **valor da função** no final do intervalo:

$$x_{n+1} = x_n + k_2$$

- Estas **três** últimas expressões ($\mathbf{k_1}$, $\mathbf{k_2}$ e $\mathbf{x_{n+1}}$) definem o método de **Runge-Kutta de 2ª ordem**.

Método de Runge-Kutta de 2ª Ordem
Declínio Radioactivo ('Prog30_02.c' e '33_02.c')

- A aplicação do método de **Runge-Kutta de 2ª ordem** ao **declínio radioactivo** dá:

$$\frac{dx}{dt} = -\lambda\, x \qquad \text{em que} \qquad f(x_n, t_n) = -\lambda\, x_n$$

- Para $\mathbf{k_1}$ e $\mathbf{k_2}$ tem-se

$$k_1 = h\, f(x_n, t_n) \qquad k_2 = h\, f(x_n + \tfrac{1}{2}\, k_1, t_n + \tfrac{1}{2}\, h)$$

$$k_1 = -h\, \lambda\, x_n \qquad k_2 = h\, (-\, \lambda\, (x_n + \tfrac{1}{2}\, k_1))$$

- Substituindo $\mathbf{k_1}$ em $\mathbf{k_2}$, obtém-se

$$k_2 = -\lambda\, h\, x_n + \tfrac{1}{2}\, \lambda^2\, h^2\, x_n$$

- E finalmente, tem-se o **termo geral**:

$$x_{n+1} = x_n + k_2$$

$$x_{n+1} = x_n - \lambda\, h\, x_n + \tfrac{1}{2}\, \lambda^2\, h^2\, x_n$$

que é uma expressão de **segunda ordem** em '**h**' (δt).

Método de Runge-Kutta de 4ª Ordem

- O método de **Runge-Kutta de 4ª ordem** é provavelmente o **método** de resolução de **equações diferenciais** mais utilizado, e generaliza o caso anterior (Ver **'Prog33_03.c'**).
- Nesta ordem, o método utiliza **quatro termos**:

$$
\begin{aligned}
k_1 &= h\,f(x_n, t_n)\\
k_2 &= h\,f(x_n + \tfrac{1}{2}\,k_1, t_n + \tfrac{1}{2}\,h)\\
k_3 &= h\,f(x_n + \tfrac{1}{2}\,k_2, t_n + \tfrac{1}{2}\,h)\\
k_4 &= h\,f(x_n + k_3, t_n + h)
\end{aligned}
$$

- O valor da **função** no final do intervalo é então dado por:

$$y_{n+1} = y_n + \tfrac{1}{6}\,k_1 + \tfrac{1}{3}\,k_2 + \tfrac{1}{3}\,k_3 + \tfrac{1}{6}\,k_4$$

  que é uma expressão com **potências** de '**h**' até à **4ª ordem**.
- Outras **variantes deste método** podem ser encontradas na **bibliografia da cadeira**, bem como **programas exemplificativos** das suas implementações.

Precisão
('Prog32_01.c')

- Para além dos **erros** inerentes aos **métodos** temos também os **erros** inerentes às **máquinas** que usamos.
- Como se sabe os **computadores** têm uma **precisão limitada** e a sua **correcta utilização** exige que conheçamos as **características específicas** dos sistemas que usamos.
- A **precisão máxima**, que podemos ter num cálculo, pode ser obtida usando um programa que **adiciona uma quantidade cada vez menor** a uma variável. Quando a variável **não é alterada** atingimos o limite.
- No tocante à **velocidade de cálculo**, é bom ter em conta que muitos **compiladores** fazem internamente o cálculo em '**double**'. Nestas situações, fazer os cálculos em '**float**' poderá conduzir a cálculos **mais lentos** devido aos '**castings**' daí resultantes.

Eficiência de Cálculo

- Em **programas** com um **elevado número de operações** é **essencial** garantir uma **elevada eficiência** das tarefas a executar.
- Em mais do que uma ocasião, já verificámos que a **opção**, por uma ou por outra solução (**algoritmo**), nos conduz a **tempos de cálculos** significativamente **diferentes**.
- Também já se encontraram situações, em que a **memória** requerida pelo programa é **superior** à que lhe pode **ser atribuída**.
- Os **compiladores**, em geral, dispõem de **opções de optimização** que permitem:
  1. Diminuir o **espaço de memória** utilizado;
  2. Diminuir o **tempo de cálculo**.
- No entanto, devemos estar bem cientes de que a **melhor optimização** é a que **fazemos** ao desenvolver um programa.

Eficiência de Cálculo

Algumas **regras básicas** são **essenciais** na **escrita do código** de um programa:

- **Antes** de começar a escrever um **programa** devemos **planificá-lo** cuidadosamente para garantir o menor número de operações;
- Eliminar **operações entre constantes** nos ciclos;
- Usar **macros** para implementar **pequenas funções** de uso muito frequente. Isto aumenta o tamanho do código mas reduz o tempo de acesso às funções;
- **Reduzir** ao mínimo o acesso de **leitura** e de **escrita** no ecran ou no disco;
- Utilizar o qualificativo '**register**' em variáveis de utilização **muito frequente** (nem sempre funciona);
- Ter em **atenção** a estruturação de **ciclos encastrados**.

Eficiência de Cálculo
('Prog32_02.c' a 'Prog32_06.c')

- Os **programas** que se seguem (**Prog32_02a6'**) ilustram as consequências de **diferentes implementações** do código do mesmo **programa básico**. São apresentados os **tempos** de **CPU** e os **tempos** de **Relógio** em segundos:

Programa	Opção de Código	CPU	Relógio
Prog32_02	**Código base sem escrita**	**0.24**	**0**
Prog32_03	**Escrita no ecran**	**31.61**	**68**
Prog32_04	**Escrita num ficheiro**	**5.55**	**6**
Prog32_05	**8./3. → 2.6666**	**0.13**	**0**
Prog32_06	**8./3. → 2.6666 e 'register'**	**0.13**	**0**

Eficiência de Cálculo
('Prog32_07.c' a 'Prog32_09.c')

- Os programas **Prog32_07a9** mostrou as diferenças de **tempos de cálculo** que se obtêm ao usar '**strlen**' na condição de um ciclo:
  1. (**Prog32_07**): usa '**strlen**' na condição de um ciclo;
  2. (**Prog32_08**): usa como condição do ciclo '**str[k] != 0**';
  3. (**Prog32_09**): usa com condição do ciclo '**k < len**', em que '**len**' é o comprimento da string.

Programa	Opção de Código	CPU	Relógio
Prog32_07	**'while (k < strlen(str))'**	**5.99**	**6**
Prog32_08	**'while (str[k] != 0)'**	**1.92**	**2**
Prog32_09	**'while (k < len)'**	**1.92**	**2**