Física Computacional

2020/2021

Universidade de Aveiro

Departamento de Física

Trabalho Prático 3

Métodos de Runge–Kutta
Aplicação de métodos de Runge–Kutta
de 2ª e 4ª ordens e de passo adaptativo

Problema 3.1: Oscilador harmónico — Runge–Kutta de 2ª ordem

Neste exercício deverá aprender a:

- ler tabelas de Butcher ;
- implementar o algorítmo de RK de ordem 2 pela definição;
- implementar o algorítmo de RK de ordem 2 por recurso a funções anónimas (caso geral);
- implementar o algorítmo de RK de ordem 2 por recurso a funções anónimas (simplificado);
- comparar os métodos de RK2 e de Euler, com relação à convergência.

Considere um sistema massa-mola, semelhante ao estudado no Problema 1.1 B do trabalho prático I. Pretende-se estudar o movimento e conhecer a posição $\boldsymbol{x}$ e a velocidade $\boldsymbol{v}$, da massa em função do tempo.

Para o estudo numérico do movimento deste sistema, considere: $x(0) = 1$ m, $v(0) = 0$ m/s, $K = 16$ N/m e $m = 1$ kg.

a) Escreva a equação diferencial que resulta da aplicação da segunda lei de Newton ao problema. Transforme-a num sistema de equações de 1ª ordem.

b) Escreva o algorítmo de Runge-Kutta de ordem dois, apresentado na seguinte **tabela de Butcher**:

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

OBS: Pode encontrar o algoritmo, (já aplicado a uma ODE de segunda ordem), nos slides 11 a 14 da apresentação 3.

c) Usando este algorítmo, integre o sistema de equações obtido até $t_{\text{fin}} = 10$ s, para h =0.01 s, e compare com a solução analítica $x(t)$.

d) Implemente um código para o mesmo algorítmo, por recurso a funções anónimas. Considere o caso geral. Quantas equações obteve? Compare-o com o código obtido em c).

Uso de funções anónimas no MATLAB

Para obter $\boldsymbol{rix}$ e $\boldsymbol{riv}$, com i = 1, 2, tem que escrever duas vezes a expressão de cada uma das funções calculando-a para diferentes valores dos seus argumentos. Se quiser adaptar o programa para outro tipo de oscilador, vai ter que reescrever essas mesmas linhas.

Fazer isto desta maneira dá-nos mais trabalho e mais oportunidades para nos enganarmos!
Nestas situações, torna-se mais fácil usar funções. Uma alternativa seria escrever um ficheiro externo com essa função, mas é mais simples usar o conceito de **função anónima** disponibilizado pelo MATLAB.

Caso Geral

Para este exemplo concreto, na parte inicial do programa, depois de atribuídos os valores às constantes, define-se as funções:

```
fx = @(t,X,V)  V;      %derivada em ordem ao tempo de X

fv = @(t,X,V)  -K*X/m; %derivada em ordem ao tempo de V
```

Depois, dentro do ciclo que calcula a solução numérica, chama-se as funções com os argumentos apropriados a este método:

```
r1x = fx(t(k),x(k),v(k))
r1v = fv(t(k),x(k),v(k));

r2x = fx(t(k)+h/2,x(k)+r1x*h/2,v(k)+r1v*h/2);
r2v = fv(t(k)+h/2,x(k)+r1x*h/2,v(k)+r1v*h/2);
```

(...)

e) Será necessário escrever sempre os códigos na forma anterior? Construa um novo código a partir do anterior, simplificando as equações para o caso particular do problema considerado. Compare o código com os obtidos em c) e d).

Simplificação

No nosso caso particular, **fx** só depende de **V** e **fv** só depende de **X**. Então, podemos escrever:

```
fx = @(V)  V;
fv = @(X)  -K*X/m;

  e
r1x = fx(v(k))
r1v = fv(x(k));
r2x = fx(v(k)+r1v*h/2);
r2v = fv(x(k)+r1x*h/2);
```

(...)

Na escrita de cada função anónima, nomeadamente, **fx** e **fv,** só precisamos de indicar a variável *ou* variáveis, das quais esta depende explicitamente.

E na escrita de cada ***rix*** e ***riv,*** no argumento de **fx** e **fv** , só precisamos de indicar a *ou* as variáveis das quais a função anónima depende explicitamente, bem como a respectiva incrementação.

Torna-se muito mais simples, **mas é preciso muito cuidado para evitar enganos!!!**

f) Compare com os resultados obtidos pelo método de Euler, para um passo h =0.10 s.

Represente a energia total em função do tempo, para ambos os métodos, para os seguintes casos:

i) t_{fin} = 15 s.

Deve observar que a energia, que devia ser constante, cresce com o tempo. Além disso, a segunda derivada do gráfico é positiva, ou seja, a taxa de crescimento de energia também aumenta com o tempo, como se pode ver pela curvatura das linhas.

ii) t_{fin} = 500 s.

iii) t_{fin} = 12000 s.

O que podemos concluir dos resultados desta alínea é que nenhum dos métodos é estável para h = 0.10 s.

Na realidade, para o problema do oscilador harmónico, estes métodos não são estáveis para nenhum valor finito de h.

No caso do método de Runge–Kutta de 2ª ordem isto é mais difícil de observar para valores de h mais pequenos, porque é preciso usar valores bastante maiores de t_{fin} para obter um comportamento comparável com o caso de h = 0.10 s. Para h = 0.01 s, por exemplo, a curvatura do gráfico não é muito clara para t_{fin} = 1000 s, mas já é bastante evidente para t_{fin} = 10000 s. Por outro lado, é fácil de observar o aumento da energia com o tempo.

Problema 3.2: <u>Oscilador harmónico — Runge–Kutta de 4ª ordem</u>

Neste problema vamos repetir o problema anterior, usando um método de Runge-Kutta de 4ªordem.

a) Escreva o algorítmo de Runge-Kutta de quarta ordem, apresentado na seguinte **tabela de Butcher**:

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Consulte os slides correspondentes 14-16 na apresentação 3.

b) Para a implementação do código use funções anónimas. Considere a forma simplificada. Use um espaçamento $h = 0.10$ s, para o qual o método é estável.

c) Aplicado a este problema, o método de Runge–Kutta de 4ª ordem que estamos a usar é estável para $\omega \cdot h < 2\sqrt{2}$, onde $\omega = \sqrt{K/m}$. Experimente valores de h acima e abaixo do limite imposto por este critério e observe os resultados. Note que não basta que o método seja estável para que um dado valor de h seja uma boa escolha (veja o caso de $h = 0.5$ s, por exemplo).

d) Para vários valores de h, sempre abaixo do valor imposto pelo critério de estabilidade, faça um gráfico do logaritmo do módulo do erro numérico em $t = 10$ s em função do logaritmo de h, e confirme que se trata de um método de 4ª ordem.

Problema 3.3: Oscilador harmónico simples — Runge–Kutta de passo adaptativo

Repita a primeira alínea dos problemas anteriores usando a rotina ***ode45*** do Matlab. Esta rotina usa uma função que tem que ser fornecida pelo utilizador num ficheiro diferente e que deve conter as expressões para as derivadas das variáveis dependentes.

Assim, neste caso, ela poderá ter a forma:

```
function derivadas = f(t,solucao)
derivadas = zeros(2,1);
  % Esta linha é necessária e faz do vetor de saída um
  vetor coluna.
```

```
   …
% O vetor solucao tem os valores de x e v para o tempo
% t em que a função é chamada pela rotina ode45.

derivadas(1) = …
% Escreva acima a expressão da derivada de x em função
%de solucao(1) e de solucao(2).

derivadas(2) = …
% Escreva a acima expressão da derivada de v em função
de solucao(1)  e de solucao(2).
```

Se achar que torna o programa mais claro, pode incluir na função as linhas,

```
x = solucao(1);
v = solucao(2);
```

para poder depois escrever **derivadas(1)** e **derivadas(2)** diretamente em função de x e v.

No programa principal, vai ter que usar os seguintes comandos:

```
options = odeset('RelTol',reltol,'AbsTol',[abstol_1 abstol_2]);
[t,sol] = ode45(@f,[t0 tf],[x0 v0],options);
```

Note que no programa principal, o vetor **sol** tem os valores de x e v, um em cada coluna, para todos os tempos t entre t_0 e t_f.

reltol, abstol_1 e abstol_2 são os números que determinam a escolha do passo em cada iteração (leia a documentação do MATLAB). O erro estimado tem que ser menor que o maior dos valores calculados usando os dois critérios: absoluto e relativo.

Note, por exemplo, que se só fosse usado o critério da tolerância relativa, para valores de x e v muito próximos de zero, a ordem de grandeza do último algarismo significativo da estimativa poderia ser excessivamente pequena.

Neste programa, use sempre **reltol = 3E-14** (escolhido porque é praticamente o valor mínimo que o MATLAB aceita) e comece por usar **abstol** iguais a **1E-13**.