Advances
in Applied Mathematics

An Introduction to Partial Differential Equations with MATLAB®

3rd edition

Chapter 3: Using MATLAB for Solving Differential Equations and Visualizing Solutions

MColeman@fairfield.edu* ∘ VladislavBukshtynov@yahoo.com**

CRC Press
https://www.crcpress.com/

3.1 Visualizing Solutions of ODEs

Review: MATLAB keywords and commands for visualization

`https://www.mathworks.com/help/matlab/getting-started-with-matlab.html`

keyword	description: `example`
`figure`	creates a new figure window: `figure(1)`
`plot`	creates a 2D line plot based on vectors `x` and `y`: `plot(x,y)`
`hold on`	new plots added to the figure do not delete existing plots: `hold on`
`length`	returns the length of a vector: `length(x)`
`linspace`	returns a vector of n evenly spaced points between x_1 and x_2: `linspace(x1,x2,n)`
`function`	declares a function with inputs $x_1, \ldots, x_m$ and outputs $y_1, \ldots, y_n$: `function [y1,y2] = myfun(x1,x2,x3)`
`@`	creates a function handle: `f = @myfunction` or `cube = @(x) x.^3`
`meshgrid`	returns 2D/3D grid coordinates based on vectors `x`, `y`, and `z`: `[X,Y] = meshgrid(x,y)` or `[X,Y,Z] = meshgrid(x,y,z)`
`quiver`	displays velocity vectors as arrows with components (u, v) at points (x, y): `quiver(x,y,u,v)`
`contour`	creates a contour plot of c-isolines: `contour(X,Y,Z,c)`
`surf`	creates a 3D surface plot: `surf(X,Y,Z)`
`mesh`	creates a 3D mesh plot: `mesh(X,Y,Z)`
`imagesc`	displays the data in vector `C` at locations (x, y) using the full range of colors in the specified colormap: `imagesc(x,y,C)`
`colorbar`	displays a vertical colorbar for the current colormap: `colorbar`

3.1 Visualizing Solutions of ODEs (cont'd)

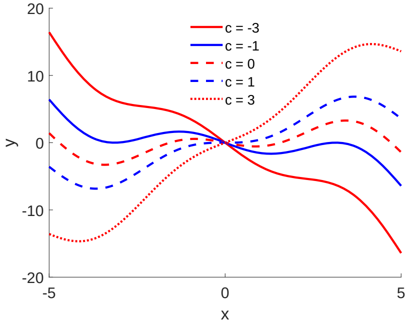
MATLAB: `Chapter_3_visualize_ODEs.m`

Review: n-parameter **family of solutions** for ODE $F\left(x, y, y', y'', \ldots, y^{(n)}\right) = 0$ represented (in general) by functions y given implicitly by

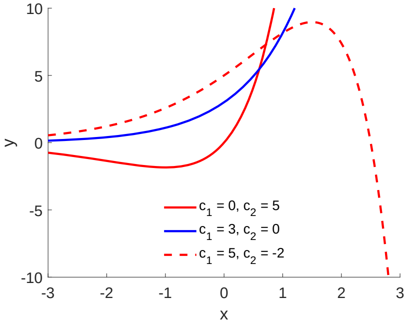
$$G(x, y, c_1, c_2, \ldots, c_n) = 0.$$

Example 1: Visualize solutions for the ODEs:

$$(a)\ xy' - y = x^2 \sin x \qquad \text{and} \qquad (b)\ y'' - 2y' + y = 0.$$



(a) $y = cx - x\cos x$



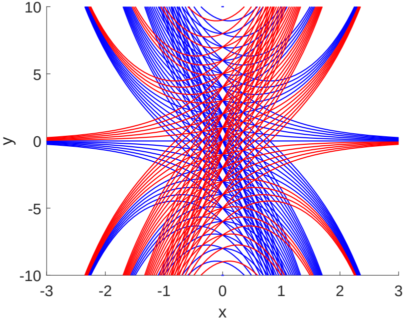
(b) $y = c_1 e^x + c_2 x e^x$

MATLAB: `Chapter_3_visualize_ODEs.m`

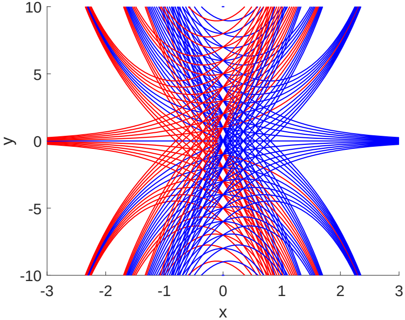
Example 2: Visualize solutions for the second-order ODE

$$y'' - y = 0 \qquad \Rightarrow \qquad y = c_1 e^x + c_2 e^{-x}.$$

Review: for $x > 0$ and $x \to \infty$, this solution is a linear combination of a so-called **steady-state** term e^x and a **transient** term e^{-x}, (the latter $\to 0$ as $x \to \infty$, while the former does not).



$c_1 > c_2$ (red)



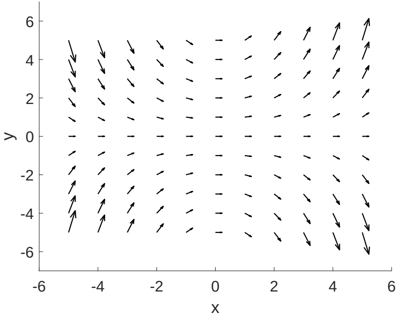
$|c_1| > |c_2|$ (red)

MATLAB: `Chapter_3_visualize_ODEs.m`

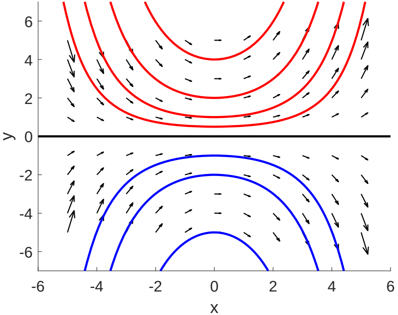
Example 3: Create the direction field for the first-order ODE

$$\frac{dy}{dx} = 0.2xy.$$

Review: We can garner a lot of good information without solving the ODE by plotting direction fields to suggest the shapes of the solution curves $y(x)$ by evaluating the slopes $\frac{dy}{dx} = f(x, y)$ at various points (x, y).



direction field (by command `quiver`)



added solution curves $y(x) = ce^{0.1x^2}$

3.1 Visualizing Solutions of ODEs (cont'd)

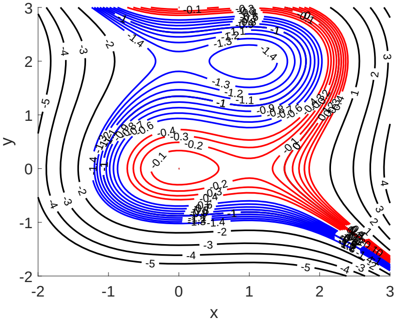
MATLAB: `Chapter_3_visualize_ODEs.m`

Example 4: Visualize (implicit) solutions of the first-order ODE

$$\frac{dy}{dx} = \frac{x(1-x)}{y(y-2)} \qquad \Rightarrow \qquad G(x,y) = \frac{1}{3}y^3 - y^2 + \frac{1}{3}x^3 - \frac{1}{2}x^2.$$

MATLAB plots **level curves** $G(x,y) = c$ with `contour(X, Y, Z, c)`:

- Matrices `X` and `Y` contain various x- and y-values in the mesh/grid `[X, Y]`.
- `Z` contains $G(x,y)$ values for the corresponding x- and y-values in the first two matrices.
- `contour(X, Y, Z, c)` then picks out those values of `Z` which are equal to "height" c, for each chosen value of c, creating a contour plot (level curves or isoclines of `Z`).



3.2 Symbolic Math Toolbox for Solving ODEs

Read more here: `https://www.mathworks.com/products/symbolic.html`

Symbolic computations: meaning analytically, as opposed to numerically or approximately) to perform differentiation, integration, simplification, transforms, and solving various equations, including differential equations.

Example 1: first-order nonlinear **logistic equation**

$$\frac{dP}{dt} = P(a - bP),$$

with some modifications:

- with source term h

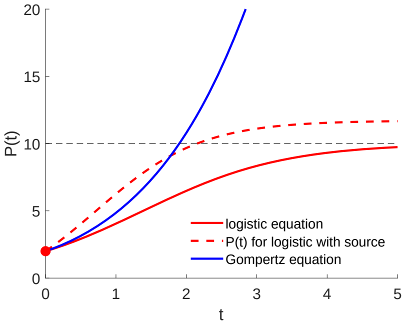
$$\frac{dP}{dt} = P(a - bP) + h,$$

- describing changes due to immigration

$$\frac{dP}{dt} = P(a - bP) + ce^{-kP}, \quad c, k > 0,$$

- and the **Gompertz equation**

$$\frac{dP}{dt} = P(a - b \ln P).$$



3.2 Symbolic Math Toolbox for Solving ODEs (cont'd)

Read more here: **https://www.mathworks.com/help/symbolic/solve-a-single-differential-equation.html**

MATLAB: **Chapter_3_symbolic_math_IVPs.m**

```
disp('(a) logistic equation, solution');
a = 1; b = 0.1; P0 = 2;           % ODE constants
syms P(t);                        % creating symbolic function P(t)
ode = diff(P,t) == P*(a-b*P);     % defining ODE
cond = P(0) == P0;                % setting IC
Psol(t) = dsolve(ode,cond);       % solving IVP using dsolve
Psol = simplify(Psol)             % simplifying solution
tt = 0:0.01:5;                    % time discretization for plotting
P1 = eval(Psol(tt));              % evaluating solution over grid tt
plot(tt,P1,'-r','LineWidth',2); % plotting solution
```

MATLAB output:

```
(a) logistic equation, solution
Psol(t) =
(10*exp(t))/(exp(t) + 4)
```

identical to the solution obtained by hand

$$P_{logistic}(t) = \frac{10e^t}{e^t + 4}.$$

3.2 Symbolic Math Toolbox for Solving ODEs (cont'd)

Example 1 (cont'd): the same goes for other two cases:

```
(b) logistic equation with source, solution
Psol(t) =
5 - 3*5^(1/2)*tanh(atanh(5^(1/2)/5) - (3*5^(1/2)*t)/10)
```

and

```
(d) Gompertz equation, solution
Psol(t) =
exp(exp(-t/10)*(10*exp(t/10) + log(2) - 10))
```

which are exactly the solutions computed analytically:

$$P_{source}(t) = 5 - 3\sqrt{5}\tanh\left[\tanh^{-1}\frac{\sqrt{5}}{5} - \frac{3\sqrt{5}t}{10}\right],$$

$$P_{Gompertz}(t) = \exp\left[e^{-t/10}(10e^{t/10} + \ln 2 - 10)\right].$$

But how about the equation, containing source (immigration) term ce^{-kP}?

```
(c) logistic equation with immigration, solution
Warning: Unable to find explicit solution.
> In dsolve (line 201)
  In Chapter_3_symbolic_math_IVPs (line 44)
Psol(t) =

 [ empty sym ]
```

3.2 Symbolic Math Toolbox for Solving ODEs (cont'd)

Example 2: How about boundary value problems?

$$y'' + \lambda y = 0, \quad 0 < x < 1,$$
$$y(0) = y(1) = 0,$$

subject to the normalization condition

$$y'(0) = 1.$$

MATLAB: `Chapter_3_symbolic_math_BVPs.m`

```
syms y(x) lambda;                     % creating symbolic functions
D = diff(y,x);                        % defining derivative y'(x)
ode = diff(D)+lambda*y == 0;          % defining ODE (2-order)
cond = [y(0) == 0 y(1) == 0 D(0) == 1]; % setting all conditions
Ysol(x) = dsolve(ode,cond);           % solving BVP using dsolve
Ysol = simplify(Ysol)                 % simplifying & displaying
```

Attempt 1: solving as is (with unknown eigenvalue λ and all three side conditions)

```
Warning:  Unable to find explicit solution.
> In dsolve (line 201)
  In Chapter_3_symbolic_math_BVPs (line 24)
Ysol(x) =

 [ empty sym ]
```

3.2 Symbolic Math Toolbox for Solving ODEs (cont'd)

Attempt 2: simplifying the problem (making λ a known constant):

```
clear lambda;
n = 1; lambda = (n*pi)^2;
```

It gives us the same result (probably because the problem was overdetermined)!

Attempt 3: removing the normalization condition

```
cond = [y(0) == 0 y(1) == 0];
```

and we get

```
Ysol(x) =
0
```

Much better! However, it's only the trivial solution $y(x) = 0$ (not an eigenfunction).

Attempt 4: checking the general ability of the toolbox to solve BVPs by solving completely new problem

$$y'' + y = 0, \quad 0 < x < 1,$$
$$y(0) = y(1) = 1,$$

to ensure that MATLAB gives us its unique solution

$$y(x) = \frac{1 - \cos 1}{\sin 1} \sin x + \cos x.$$

3.2 Symbolic Math Toolbox for Solving ODEs (cont'd)

And now it works!

Compare: MATLAB's solutions before and after applying the keyword `simplify`:

```
Ysol(x) =
cos(x) - (sin(x)*(cos(1) - 1))/sin(1)

Ysol(x) =
-(sin(x - 1) - sin(x))/sin(1)
```

MATLAB's Symbolic Math Toolbox works for BVPs, but it requires the solution to exist and to be unique!

Conclusion:

- MATLAB's computational functionality to search for analytical (symbolic) solutions of various ODEs is wonderful!
- However, we recognize its limitations – our knowledge of the theory and solution algorithms is necessary in order to supervise the symbolic computations!

3.3 Solving BVPs Numerically Using `bvp4(5)c`

`bvp4(5)c`: both solve boundary-value problems by employing **Runge–Kutta methods** of the 4th and 5th orders, respectively (subject to given boundary conditions and initial solution guess)

Read more here: `https://www.mathworks.com/help/matlab/ref/bvp4c.html` and `https://www.mathworks.com/help/matlab/ref/bvp5c.html`

MATLAB: syntax (**mandatory**/optional)

sol = bvp4c(**odefun**, **bcfun**, **solinit**, options)
sol = bvp5c(**odefun**, **bcfun**, **solinit**, options)

- **sol**: output solution structure with multiple fields
- **odefun**: function handle that defines the functions to be integrated
- **bcfun**: function handle that defines the boundary conditions (must accept the same number of input arguments as **odefun**)
- **solinit**: initial guess for the solution (we can use the function `bvpinit` to create a **solinit** structure)
- options: some options for setting optional parameters (if omitted, default values are used)

Example: Find all eigenvalues and eigenfunctions of the eigenvalue problem

$$y'' + \lambda y = 0,$$
$$y(0) = y(1) = 0,$$

subject to normalization condition $y'(0) = 1$ or $c_n = \dfrac{1}{n\pi}, \quad n = 1, 2, 3, \ldots$.

MATLAB: Chapter_3_BVPs_bvp4c.m

```
figure(1); hold on;                       % figure #1 (solutions)
figure(2); hold on;                       % figure #2 (errors)
x = 0:0.01:1;                             % x-interval
lType = {'r','b','--r','--b',':r'};       % line types
for n = 1:5
  lambda = (n*pi)^2;                   % initial guess (nth eigenvalue)
  solinit = bvpinit(x,@guess,lambda);     % initial guess (solution)
  sol = bvp4c(@odes,@bcs,solinit);        % solve using bvp4c
  figure(1);                              % plotting y(1) = y(x)
  plot(sol.x,sol.y(1,:),lType{n},'LineWidth',2.5);
  solEx = (1/(n*pi))*sin(n*pi*sol.x);     % computing exact y(x)
  figure(2);                              % creating error plot
  plot(sol.x,abs(sol.y(1,:)-solEx),lType{n},'LineWidth',2.5);
end
```

3.3 Solving BVPs Numerically Using bvp4(5)c (cont'd)

Representing second-order ODE as a system of two first-order ODEs: substitution $u = y'$

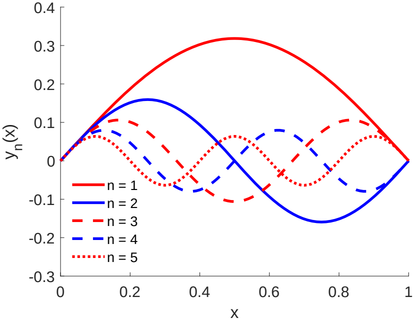
$$y'' + \lambda y = 0 \quad \Longleftrightarrow \quad \begin{aligned} u &= y', \\ u' &= -\lambda y \end{aligned} \qquad \text{or} \qquad \frac{d}{dx}\begin{bmatrix} y \\ u \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\lambda & 0 \end{bmatrix}\begin{bmatrix} y \\ u \end{bmatrix}.$$

MATLAB: `Chapter_3_BVPs_bvp4c.m` (user functions)

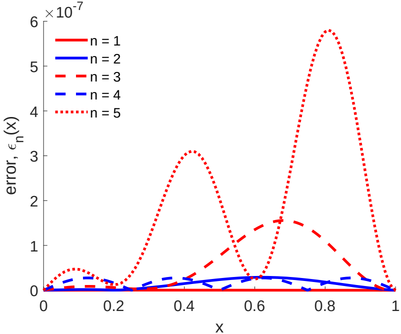
```
% ODE-2 as a system of two ODE-1
function dydx = odes(x,y,lambda)
  dydx = [y(2)              % u = y'
         -lambda*y(1)];     % u' = y'' = -lambda y
end

% boundary conditions
function res = bcs(yl,yr,lambda)
  res = [yl(1)              % y(0)  = 0 (left)
         yr(1)              % y(1)  = 0 (right)
         yl(2)-1];          % y'(0) = 1 (left, normalization)
end

% initial guess (specific to problem)
function g = guess(x)
  g = [sin(x)               % y(x)
       cos(x)];             % y'(x)
end
```

3.3 Solving BVPs Numerically Using `bvp4(5)c` (cont'd)

Visualizing results: eigenfunctions vs. (absolute) error functions $\epsilon_n(x) = |y_n(x) - y_n^{ex}(x)|$



eigenfunctions $y_n(x)$



error plot

"Normalization" process: any constant multiple of eigenfunction $y_n(x)$ is an eigenfunction $\Rightarrow$ normalization condition $y'(0) = 1$ sets all eigenfunctions with slope $y' = 1$ at the left end $x = 0$. In general, we may choose any condition that does not contradict existing requirements and allows us to identify c_n uniquely.

Review: structure of MATLAB script `Chapter_1_bvp4c_eigenproblem.m` (in Chapter 1)

3.4 Solving PDEs Numerically Using pdepe

As of now, **there is no official MATLAB tool that deals with PDEs symbolically**.

There is only one function, `pdepe`, for solving PDEs numerically (only for selected equations in two independent variables).
Read more on pdepe: `https://www.mathworks.com/help/matlab/ref/pdepe.html`

Example: heat equation

$$u_t = u_{xx},$$
$$u(x,0) = 7\cos\frac{5x}{2},$$
$$u_x(0,t) = u(\pi,t) = 0.$$

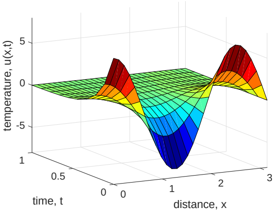
Analytical solution

$$u(x,t) = 7e^{-25t/4}\cos\frac{5x}{2}.$$

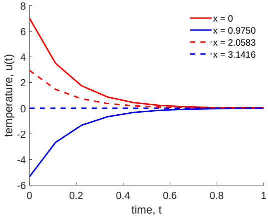
MATLAB: excerpt from `Chapter_3_PDEs_pdepe.m` for plotting using `surf` and `mesh`

```
x = linspace(0,pi,30);              % discretizing x-interval
t = linspace(0,1,10);               % discretizing t-interval
[X,T] = meshgrid(x,t);              % creating (x,t)-grid
uAn = 7*exp(-25*T/4).*cos(5*X/2);   % solution fn on (x,t)-grid
figure(1); surf(X,T,uAn);           % surface plot using surf
figure(2); mesh(X,T,uAn);           % mesh plot using mesh
```

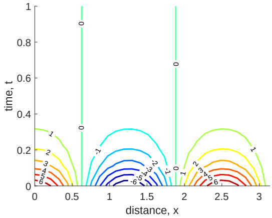
Visualizing results: using different methods



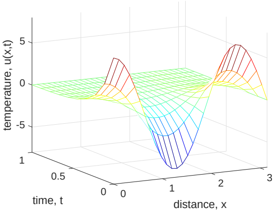
by `surf`



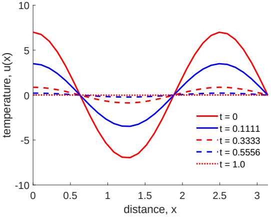
$u(x,t)$ for fixed x



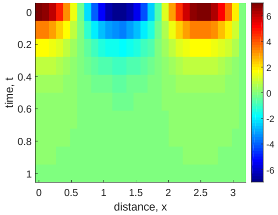
2D plot using `contour`



by `mesh`



$u(x,t)$ for fixed t



2D plot using `imagesc`

Read more: `https://www.mathworks.com/help/matlab/2-and-3d-plots.html`

3.4 Solving PDEs Numerically Using pdepe (cont'd)

MATLAB: pdepe to solve two-variable parabolic and elliptic equations of the form:

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right)\right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right)$$

MATLAB: syntax (**mandatory**/optional)

sol = pde(**m**, **pdefun**, **icfun**, **bcfun**, **xmesh**, **tspan**, options)

- **sol**: output solution structure with multiple fields
- **m**: symmetry constant ($m = 0$ for 1D Cartesian coordinates with no symmetry, $m = 1$ for 2D cylindrical, and $m = 2$ for 3D spherical coordinates, with symmetry)
- **pdefun**: function handle to define the coefficients c, f, and s of the PDE as functions of x, t, u, and $\frac{\partial u}{\partial x}$
- **icfun**: function handle to define the initial condition
- **bcfun**: function handle to define the boundary conditions
- **xmesh**: spatial mesh given as a vector $[x_0\ x_1\ \ldots\ x_n]$ specifying points where a numerical solution is requested for every value in **tspan**
- **tspan**: time span of integration given as a vector $[t_0\ t_1\ \ldots\ t_f]$ specifying points where a numerical solution is requested for every value in **xmesh**
- options: various options for setting optional parameters (if omitted, default values are used)

3.4 Solving PDEs Numerically Using pdepe (cont'd)

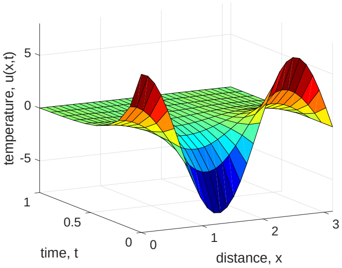
MATLAB: excerpt from Chapter_3_PDEs_pdepe.m

```
x = linspace(0,pi,30);             % discretizing x-interval
t = linspace(0,1,10);              % discretizing t-interval
m = 0;                             % 1D case with no symmetry
u = pdepe(m,@heatEqn,@ic,@bcs,x,t); % solving PDE
surf(x,t,u);                       % plotting solution

return

% specify c, f, s to define PDE: 1*u_t = d/dx(du/dx) + 0
function [c,f,s] = heatEqn(x,t,u,dudx)
  c = 1;
  f = dudx;
  s = 0;
end

% boundary conditions
function [pl,ql,pr,qr] = bcs(x1,ul,xr,ur,t)
  pl = 0;  ql = 1; % left:  0 + 1*dudx = 0, i.e.,  u_x(0,t) = 0
  pr = ur; qr = 0; % right: u + 0 = 0,      i.e.,  u(pi,t)  = 0
end

% initial condition
function value = ic(x)
  value = 7*cos(5*x/2);
end
```

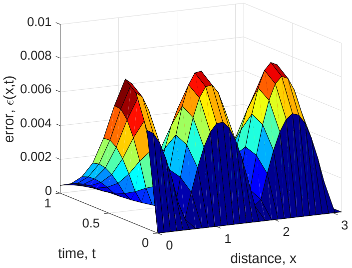
Format for boundary conditions:

$$p(x,t,u) + q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0$$

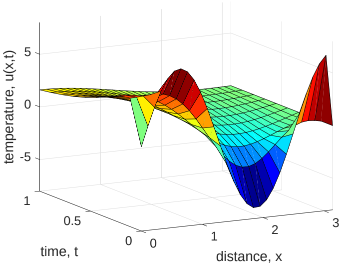
3.4 Solving PDEs Numerically Using pdepe – Visualizing Results



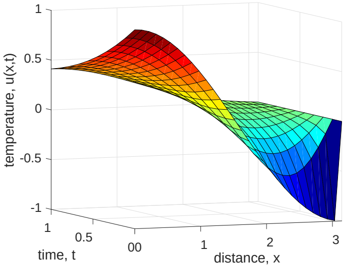
numerical solution $u(x,t)$



error plot



IC: $u(x,0) = 7\sin\frac{5x}{2}$



IC: $u(x,0) = \cos x$