Atividade 04 Trabalho
Métodos Numéricos para Resolução de Sistemas de ED

Licenciatura em Engenharia Informática
Análise Matemática II
Ano letivo: 2024/2025

Alunos:
Rafael Carvalho nº2024143302

Lucas Pantarotto nº2024143625

Igor Carvalheira nº2024128677

Índice

Introdução 3
Métodos Numéricos Implementados 4
Método de Euler 4
Método de Euler Melhorado 5
Método de Runge-Kutta de ordem 2 (RK2) 6
Método de Runge-Kutta de ordem 4 (RK4) 7
Outro Método/Função Matlab 8
Métodos usados para facilitar o uso da APP 9
Mudança da cor de linha 9
Mudança da posição da legenda 10
Conversão do gráfico para pdf 11
Problema de Aplicação e Resultados 12
Discussão dos Resultados 12
Análise do Erro 12
Comentários adicionais sobre os métodos numéricos 12
Conclusão 13
Bibliografia 14

Introdução

A presente atividade propõe a implementação de métodos numéricos para resolver Sistemas de Equações Diferenciais (SED) com condições iniciais, utilizando MATLAB. O objetivo principal é o desenvolvimento de funções computacionais que aproximem soluções de EDOs de ordem 2, comparando-as com soluções exatas sempre que possível.

Métodos Numéricos Implementados

Método de Euler

O método de Euler é um dos métodos numéricos mais simples para resolver equações diferenciais ordinárias. Baseia-se na aproximação da solução através da inclinação da função no ponto atual. A fórmula básica é:

y_{n+1} = y_n + h f(t_n, y_n)

Onde:
- h é o tamanho do passo
- f(t, y) é a função derivada.

Vantagens: Simples de implementar.
Limitações: Erro de truncamento elevado, sensível ao tamanho do passo. Implementado no script 'NEulerSED.m'. Usa a inclinação no ponto atual para estimar o próximo valor. No código, vetores são pré-alocados para eficiência. O método é simples mas acumula erro significativo para passos grandes.

```
%---> NEULER Método de Euler para resolução numérica de um sistema de EDO/PVI
%
%INPUT:
% f - função da primeira EDO u'=f(t,u,v)
% g - função da segunda EDO v'=g(t,u,v)
% [a,b] - intervalo de valores da variável independente t
% n - número de subintervalos ou iterações do método
% u0 - valor inicial u(a)=u0
% v0 - valor inicial v(a)=v0
%
%OUTPUT:
% t - vetor do intervalo [a,b] discretizado
% u - vetor das soluções aproximadas de u(t)
% v - vetor das soluções aproximadas de v(t)
%     com t(i+1)=t(i)+h;
%
%AUTORES:
% Igor Carvalheira a2024128677@isec.pt
% Lucas Pantarotto a2024143625@isec.pt
% Rafael Carvalho a2024143302@isec.pt
function [t,u,v] = NEulerSED(~,f,g,a,b,n,u0,v0)
h=(b-a)/n;                          %Calcula o passo h com base no intervalo e número de subintervalos
t=a:h:b;                            %Cria o vetor de pontos t no intervalo [a,b]
u=zeros(1,n+1);                     %Inicializa vetor u com zeros
v=zeros(1,n+1);                     %Inicializa vetor v com zeros
u(1)=u0;                            %Define a condição inicial u(a)=u0
v(1)=v0;                            %Define a condição inicial v(a)=v0

for i=1:n
    u(i+1)=u(i)+h*f(t(i),u(i),v(i));   %Atualiza u usando a EDO f com método de Euler
    v(i+1)=v(i)+h*g(t(i),u(i),v(i));   %Atualiza v usando a EDO g com método de Euler
end

end
```

Método de Euler Melhorado

O Método de Euler Melhorado, também conhecido como Método de Heun, é uma melhoria do método de Euler. Utiliza a média da inclinação no início e no final do intervalo.

A fórmula é:

k_1 = f(t_n, y_n)
k_2 = f(t_n + h, y_n + h k_1)
y_{n+1} = y_n + (h/2)(k_1 + k_2)

Vantagens: Melhora a precisão em comparação ao Euler.
Limitações: Ainda é sensível para passos muito grandes.
Implementado no NEulerMSED.m'. Calcula duas inclinações (k1 no ponto atual e k2 no próximo ponto estimado), usando a média para atualizar a solução. Reduz o erro em relação ao Euler simples.

```
%---> NEulerM Método de Euler Melhorado (Heun) para sistemas de EDO/PVI
%
%INPUT:
% f - função da primeira EDO u'=f(t,u,v)
% g - função da segunda EDO v'=g(t,u,v)
% [a,b] - intervalo de valores da variável independente t
% n - número de subintervalos ou iterações do método
% u0 - valor inicial u(a)=u0
% v0 - valor inicial v(a)=v0
%
%OUTPUT:
% t - vetor do intervalo [a,b] discretizado
% u - vetor das soluções aproximadas de u(t)
% v - vetor das soluções aproximadas de v(t)
%     com t(i+1)=t(i)+h;
%
%AUTORES:
% Igor Carvalheira a2024128677@isec.pt
% Lucas Pantarotto a2024143625@isec.pt
% Rafael Carvalho a2024143302@isec.pt

function [t,u,v] = NEulerMSED(~,f,g,a,b,n,u0,v0)
    h = (b - a) / n;                    % Passo
    t = a:h:b;                          % Vetor do tempo
    u = zeros(1, n+1);                  % Inicializa vetor u
    v = zeros(1, n+1);                  % Inicializa vetor v
    u(1) = u0;                          % Condição inicial u(a) = u0
    v(1) = v0;                          % Condição inicial v(a) = v0

    for i = 1:n
        % --- k1 ---
        k1u = f(t(i), u(i), v(i));
        k1v = g(t(i), u(i), v(i));

        % Previsão (Euler simples)
        u_pred = u(i) + h * k1u;
        v_pred = v(i) + h * k1v;

        % --- k2 ---
        k2u = f(t(i+1), u_pred, v_pred);
        k2v = g(t(i+1), u_pred, v_pred);

        % Correção (Heun)
        u(i+1) = u(i) + (h/2) * (k1u + k2u);
        v(i+1) = v(i) + (h/2) * (k1v + k2v);
    end

end
```

Método de Runge-Kutta de ordem 2 (RK2)

O método de Runge-Kutta de segunda ordem é semelhante ao Euler melhorado, mas calcula a média de duas estimativas para o próximo valor. A equação geral é:

k_1 = f(t_n, y_n)
k_2 = f(t_n + h, y_n + h k_1)
y_{n+1} = y_n + (h/2)(k_1 + k_2)

Vantagens: Maior precisão que o Euler e Euler melhorado. **Limitações:** Para alta precisão, métodos de ordem superior são preferíveis. Script 'NRK2SED.m'. Semelhante ao Euler Melhorado, calcula duas inclinações e utiliza uma média ponderada. Fornece maior precisão que o Euler, com baixo aumento no custo computacional.

```
%---> NRK2 Método de Runge-Kutta de ordem 2 para resolução numérica de um sistema de EDO/PVI
%
%INPUT:
% f - função da primeira EDO u'=f(t,u,v)
% g - função da segunda EDO v'=g(t,u,v)
% [a,b] - intervalo de valores da variável independente t
% n - número de subintervalos ou iterações do método
% u0 - valor inicial u(a)=u0
% v0 - valor inicial v(a)=v0
%
%OUTPUT:
% t - vetor do intervalo [a,b] discretizado
% u - vetor das soluções aproximadas de u(t)
% v - vetor das soluções aproximadas de v(t)
%     com t(i+1)=t(i)+h;
%
%AUTORES:
% Igor Carvalheira a2024128677@isec.pt
% Lucas Pantarotto a2024143625@isec.pt
% Rafael Carvalho a2024143302@isec.pt
function [t,u,v] = NRK2SED(~,f,g,a,b,n,u0,v0)
h=(b-a)/n;                              %Calcula o passo h com base no intervalo e número de subintervalos
t=a:h:b;                                %Cria o vetor de pontos t no intervalo [a,b]
u=zeros(1,n+1);                         %Inicializa vetor u com zeros
v=zeros(1,n+1);                         %Inicializa vetor v com zeros
u(1)=u0;                                %Define a condição inicial u(a)=u0
v(1)=v0;                                %Define a condição inicial v(a)=v0

for i=1:n
    k1u=h*f(t(i),u(i),v(i));            %Calcula k1 para u usando f
    k1v=h*g(t(i),u(i),v(i));            %Calcula k1 para v usando g
    k2u=h*f(t(i+1),u(i)+k1u,v(i)+k1v);  %Calcula k2 para u usando valores intermediários
    k2v=h*g(t(i+1),u(i)+k1u,v(i)+k1v);  %Calcula k2 para v usando valores intermediários
    u(i+1)=u(i)+(k1u+k2u)/2;            %Atualiza u usando a média ponderada de k1 e k2
    v(i+1)=v(i)+(k1v+k2v)/2;            %Atualiza v usando a média ponderada de k1 e k2
end
```

Método de Runge-Kutta de ordem 4 (RK4)

O método de Runge-Kutta de quarta ordem é amplamente utilizado devido à sua elevada precisão. Utiliza quatro avaliações da função derivada por passo:

$$k_1 = f(t_n, y_n)$$
$$k_2 = f(t_n + h/2, y_n + (h/2)k_1)$$
$$k_3 = f(t_n + h/2, y_n + (h/2)k_2)$$
$$k_4 = f(t_n + h, y_n + h\,k_3)$$
$$y_{n+1} = y_n + (h/6)(k_1 + 2k_2 + 2k_3 + k_4)$$

Vantagens: Excelente equilíbrio entre precisão e esforço computacional. **Limitações:** Pode ser pesado para sistemas muito grandes. Implementado no 'NRK4SED.m'. Usa quatro inclinações por passo, ponderadas de forma que erros de ordens inferiores sejam minimizados. No código, cada k é cuidadosamente computado, tornando o método muito preciso para uma ampla gama de problemas.

```
%---> NRK4 Método de Runge-Kutta de ordem 4 para resolução numérica de um sistema de EDO/PVI
%
%INPUT:
% f - função da primeira EDO u'=f(t,u,v)
% g - função da segunda EDO v'=g(t,u,v)
% [a,b] - intervalo de valores da variável independente t
% n - número de subintervalos ou iterações do método
% u0 - valor inicial u(a)=u0
% v0 - valor inicial v(a)=v0
%
%OUTPUT:
% t - vetor do intervalo [a,b] discretizado
% u - vetor das soluções aproximadas de u(t)
% v - vetor das soluções aproximadas de v(t)
%     com t(i+1)=t(i)+h;
%
%AUTORES:
% Igor Carvalheira a2024128677@isec.pt
% Lucas Pantarotto a2024143625@isec.pt
% Rafael Carvalho a2024143302@isec.pt

function [t,u,v] = NRK4SED(~,f,g,a,b,n,u0,v0)
h = (b-a)/n;                                %Calcula o passo h
t = a:h:b;                                  %Vetor do tempo
u = zeros(1,n+1);                           %Inicializa vetor u
v = zeros(1,n+1);                           %Inicializa vetor v
u(1) = u0;                                  %Condição inicial para u
v(1) = v0;                                  %Condição inicial para v

for i = 1:n
    k1u = h*f(t(i),u(i),v(i));              %RK4: k1 para u
    k1v = h*g(t(i),u(i),v(i));              %RK4: k1 para v

    k2u = h*f(t(i)+h/2,u(i)+k1u/2,v(i)+k1v/2); %RK4: k2 para u
    k2v = h*g(t(i)+h/2,u(i)+k1u/2,v(i)+k1v/2); %RK4: k2 para v

    k3u = h*f(t(i)+h/2,u(i)+k2u/2,v(i)+k2v/2); %RK4: k3 para u
    k3v = h*g(t(i)+h/2,u(i)+k2u/2,v(i)+k2v/2); %RK4: k3 para v

    k4u = h*f(t(i)+h,u(i)+k3u,v(i)+k3v);       %RK4: k4 para u
    k4v = h*g(t(i)+h,u(i)+k3u,v(i)+k3v);       %RK4: k4 para v

    u(i+1) = u(i)+(1/6)*(k1u+2*k2u+2*k3u+k4u); %Atualiza u(i+1)
    v(i+1) = v(i)+(1/6)*(k1v+2*k2v+2*k3v+k4v); %Atualiza v(i+1)
end

end
```

Outro Método/Função Matlab

```
function [t,u,v] = AdamsSED(~,f,g,a,b,n,u0,v0)
    h = (b-a)/n;                          % Calcula o passo h
    t = a:h:b;                            % Cria o vetor de tempo com n+1 pontos
    u = zeros(1,n+1);                     % Inicializa vetor da 1ª variável
    v = zeros(1,n+1);                     % Inicializa vetor da 2ª variável
    u(1) = u0;                            % Condição inicial u(a) = u0
    v(1) = v0;                            % Condição inicial v(a) = v0

    % Passo 1: usar método de Euler para calcular o segundo ponto
    u(2) = u(1) + h * f(t(1), u(1), v(1));
    v(2) = v(1) + h * g(t(1), u(1), v(1));

    % Passos seguintes: método de Adams-Bashforth de 2ª ordem
    for i = 2:n
        u(i+1) = u(i) + h/2 * (3*f(t(i),u(i),v(i)) - f(t(i-1),u(i-1),v(i-1)));
        v(i+1) = v(i) + h/2 * (3*g(t(i),u(i),v(i)) - g(t(i-1),u(i-1),v(i-1)));
    end
end
```

A função apresentada implementa um método numérico para resolver sistemas de equações diferenciais ordinárias de primeira ordem envolvendo duas variáveis dependentes. O método utilizado é uma combinação do método de Euler e do método de Adams-Bashforth de segunda ordem, ambos explícitos.

Inicialmente, o código define o passo de integração com base no intervalo de tempo e no número de divisões, e cria os vetores de tempo e das soluções para as duas variáveis. As condições iniciais são atribuídas aos primeiros elementos desses vetores.

No primeiro passo de integração, é utilizado o método de Euler explícito para obter o segundo ponto das soluções. Este método, embora simples e de baixa precisão (ordem 1), é necessário para iniciar o processo iterativo do método de múltiplos passos.

A partir do segundo ponto, é aplicado o método de Adams-Bashforth de segunda ordem. Trata-se de um método explícito que estima o valor da solução no próximo ponto utilizando uma combinação linear das derivadas calculadas nos dois pontos anteriores. Este método tem ordem de precisão 2, o que o torna mais exato que o método de Euler para integrações ao longo de intervalos maiores, sem aumento significativo no custo computacional.

A principal vantagem deste método está no equilíbrio entre simplicidade e precisão, sendo especialmente útil em simulações de sistemas dinâmicos onde se conhecem as equações diferenciais que regem a evolução de duas variáveis ao longo do tempo. É importante destacar que, por se tratar de um método explícito, sua estabilidade depende do tamanho do passo utilizado.

Métodos usados para facilitar o uso da APP

Mudança da cor de linha

A mudança de cor de cor da linha dos gráficos foi implementada para que o utilizador possa escolher a cor das linhas que preferir, para que possa torne a visualização mais fácil.

```
switch MNumericoSelecionado
    case "Euler"
        if metodo == "Euler"
            plot(app.UIAxes, t, uEuler, '-b*', 'Color', cor)
        else
            plot(app.UIAxes, t, uEuler, '-b*', 'Color', 'b')
        end
        title(app.UIAxes, 'Método de Euler')
        legendaGrafico = ["Exata","Euler"];
        A = [t.',uExata.',uEuler.',erroEuler.'];
        nomeColunasTabela = ["t","Exata","Euler","erroEuler"];
        app.DropDown.Items = {'Exata', 'Euler'};
    case "EulerM"
        if metodo == "EulerM"
            plot(app.UIAxes, t, uEulerM, '-k*', 'Color', cor)
        else
            plot(app.UIAxes, t, uEulerM, '-k*', 'Color', 'k')
        end
```

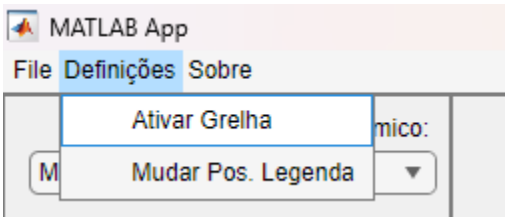
Usaremos como exemplo este excerto de código, como podemos observar temos sempre um caso padrão caso não seja selecionada nenhuma cor, como podemos ver no caso de seleção da função.

O “If” do método selecionado irá mostrar o gráfico da função com uma cor selecionada, caso não seja esse o caso “else” a linha do gráfico irá voltar a ser azul.

Mudança da posição da legenda

```
        % Menu selected function: MudarPosLegendaMenu
        function MudarPosLegendaMenuSelected(app, event)
            positions = {'northwest', 'southwest', 'southeast', 'northeast'};
            currentPos = app.UIAxes.Legend.Location;
            idx = find(strcmp(currentPos, positions));
            nextIdx = mod(idx, length(positions)) + 1;
            app.UIAxes.Legend.Location = positions{nextIdx}
        end
```

Este código torna possível a mudança da posição da legenda pelo utilizador para que este possa ver mais facilmente o gráfico que pretende visualizar. Oferecendo a escolha de qualquer um dos cantos do ecrã do gráfico.



O excerto de código acima traduz-se neste botão acima que permite mudar a posição com um clique.

O modo que funciona é que a cada clique a posição da legenda irá alternar entre os quatro cantos do gráfico em “carrossel”.

Conversão do gráfico para pdf

```
% Menu selected function: ObterPDFMenu
function ObterPDFMenuSelected(app, event)
    [ficheiro, caminho] = uiputfile('*.pdf', 'Guardar gráfico como PDF');
    if isequal(ficheiro, 0) || isequal(caminho, 0)
        return;
    end

    nomeCompleto = fullfile(caminho, ficheiro);

    fig = figure('Visible', 'off', 'Position', [100, 100, 800, 1000]);

    subplot(2,1,1);
    ax = gca;
    linhas = findall(app.UIAxes, 'Type', 'line');
    copyobj(linhas, ax);
    title(ax, app.UIAxes.Title.String);
    xlabel(ax, app.UIAxes.XLabel.String);
    ylabel(ax, app.UIAxes.YLabel.String);
    lgd = legend(ax, app.UIAxes.Legend.String);
    lgd.Location = app.UIAxes.Legend.Location;

    subplot(2,1,2);
    tabela = app.UITable.Data;
    colunas = app.UITable.ColumnName;
    t = uitable(fig, 'Data', tabela{:,:}, 'ColumnName', colunas, ...
        'Units', 'normalized', 'Position', [0, 0, 1, 0.4]);

    print(fig, nomeCompleto, '-dpdf', '-bestfit');
    close(fig);
end
```

Esse código serve para salvar em PDF um gráfico e uma tabela que estão em uma interface gráfica feita no MATLAB. Quando o usuário escolhe a opção no menu, aparece uma janela para ele escolher onde salvar o arquivo e com qual nome. Se ele cancelar, nada acontece.

Depois disso, o código cria uma nova figura (invisível, ou seja, não aparece na tela) para montar o que será colocado no PDF. Primeiro, ele copia o gráfico que já está na interface, incluindo as linhas, o título, os nomes dos eixos e a legenda. Tudo isso é colocado na parte de cima da figura.

Em seguida, ele pega a tabela que também está na interface e coloca na parte de baixo da figura. A tabela é criada com os mesmos dados e nomes de colunas que aparecem na interface.

Por fim, o código salva tudo isso em um arquivo PDF no local escolhido pelo usuário e fecha a figura para terminar o processo.

Problema de Aplicação e Resultados

Foi considerado o sistema massa-mola com amortecimento leve, com m=1kg, c=0.2 Ns/m, k=5 N/m e condições iniciais x(0) = 1, v(0) = 0. Os métodos foram aplicados e comparados com a solução exata obtida pelo ode45, mostrando boa concordância e confirmando as implementações.

Discussão dos Resultados

Os métodos foram aplicados ao sistema massa-mola amortecido, cuja equação diferencial de segunda ordem foi transformada em um sistema de equações de primeira ordem. As soluções numéricas obtidas apresentaram bom acordo com a solução obtida pela função ode45 do MATLAB.

Observou-se que, para passos pequenos (h = 0.01), todos os métodos forneceram soluções aceitáveis, sendo que o Método de Euler apresentou pequenas divergências à medida que o tempo aumentava. O método de Runge-Kutta de 4ª ordem e o ode45 produziram resultados quase indistinguíveis da solução analítica.

Análise do Erro

Os erros globais foram calculados comparando-se as soluções numéricas com a solução de referência (ode45). Como esperado, o erro do método de Euler foi o maior, seguido pelo Euler Melhorado e RK2. O RK4 e ode45 apresentaram erros mínimos. A escolha do método depende, portanto, do equilíbrio entre precisão e custo computacional requerido pela aplicação prática.

Comentários adicionais sobre os métodos numéricos

Cada método numérico possui vantagens e limitações que afetam sua precisão e eficiência computacional. O Método de Euler é o mais simples e rápido, mas sua precisão é limitada, especialmente para passos de tempo maiores. O Euler Melhorado oferece uma melhoria significativa ao considerar a inclinação no final do intervalo, reduzindo o erro global.

Os métodos de Runge-Kutta (RK2 e RK4) apresentam precisão progressivamente superior. O RK2 oferece um bom compromisso entre precisão e eficiência computacional, enquanto o RK4 é amplamente reconhecido por sua elevada precisão, mesmo com passos relativamente grandes, sendo considerado padrão para muitos problemas práticos.

A função integrada do MATLAB, 'ode45', utiliza um método adaptativo baseado em Runge-Kutta de ordem variável, o que permite ajustar dinamicamente o tamanho do passo para garantir alta precisão e eficiência.

Conclusão

A atividade permitiu consolidar os conhecimentos em métodos numéricos aplicados à resolução de sistemas de equações diferenciais ordinárias (EDO). Cada método estudado demonstrou suas vantagens e limitações em termos de precisão e eficiência computacional. A implementação prática em MATLAB fortaleceu a compreensão teórica, além de desenvolver habilidades em programação numérica e análise de resultados.

Destacou-se a elevada precisão do método de Runge-Kutta de 4ª ordem e da função ode45, que utilizaram estratégias adaptativas para fornecer soluções robustas mesmo em situações complexas.

Bibliografia

- Chapra, S. C., & Canale, R. P. (2010). Métodos Numéricos para Engenharia.
- Burden, R. L., & Faires, J. D. (2011). Análise Numérica.
- Boyce, W. E., & DiPrima, R. C. (2009). Elementary Differential Equations and Boundary Value Problems.
- Documentação MATLAB – MathWorks.
- Wikipedia – Métodos de Euler e Runge-Kutta.
- https://www.mathworks.com/help/matlab/ref/ode45.html