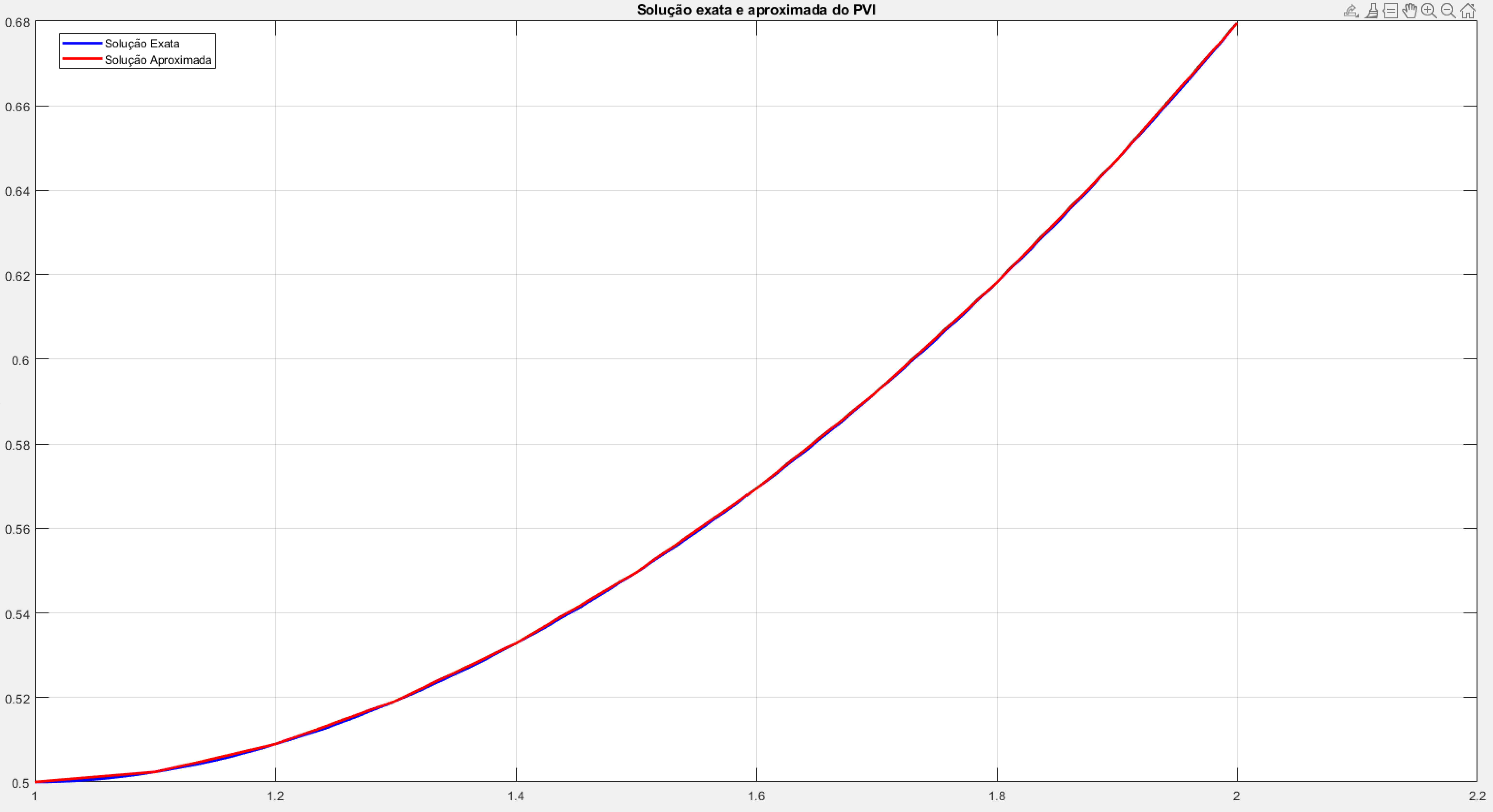
```
% Exercício 5.18. (i) (ii)

function results = metPC4(f, x0, y0, h, N)

    %Inicialização do método com os três primeiros valores
    [x, y] = runge_kutta_4(f, x0, y0, h, 3);
    results = [x, y];

    %AB4-AM4
    for k = 4:N
        %Valores de f
        f_values = [f(results(k-3,1), results(k-3,2));
                    f(results(k-2,1), results(k-2,2));
                    f(results(k-1,1), results(k-1,2));
                    f(results(k,1), results(k,2))];

        %Predição
        y_pred = results(k,2) + h/24 * (55*f_values(4) - 59*f_values(3) + 37*f_values↙
(2) - 9*f_values(1));

        %Correção
        x_next = results(k,1) + h;
        f_next = f(x_next, y_pred);
        y_next = results(k,2) + h/24 * (9*f_next + 19*f_values(4) - 5*f_values(3) +↙
f_values(2));

        %Adicionar ponto
        results = [results; x_next, y_next];
    end
end

function [x, y] = runge_kutta_4(f, x0, y0, h, steps)

    %Método de Runge-Kutta de 4ª ordem
    x = zeros(steps+1, 1);
    y = zeros(steps+1, 1);
    x(1) = x0;
    y(1) = y0;
    for i = 1:steps
        k1 = h * f(x(i), y(i));
        k2 = h * f(x(i) + h/2, y(i) + k1/2);
        k3 = h * f(x(i) + h/2, y(i) + k2/2);
        k4 = h * f(x(i) + h, y(i) + k3);
        y(i+1) = y(i) + (k1 + 2*k2 + 2*k3 + k4) / 6;
        x(i+1) = x(i) + h;
    end

end
```

```
% Exercício 5.19. alíneas a) e b)

%Função do PVI
f = @(x, y) y - y/x;

%Parâmetros iniciais
x0 = 1;
y0 = 0.5;
h = 0.1;
N = 10;

%Solução aproximada
results = metPC4(f, x0, y0, h, N);

%Solução exata
exact_solution = @(x) exp(x - 1) ./ (2 .* x);

%Tabela
fprintf('x_k         y_aprox     y_exato    erro\n');
for i = 1:size(results, 1)
    x_k = results(i, 1);
    y_aprox = results(i, 2);
    y_exato = exact_solution(x_k);
    erro = abs(y_aprox - y_exato);
    fprintf('%f    %f    %f   %e\n', x_k, y_aprox, y_exato, erro);
end


%Gerar pontos exatos para o gráfico da solução exata
x_values = linspace(1, 2, 100);
y_exact_values = exact_solution(x_values);

%Plot da solução exata
figure;
plot(x_values, y_exact_values, 'b', 'LineWidth', 2);
hold on;

%Plot dos resultados aproximados
plot(results(:, 1), results(:, 2), 'r', 'LineWidth', 2);

%Configurações
xlabel('x');
ylabel('y');
title('Solução exata e aproximada do PVI');
legend('Solução Exata', 'Solução Aproximada', 'Location', 'Best');
grid on;
hold off;

% OUTPUT:

% x_k         y_aprox     y_exato    erro
% 1.000000    0.500000    0.500000   0.000000e+00
% 1.100000    0.502350    0.502350   3.422459e-08
% 1.200000    0.508918    0.508918   5.663878e-08
% 1.300000    0.519176    0.519176   7.257900e-08
```

```
% 1.400000    0.532793    0.532795   1.175350e-06
% 1.500000    0.549572    0.549574   1.879694e-06
% 1.600000    0.569410    0.569412   2.360021e-06
% 1.700000    0.592277    0.592280   2.715255e-06
% 1.800000    0.618203    0.618206   2.999424e-06
% 1.900000    0.647261    0.647264   3.244031e-06
% 2.000000    0.679567    0.679570   3.468138e-06
```


Solução exata e aproximada do PVI
Solução Exata
Solução Aproximada
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
0.5
1
1.2
1.4
1.6
1.8
2
2.2

```
% Exercício 5.20. alínea a)

help ode23t
help ode23tb
help ode113

% ode23t - Solve moderately stiff ODEs and DAEs — trapezoidal rule
%    This MATLAB function, where tspan = [t0 tf], integrates the system of
%    differential equations y'=f(t,y) from t0 to tf with initial conditions
%    y0.
%
%    Syntax:
%      [t,y] = ode23t(odefun,tspan,y0)
%      [t,y] = ode23t(odefun,tspan,y0,options)
%      [t,y,te,ye,ie] = ode23t(odefun,tspan,y0,options)
%      sol = ode23t(___)
%
%    Input Arguments:
%      odefun - Functions to solve (function handle)
%      tspan  - Interval of integration (vector)
%      y0     - Initial conditions (vector)
%      options - Option structure (structure array)
%
%    Output Arguments:
%      t  - Evaluation points (column vector)
%      y  - Solutions (array)
%      te - Time of events (column vector)
%      ye - Solution at time of events (array)
%      ie - Index of triggered event function (column vector)
%      sol - Structure for evaluation (structure array)
%
%    Examples:
%      - ODE with Single Solution Component
%      - Solve Stiff ODE
%      - Pass Extra Parameters to ODE Function
%      - Compare Stiff ODE Solvers
%
%    See also ode15s, odeset, odeget, deval
%
%    Introduced in MATLAB before R2006a
%    Documentation for ode23t

% ode23tb - Solve stiff differential equations — trapezoidal rule + backward↙
differentiation formula
%    This MATLAB function, where tspan = [t0 tf], integrates the system of
%    differential equations y'=f(t,y) from t0 to tf with initial conditions
%    y0.
%
%    Syntax:
%      [t,y] = ode23tb(odefun,tspan,y0)
%      [t,y] = ode23tb(odefun,tspan,y0,options)
%      [t,y,te,ye,ie] = ode23tb(odefun,tspan,y0,options)
%      sol = ode23tb(___)
%
%    Input Arguments:
```

```
%      odefun - Functions to solve (function handle)
%      tspan  - Interval of integration (vector)
%      y0     - Initial conditions (vector)
%      options - Option structure (structure array)
%
%    Output Arguments:
%      t  - Evaluation points (column vector)
%      y  - Solutions (array)
%      te - Time of events (column vector)
%      ye - Solution at time of events (array)
%      ie - Index of triggered event function (column vector)
%      sol - Structure for evaluation (structure array)
%
%    Examples:
%      - ODE with Single Solution Component
%      - Solve Stiff ODE
%      - Pass Extra Parameters to ODE Function
%      - Compare Stiff ODE Solvers
%
%    See also ode15s, odeset, odeget, deval
%
%    Introduced in MATLAB before R2006a
%    Documentation for ode23tb

% ode113 - Solve nonstiff differential equations — variable order method
%    This MATLAB function, where tspan = [t0 tf], integrates the system of
%    differential equations y'=f(t,y) from t0 to tf with initial conditions
%    y0.
%
%    Syntax:
%      [t,y] = ode113(odefun,tspan,y0)
%      [t,y] = ode113(odefun,tspan,y0,options)
%      [t,y,te,ye,ie] = ode113(odefun,tspan,y0,options)
%      sol = ode113(___)
%
%    Input Arguments:
%      odefun - Functions to solve (function handle)
%      tspan  - Interval of integration (vector)
%      y0     - Initial conditions (vector)
%      options - Option structure (structure array)
%
%    Output Arguments:
%      t  - Evaluation points (column vector)
%      y  - Solutions (array)
%      te - Time of events (column vector)
%      ye - Solution at time of events (array)
%      ie - Index of triggered event function (column vector)
%      sol - Structure for evaluation (structure array)
%
%    Examples:
%      - ODE with Single Solution Component
%      - Solve Nonstiff Equation
%      - Pass Extra Parameters to ODE Function
%      - ODE with Stringent Error Tolerances
%
```

```
%    See also ode45, ode78, ode89, ode23, odeset, odeget, deval, odextend
%
%    Introduced in MATLAB before R2006a
%    Documentation for ode113
```

```
% Exercício 5.20. alínea b)

%Definir opções
options = odeset('Stats', 'on', 'AbsTol', 1e-10, 'RelTol', 1e-10);

%Definir a equação diferencial
eqdif = @(x, y) y;

%Condições
span = 0:0.2:1;
y0 = 1;

%Aumentar casas decimais
format long g

%Resolução com ode23t
[t1, y1] = ode23t(eqdif, span, y0, options)

%Resolução com ode23tb
[t2, y2] = ode23tb(eqdif, span, y0, options)

%Resolução com ode113
[t3, y3] = ode113(eqdif, span, y0, options)


%OUTPUT:

% 1263 successful steps
% 0 failed attempts
% 1272 function evaluations
% 1 partial derivatives
% 6 LU decompositions
% 1269 solutions of linear systems

% t1 =

%                            0
%                          0.2
%                          0.4
%                          0.6
%                          0.8
%                            1

% y1 =

%                            1
%             1.22140277106064
%             1.49182472912761
%             1.82211885806497
%             2.22554102241997
%             2.71828197181341

% 1057 successful steps
% 0 failed attempts
% 2120 function evaluations
```

```
% 1 partial derivatives
% 3 LU decompositions
% 3174 solutions of linear systems

% t2 =

%                          0
%                        0.2
%                        0.4
%                        0.6
%                        0.8
%                          1

% y2 =

%                          1
%           1.22140276699691
%            1.4918247192337
%           1.82211883995695
%            2.2255409929673
%           2.71828192690487

% 37 successful steps
% 0 failed attempts
% 75 function evaluations

% t3 =

%                          0
%                        0.2
%                        0.4
%                        0.6
%                        0.8
%                          1

% y3 =

%                          1
%           1.22140275815893
%            1.4918246976365
%           1.82211880037949
%           2.22554092843631
%           2.71828182836597
```

```
% Exercício 5.20. alínea c)

%Definir opções
options = odeset('Stats', 'on', 'AbsTol', 1e-10, 'RelTol', 1e-10);

%Definir a equação diferencial
eqdif = @(x, y) y;

%Condições
span = 0:0.2:1;
y0 = 1;

%Aumentar casas decimais
format long g

%Resolução com ode23
[t4, y4] = ode23(eqdif, span, y0, options)

%Resolução com ode45
[t5, y5] = ode45(eqdif, span, y0, options)


%OUPUT:

% 742 successful steps
% 0 failed attempts
% 2227 function evaluations

% t4 =

%                            0
%                          0.2
%                          0.4
%                          0.6
%                          0.8
%                            1

% y4 =

%                            1
%             1.22140275813522
%              1.4918246975803
%             1.82211880027875
%              2.2255409283104
%             2.71828182818125

% 31 successful steps
% 0 failed attempts
% 187 function evaluations

% t5 =

%                            0
%                          0.2
%                          0.4
```

```
%                          0.6
%                          0.8
%                            1

% y5 =

%                            1
%             1.22140275817467
%             1.49182469766572
%             1.82211880042853
%             2.22554092854854
%              2.7182818284885
```

```
% Exercício 5.20. alínea d)

%Definir opções
options = odeset('Stats', 'on', 'AbsTol', 1e-10, 'RelTol', 1e-10);

%Definir a equação diferencial
eqdif = @(x, y) y;

%Condições
span = 0:0.2:1;
y0 = 1;

%Calcular esforços e tempos de CPU
fprintf('\n---------------- ODE23T ------------------\n')
tic;
[t1, y1] = ode23t(eqdif, span, y0, options);
toc;
fprintf('\n---------------- ODE23TB ----------------\n')
tic;
[t2, y2] = ode23tb(eqdif, span, y0, options);
toc;
fprintf('\n---------------- ODE113 ------------------\n')
tic;
[t3, y3] = ode113(eqdif, span, y0, options);
toc;
fprintf('\n----------------- ODE23 ------------------\n')
tic;
[t4, y4] = ode23(eqdif, span, y0, options);
toc;
fprintf('\n----------------- ODE45 ------------------\n')
tic;
[t5, y5] = ode45(eqdif, span, y0, options);
toc;


%OUTPUT:

% --------------- ODE23T -----------------
% 1263 successful steps
% 0 failed attempts
% 1272 function evaluations
% 1 partial derivatives
% 6 LU decompositions
% 1269 solutions of linear systems
% Elapsed time is 0.012994 seconds.

% --------------- ODE23TB ----------------
% 1057 successful steps
% 0 failed attempts
% 2120 function evaluations
% 1 partial derivatives
% 3 LU decompositions
% 3174 solutions of linear systems
% Elapsed time is 0.016316 seconds.
```

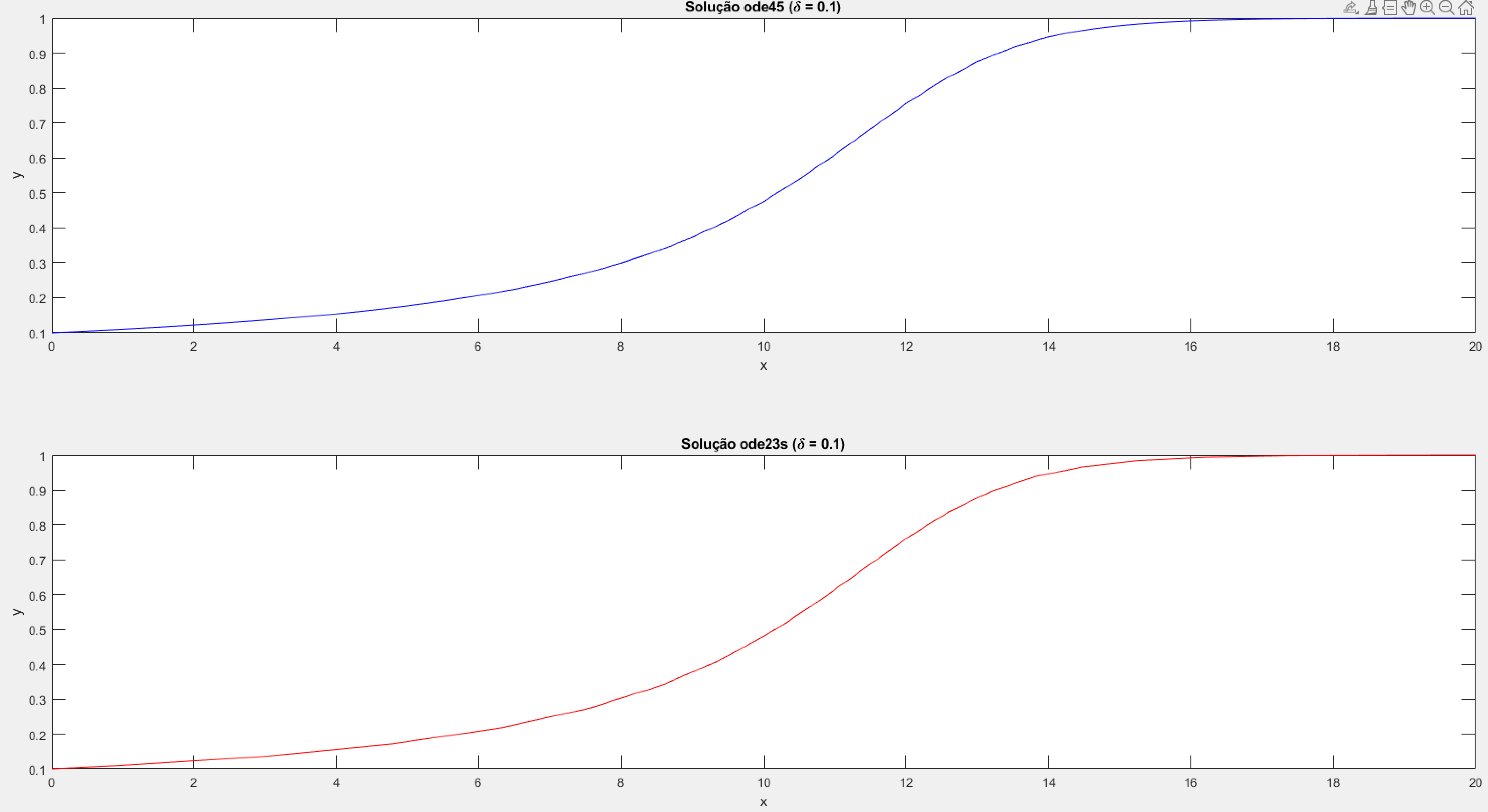
```
% ---------------- ODE113 ------------------
% 37 successful steps
% 0 failed attempts
% 75 function evaluations
% Elapsed time is 0.001820 seconds.

% ---------------- ODE23 ------------------
% 742 successful steps
% 0 failed attempts
% 2227 function evaluations
% Elapsed time is 0.002877 seconds.

% ---------------- ODE45 ------------------
% 31 successful steps
% 0 failed attempts
% 187 function evaluations
% Elapsed time is 0.002707 seconds.
```

```
% Exercício 5.26. alíneas a), c) e d)

edit vdpode;

vdpode(2);
vdpodeRK(2);

vdpode(100);
vdpodeRK(100);

vdpode(200);
vdpodeRK(200);


% Alínea d)

% Pela análise dos gráficos podemos ver que estamos perante um problema
% rígido/stiff. A rigidez ocorre porque as soluções/gráficos apresentam
% regiões onde oscilam lentamente e outras onde oscilam rapidamente.
% Em regiões de oscilação rápida, pequenos passos de tempo são necessários
% para garantir estabilidade, no entanto, em regiões de oscilação lenta,
% passos maiores serão suficientes (mexer na variável tspan).
```

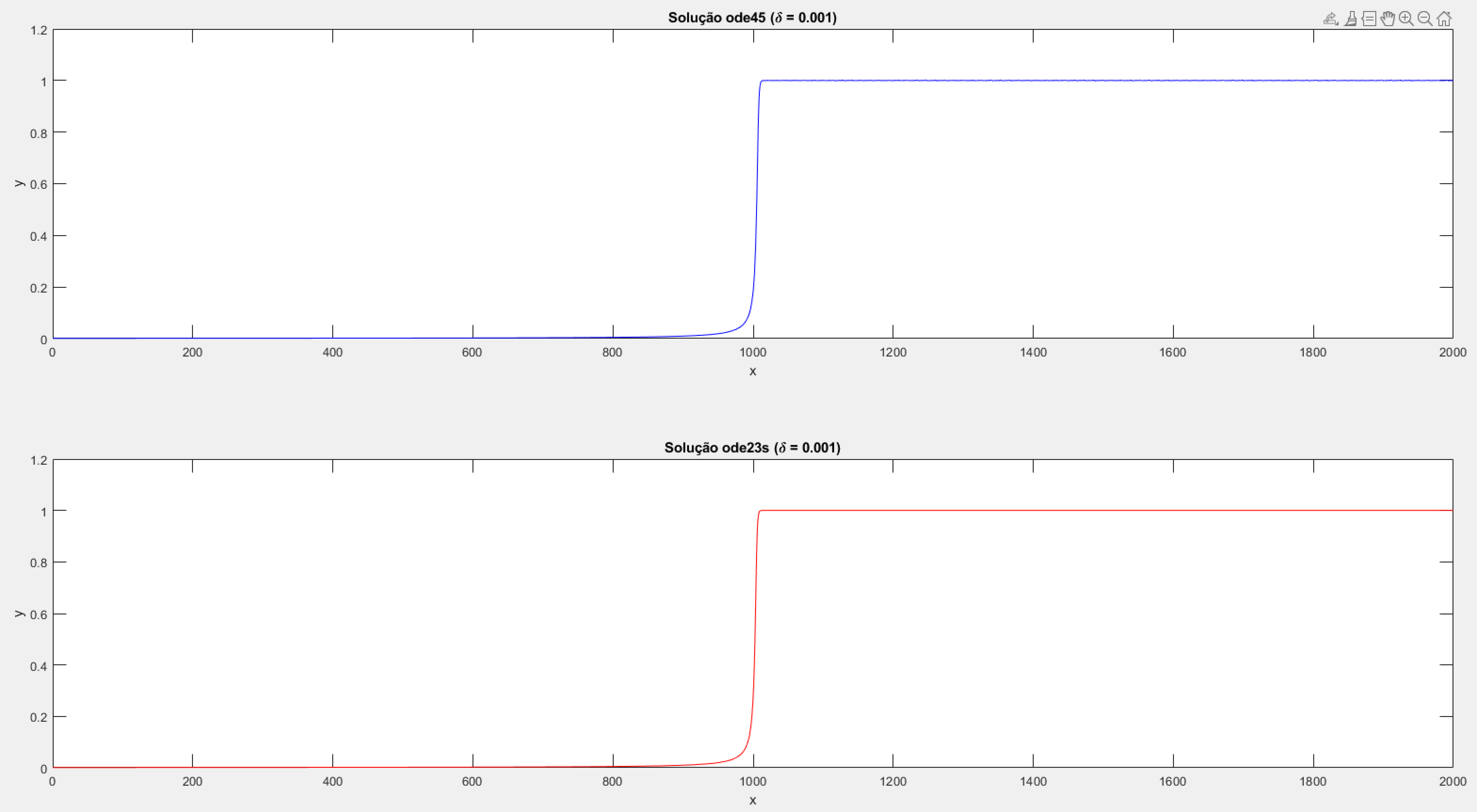
```
% Exercício 5.26. alínea b)

function vdpodeRK(MU)
%VDPODERK  Parameterizable van der Pol equation (stiff for large MU).
%   For the default value of MU = 1000 the equation is in relaxation
%   oscillation, and the problems becomes very stiff.  The limit cycle has
%   portions where the solution components change slowly and the problem is
%   quite stiff, alternating with regions of very sharp change where it is
%   not stiff (quasi-discontinuities). The initial conditions are close to
%   an area of slow change so as to test schemes for the selection of the
%   initial step size.
%
%   The nested function J(T,Y) returns the Jacobian matrix dF/dY evaluated
%   analytically at (T,Y). By default, the stiff solvers of the ODE Suite
%   approximate Jacobian matrices numerically. However, if the ODE Solver
%   property Jacobian is set to @J with ODESET, a solver calls the function
%   to obtain dF/dY. Providing the solvers with an analytic Jacobian is not
%   necessary, but it can improve the reliability and efficiency of
%   integration.
%
%   L. F. Shampine, Evaluation of a test set for stiff ODE solvers, ACM
%   Trans. Math. Soft., 7 (1981) pp. 409-420.
%
%   See also ODE15S, ODE23S, ODE23T, ODE23TB, ODESET, FUNCTION_HANDLE.

%   Mark W. Reichelt and Lawrence F. Shampine, 3-23-94, 4-19-94
%   Copyright 1984-2014 The MathWorks, Inc.

% Problem parameter, shared with nested functions.
if nargin < 1
   MU = 1000;      % default
end

tspan = [0; max(20,3*MU)];    % several periods
y0 = [2; 0];
options = odeset('Jacobian',@J);

[t,y] = ode45(@f,tspan,y0,options);

figure;
plot(t,y(:,1));
title(['Solution of van der Pol Equation with ode45, \mu = ' num2str(MU)]);
xlabel('time t');
ylabel('solution y_1');

axis([tspan(1) tspan(end) -2.5 2.5]);

% ---------------------------------------------------------------------------
% Nested functions -- MU is provided by the outer function.
%

   function dydt = f(t,y)
      % Derivative function. MU is provided by the outer function.
      dydt = [             y(2)
         MU*(1-y(1)^2)*y(2)-y(1) ];
```

```
   end
% ------------------------------------------------------------------------

   function dfdy = J(t,y)
      % Jacobian function. MU is provided by the outer function.
      dfdy = [         0                   1
         -2*MU*y(1)*y(2)-1    MU*(1-y(1)^2) ];
   end
% ------------------------------------------------------------------------

end  % vdpodeRK
```

```
% Exercício 5.27. alínea a)

%Equação Diferencial Ordinária
EDO = @(x, y) y^2 - y^3;

%Condições
delta = 0.1;
span = [0, 2/delta];
y0 = delta;

%Solução ode45
[x_ode45, y_ode45] = ode45(EDO, span, y0);

%Solução ode23s
[x_ode23s, y_ode23s] = ode23s(EDO, span, y0);


figure;

%Gráfico ode45
subplot(2, 1, 1);
plot(x_ode45, y_ode45, 'b-');
title('Solução ode45 (\delta = 0.1)');
xlabel('x');
ylabel('y');

%Gráfico ode23s
subplot(2, 1, 2);
plot(x_ode23s, y_ode23s, 'r-');
title('Solução ode23s (\delta = 0.1)');
xlabel('x');
ylabel('y');

%Número de pontos usados
fprintf('Pontos ode45: %d\n', length(x_ode45));
fprintf('Pontos ode23s: %d\n', length(x_ode23s));

%NOTA: Correr para ver os gráficos

%OUTPUT:

%Pontos ode45: 45
%Pontos ode23s: 21
```


Solução ode45 (δ = 0.1)
y
x
Solução ode23s (δ = 0.1)
y
x

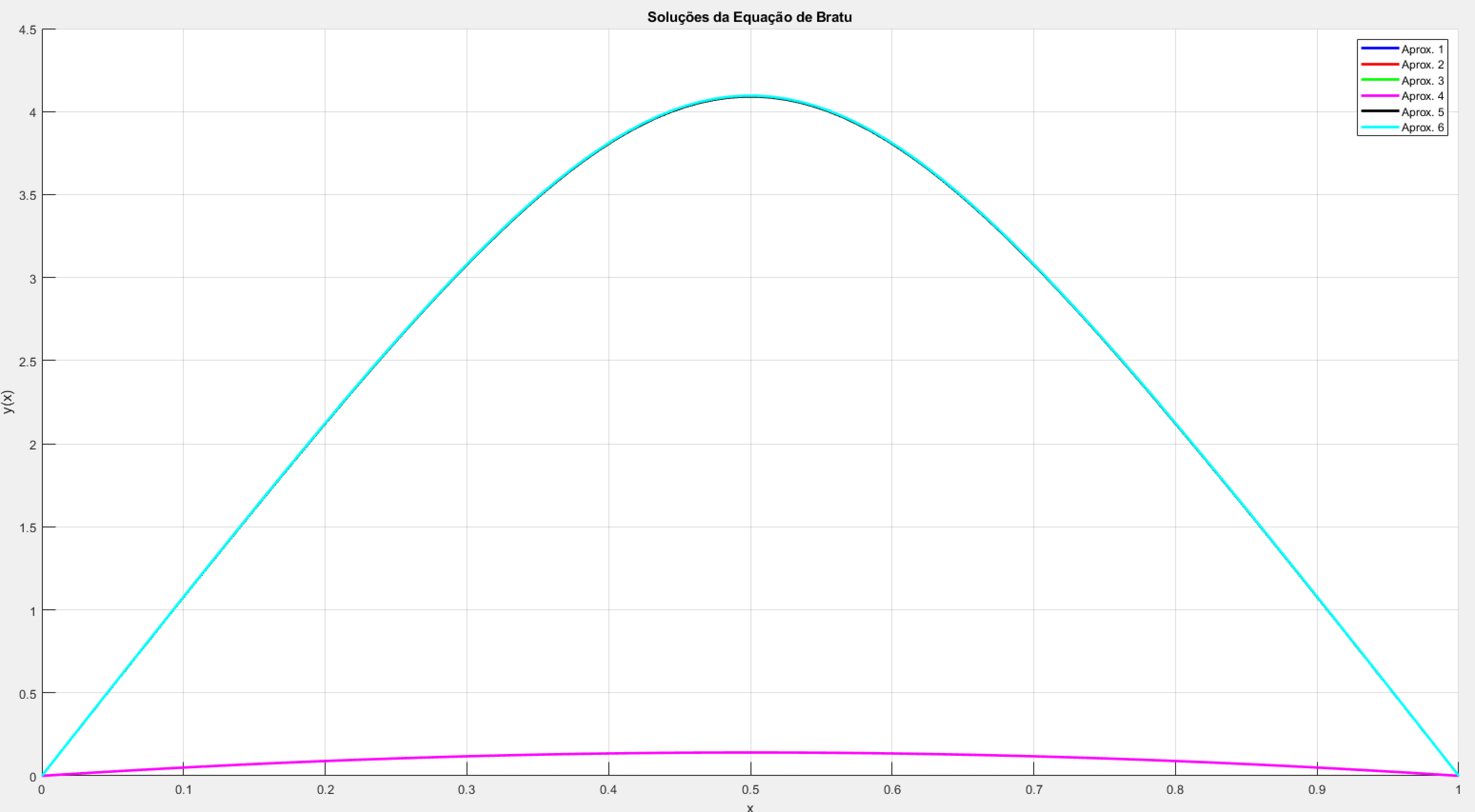
```
% Exercício 5.27. alínea b)

%Equação Diferencial Ordinária
EDO = @(x, y) y^2 - y^3;

%Condições
delta = 0.001;
span = [0, 2/delta];
y0 = delta;

%Solução ode45
[x_ode45, y_ode45] = ode45(EDO, span, y0);

%Solução ode23s
[x_ode23s, y_ode23s] = ode23s(EDO, span, y0);


figure;

%Gráfico ode45
subplot(2, 1, 1);
plot(x_ode45, y_ode45, 'b-');
title('Solução ode45 (\delta = 0.001)');
xlabel('x');
ylabel('y');

%Gráfico ode23s
subplot(2, 1, 2);
plot(x_ode23s, y_ode23s, 'r-');
title('Solução ode23s (\delta = 0.001)');
xlabel('x');
ylabel('y');

%Número de pontos usados
fprintf('Pontos ode45: %d\n', length(x_ode45));
fprintf('Pontos ode23s: %d\n', length(x_ode23s));

%NOTA: Correr para ver os gráficos

%OUTPUT:

%Pontos ode45: 1257
%Pontos ode23s: 47
```


Solução ode45 (δ = 0.001)
y
x
0
0.2
0.4
0.6
0.8
1
1.2
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Solução ode23s (δ = 0.001)
y
x

```
% Exercício 5.27. alínea c)


% Na alínea c), podemos concluir que, à medida que o valor de delta diminui,
% o número de pontos usados pelas funções ode45 e ode23s aumenta.
% Isso indica que a solução da equação diferencial apresenta variações mais
% rápidas para valores de delta menores, exigindo uma maior quantidade de
% pontos para manter a precisão.
% Esse comportamento sugere que, com valores menores de delta
% a solução torna-se mais rígida, possivelmente devido à presença de regiões
% de variação rápida.
% A função ode45, precisa de muitos mais pontos para capturar essas variações
% enquanto ode23s, sendo mais indicado para problemas rígidos, consegue
% resolver o problema com menos pontos.
% Portanto, podemos concluir que o problema se torna mais rígido para valores
% menores de delta, tornando métodos como ode23s mais eficientes do que
% métodos como ode45.
```

```
% Exercício 6.6. alínea a)

help bvp4c;

% OUTPUT:

% bvp4c - Solve boundary value problem — fourth-order method
% This MATLAB function integrates a system of differential equations of
% the form y′ = f(x,y) specified by odefun, subject to the boundary
% conditions described by bcfun and the initial solution guess solinit.

% Syntax
% sol = bvp4c(odefun,bcfun,solinit)
% sol = bvp4c(odefun,bcfun,solinit,options)

% Input Arguments
% odefun - Functions to solve
% function handle
% bcfun - Boundary conditions
% function handle
% solinit - Initial guess of solution
% structure
% options - Option structure
% structure

% Output Arguments
% sol - Solution structure
% structure

% Examples
% Solve Second-Order BVP
% Compare bvp4c and bvp5c Solvers

% See also bvp5c, bvpget, bvpinit, bvpset, bvpxtend, deval

% Introduced in MATLAB before R2006a
% Documentation for bvp4c
```

```matlab
% Exercício 6.6. alínea b)

%Lambda
lambda = 1;

%Equação diferencial ordinária
bratuEDO = @(x, y) [y(2); -lambda * exp(y(1))];

%Condições de fronteira
bratuFronteira = @(ya, yb) [ya(1); yb(1)];

%Aproximações iniciais
aproxs_ini = {
    @(x) [0.1; 0],          %Aproximação inicial alínea a)
    @(x) [3; 0],            %Aproximação inicial alínea b)
    @(x) [1*x.*(1-x); 1*(1-2*x)],  %Aproximação inicial alínea c) k=1
    @(x) [5*x.*(1-x); 5*(1-2*x)],  %Aproximação inicial alínea c) k=5
    @(x) [20*x.*(1-x); 20*(1-2*x)], %Aproximação inicial alínea c) k=20
    @(x) [30*x.*(1-x); 30*(1-2*x)] %Aproximação inicial alínea c) k=30
};

xmesh = linspace(0,1,10);
xplot = linspace(0,1,100);

figure;
hold on;
colors = {'b', 'r', 'g', 'm', 'k', 'c'};
legends = {};


for i = 1:length(aproxs_ini)
    solinit = bvpinit(xmesh, aproxs_ini{i});
    sol = bvp4c(bratuEDO, bratuFronteira, solinit);
    y = deval(sol, xplot);

    plot(xplot, y(1,:), colors{i}, 'LineWidth', 2);
    legends{end+1} = sprintf('Aprox. %d', i);
end

xlabel('x'); ylabel('y(x)');
title('Soluções da Equação de Bratu');
legend(legends);
grid on;


% Como evidenciado no enunciado, a equação de Bratu admite duas soluções
% distintas. No entanto, a convergência para uma dessas soluções depende
% fortemente da aproximação inicial utilizada no método numérico. Ao
% analisar o gráfico das soluções obtidas, observa-se que as aproximações
% 2, 5 e 6 convergiram para uma das soluções, enquanto as restantes
% aproximações foram atraídas para a outra solução. Isso evidencia a
% sensibilidade da equação de Bratu às condições iniciais como seria de
% expectar com o gráfico.
```


Soluções da Equação de Bratu
y(x)
x
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Aprox. 1
Aprox. 2
Aprox. 3
Aprox. 4
Aprox. 5
Aprox. 6