Practical MATLAB Modeling with Simulink

Programming and Simulating Ordinary and Partial Differential Equations

Sulaymon L. Eshkabilov

Apress®

Practical MATLAB Modeling with Simulink

Programming and Simulating Ordinary and Partial Differential Equations

Sulaymon L. Eshkabilov

Apress®

Practical MATLAB Modeling with Simulink: Programming and Simulating Ordinary and Partial Differential Equations

Sulaymon L. Eshkabilov
Ag & Biosystems Engineering Department, North Dakota State University, Fargo, USA

ISBN-13 (pbk): 978-1-4842-5798-2 ISBN-13 (electronic): 978-1-4842-5799-9
https://doi.org/10.1007/978-1-4842-5799-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image from rawpixel.com

Distributed to the book trade worldwide by Springer Science+Business Media, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail editorial@apress.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484257982. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

To the memory of my father.

To my mother.

To my wife, Nigora, after 25 wonderful years together.

Table of Contents

About the Author

Dr. Sulaymon L. Eshkabilov is currently a visiting professor in the Department of Agriculture and Biosystems Engineering at North Dakota State University. He obtained his ME diploma from Tashkent Automobile Road Institute in 1994, his MSc from Rochester Institute of Technology, USA in 2004, and his PhD from Academy Sciences of Uzbekistan in 2005. He was an associate professor at Tashkent Automobile Road Institute from December 2006 to January 2017. He also held visiting professor and researcher positions at Ohio University from 2010 to 2011 and Johannes Kepler University, Austria, from January to September 2017. He has taught the following courses: "MATLAB/Simulink Applications for Mechanical Engineering and Numerical Analysis" and "Modeling of Engineering Systems" for undergraduate students, an "Advanced MATLAB/Mechatronics" seminar/class, and "Control Applications," "System Identification," "Experimentation and Testing with Analog and Digital Devices" for graduate students.

His research areas are mechanical vibrations, control, mechatronics, system dynamics, image processing, and microstructure analysis of materials. He is the author of more than 30 research papers and 5 books. Three of the five books are devoted to MATLAB/Simulink applications for mechanical engineering students and numerical analysis. From 2009 to 2020, he was an external academic expert for the European Commission, assessing academic projects.

About the Technical Reviewer

Karpur Shukla is a research fellow at the Centre for Mathematical Modelling at FLAME University in Pune, India. His current research interests focus on topological quantum computation, nonequilibrium and finite-temperature aspects of topological quantum field theories, and applications of quantum materials effects for reversible computing. He received an MSc in physics from Carnegie Mellon University, with a background in theoretical analysis of materials for spintronics applications as well as Monte Carlo simulations for the renormalization group of finite-temperature spin lattice systems.

Acknowledgments

I express my special gratitude to the technical reviewers, proofreaders, and editors of Apress for their very thorough work while reviewing the content and code in this book. Without their critical insights and corrections at many points of the book, I would not have been able to complete it with this quality. In addition, I would like to express my special gratitude to Mark Powers for his on-time and well-planned correspondence throughout this book project.

My cordial gratitude goes to my mother for her limitless support and love. Up until the very last point of this book, she was always checking in about my progress.

I would like to thank my wife, Nigora, because without her great support, I would not have been able to take up the challenging task of writing this book. I have spent many weekends in my office writing and editing the book content and the MATLAB/Simulink scripts and models. In addition, I would like to thank our children, Anbara, Durdona, and Dovud, for being such delightful people and being the inspiration for my efforts while writing this book.

Introduction

This book covers the most essential and hands-on tools and functions of the MATLAB and Simulink packages and the Symbolic Math Toolbox (MuPAD notes) to solve, model, and simulate ordinary differential equations (ODEs) and partial differential equations (PDEs). It explains how to solve ODEs and PDEs symbolically and numerically via interactive examples and case studies. The main principle of the book is "learn by doing," moving from the simple to the complex. This book contains dozens of solved problems and simulation models embedded in MATLAB scripts and Simulink models, which will help you to master programming and modeling essentials, as well as learn how to program and model more difficult and complex problems that involve ODEs and PDEs.

Practical MATLAB Modeling with Simulink explains various practical issues of programming and modeling in parallel by comparing the programming tools of MATLAB to the modeling tools of Simulink. By studying this book, you'll be proficient at using the MATLAB/Simulink packages and at using the source codes and models from the book's examples as templates for your own projects to solve modeling and simulation, or engineering problems with ODEs and PDEs.

What You Will Learn

By the end of the book, you'll have learned how to do the following:

- Model complex problems using MATLAB and Simulink
- Use MATLAB and Simulink to solve ODEs and PDEs
- Use numerical methods to solve first-, second-, and higher-order and coupled ODEs
- Solve stiff and implicit ODEs
- Employ numerical methods to solve first- and second-order linear PDEs

- Solve ODEs symbolically with the Symbolic Math Toolbox of MATLAB
- Understand the applications and modeling aspects of differential equations in solving various simulation problems

This book is aimed at engineers, programmers, data scientists, and students majoring in engineering, applied/industrial math, data science, and scientific computing. This book continues where Apress' *Beginning MATLAB and Simulink* leaves off.

The book is composed of three parts. Part 1 consists of the following chapters:

- Chapter 1 is dedicated to general formulations and solving ODEs symbolically using The Symbolic Math Toolbox functions and MuPAD note commands.
- Chapter 2 covers the most essential programming aspects to solve first-order ODEs numerically by writing scripts based on the Euler, Runge-Kutta, Milne, Adams-Bashforth, Ralston, and Adams-Moulton methods. You'll write the scripts using MATLAB's built-in ODE solvers, such as ode23, ode23t, ode45, ode15s, ode113, odeset, etc., and Simulink modeling.
- Chapter 3 is dedicated to solving second-order ODEs symbolically and numerically using the Euler, Runge-Kutta, Milne, Adams-Bashforth, Ralston, and Adams-Moulton methods. You'll do this using MATLAB's built-in ODE solvers, such as ode23, ode23t, ode45, ode15s, ode113, odeset, etc., and Simulink modeling.
- Chapter 4 addresses the issues of how to solve stiff ODEs by adjusting a step size, specifying a solver setting, selecting an appropriate solver type, and so forth.
- Chapter 5 is dedicated to solving higher-order and coupled ODEs.
- Chapter 6 is devoted to solving implicitly defined IVPs and differential algebraic equations using MATLAB's ode15i and Simulink modeling.

- Chapter 7 is dedicated to a comparative analysis of solving ODEs with MATLAB's ODE solvers, Simulink modeling, script writing, and computing analytical solutions with the Symbolic MATH Toolbox (MuPAD notes).

Part 2 consists of the following chapter:

- Chapter 8 covers most essential tools of solving boundary value problems of ODEs numerically by using MATLAB solvers such as bvp4c, bvp5c, bvpinit, deval, etc.

Part 3 consists of the following chapters:

- Chapter 9 is devoted to applications of ODEs, specifically, modeling and simulations of spring-mass-damper systems.
- Chapter 10 is devoted to applications of ODEs, specifically, modeling and simulations of mechanical and electromechanical systems.
- Chapter 11 is devoted to applications of ODEs, specifically, modeling and simulations of trajectory problems.
- Chapter 12 is devoted to applications of ODEs, specifically, modeling and simulations of several simulation problems, such as the Lotka-Voltera and Lorenz systems.

Part 4 consists of the following chapter:

- Chapter 13 is devoted to solving partial differential equations using PDE toolbox functions, such as `pdepe()` and the Laplacian operator `del2()`, and writing scripts based on Gauss-Seidel and finite difference methods.

How to Access the Source Code

All of the source code (such as M/MN files, Simulink models, and SLX/MDL files) discussed in the book is available to readers via the Download Source Code button at `www.apress.com/9781484257982`.

A Note to Users

The given scripts in the context of the book may not be the best solutions to the given problems, which was done intentionally in some cases to emphasize methods used to improve them; in other cases, the given scripts are the most appropriate solutions to the best knowledge of the author. Should I spot any better alternative solutions to exercises given in the context of the book, I intend to publish them via the MathWorks MATLAB Central User Community's file exchange via my file exchange link there—under my name.

No matter how hard we have worked to proofread the content of the book, it is inevitable that there might be some typographical errors that have slipped through and will appear in print. My apologies.

Sulaymon L. Eshkabilov
January 2020

PART I

Ordinary Differential Equations

CHAPTER 1

Analytical Solutions for ODEs

Many modeling problems in engineering applications can be formulated using ordinary differential equations. There are a few different definitions of *differential equation*; one of the simplest ones is "A differential equation is any equation which contains derivatives, either ordinary derivatives or partial derivatives" given in [1]. From this definition, we can derive that there are two types of differential equations: ordinary differential equations (ODEs) and partial differential equations (PDEs). ODEs contain one type of derivative or one independent variable, while PDEs contain two or more derivatives or independent variables. For example, a general form for first-order ODEs can be expressed by:

$$\frac{dy}{dx} = f(y,x) \tag{1-1}$$

where $y(x)$ is a dependent variable whose values depend on values of the independent variable of x. Another good example of an ODE is Newton's second law of motion formulated by:

$$ma = \frac{dp}{dt} = \frac{mdv}{dt} = F(t,v) \tag{1-2}$$

where $F(t, v)$ is the force, which is a function of time (t) and velocity (v); $\frac{dv}{dt}$ is a velocity change rate (acceleration) of a moving object; m is the mass of a moving object; a is an acceleration of a moving object; p is momentum; and $\frac{dp}{dt}$ is its derivative.

S. L. Eshkabilov, *Practical MATLAB Modeling with Simulink*, https://doi.org/10.1007/978-1-4842-5799-9_1

The previous formulation of Newton's second law can be also rewritten in the following way:

$$\frac{md}{dt}\left(\frac{dx}{dt}\right)=\frac{md^2x}{dt^2}=F\left(t,x,\frac{dx}{dt}\right) \tag{1-3}$$

where the derivative $\left(\frac{dx}{dt}\right)$ of the displacement (x) of a moving object is the velocity (v). In other words, the velocity is a change rate of the displacement $x(t)$ of a moving object in time. This can be visualized with the flowchart displayed in Figure 1-1.

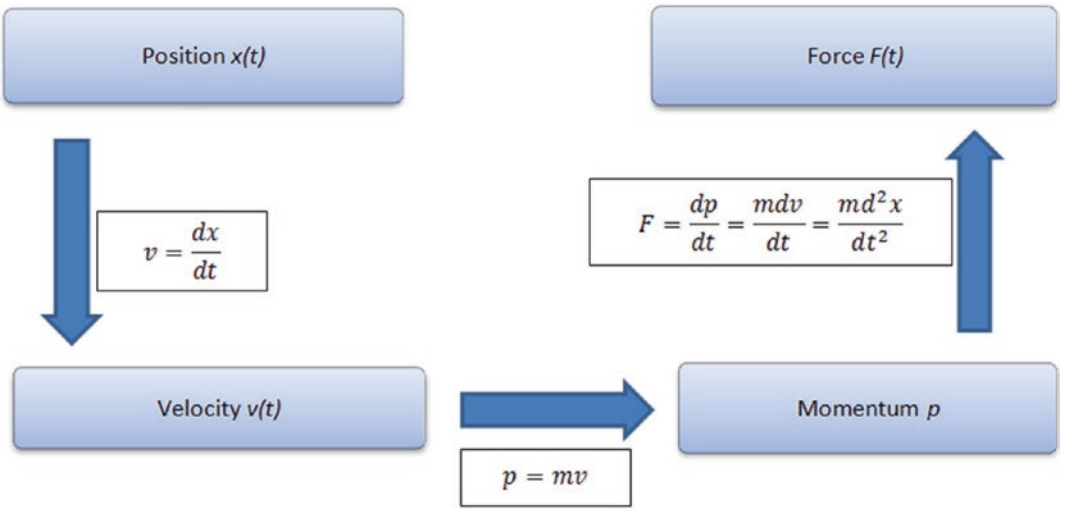

Figure 1-1. *Flowchart expressing motion and exerted force of a moving object*

Classifying ODEs

There are two classifications of ODE-related problems.

- **Initial value problems (IVPs):** Here's an example: $\ddot{x}=xt-3\dot{x}$ with these initial conditions: $x(0)=3,\ \dot{x}(0)=1$.
- **Boundary value problems (BVPs):** Here's an example: $\ddot{x}=xt-3\dot{x}$ with these boundary conditions: $x(0)=3,\ x(2)=1.50$.

IVPs are defined with ODEs together with a specified value, called the *initial condition*, of the unknown function at a given point in the solution domain. In the IVP of ODEs, there might be a unique solution, no solution, or many solutions. By definition,

the IVP of ODEs can be explicitly defined or implicitly defined. Most IVPs are explicitly defined.

We will start with explicitly defined IVPs and then move on to implicitly defined ones. Besides being categorized by solution type (how the solution values change over the solution search space), IVPs are divided into stiff and nonstiff problems. Moreover, ODEs are either linear or nonlinear and are either homogeneous or nonhomogeneous.

Here are some specific examples of different ODE types, categories, and groups:

Stiff ODEs: $\begin{cases} \dot{x} = -0.04x + 10^4 yz \\ \dot{y} = 0.04x - 10^4 yz - 3*10^7 y^2 \\ \dot{z} = 3*10^7 y^2 \end{cases} \quad t \in [0, 40]$

Nonstiff ODEs: $\dot{y} + 2y = 2t, \ \ddot{w} + \dot{w} = 5$

Linear ODEs: $\frac{dv}{dt} = 9.81 - 0.198v, \ \ddot{x} + 3\dot{x} + 5x = 0$

Nonlinear ODEs: $\frac{dv}{dt} = 9.81 - 0.198v^2, \ \ddot{x} + 3x\dot{x} + 5x^2 = 0$

Homogeneous ODEs: $\dot{y} + 2y = 0, \ \ddot{x} + 3\dot{x} + 5x = 0$

Nonhomogeneous ODEs: $\dot{y} + 2y = \sin(x), \ \ddot{x} + 3\dot{x} + 5x = e^{2t}$

This chapter contains several examples for ODEs and their application areas.

Example 1

This example shows an exponential growth problem. This equation could describe unconstrained growth of biological organisms (bacteria), values of real estate or investments, membership of a popular networking site, growth in retail business, positive feedback of electrical systems, or generated chemical reactions. It is formulated by the following first-order ODE:

$$\frac{dy}{dt} = \mu y \ \text{ has a solution: } y(t) = y_0 e^{\mu t}$$

Example 2

This example shows exponential decay. This equation could describe many phenomena in nature and engineering, such as radioactive decay, washout of chemicals in a reactor, discharge of a capacitor, and decomposition of material in a river. Exponential decay is expressed with the following first-order ODE:

$$\frac{dy}{dt} = -\mu y \text{ has a solution: } y(t) = y_0 e^{-\mu t}$$

Examples 1 and 2 are two simple examples of first-order ODEs.

Example 3

The motion of a falling object is expressed in the following way using Newton's second law:

$$\frac{md^2y}{dt^2} = mg - \frac{\gamma dy}{dt}$$

This is a second-order ODE that has a general solution in the following form:

$$y(t) = C_1 e^{-\left(\frac{\gamma t}{m}\right)} - \frac{m(mg - g\gamma t)}{\gamma^2} + C_2$$

where m is the mass of a falling object; g is gravitational acceleration; and γ is an air-drag coefficient of a falling object. There are three parameters. Specifically, m, g, and γ are constant, and two parameters change over time: $\frac{d^2y}{dt^2}$ (acceleration) and $\frac{dy}{dt}$ (velocity). In the general solution of a falling object using the equation of motion, C_1 and C_2 are arbitrary numbers that are dependent on the initial conditions or, in other words, can be computed considering the initial conditions of a falling object.

There are a few methods to evaluate the analytical solutions of ODEs, including the separation of variables, introduction of new variables, and others. We show via specific examples of these types of ODEs and explain how to evaluate their analytical solutions and compute numerical solutions by employing different techniques in the MATLAB/Simulink environment through scripts and building models. We evaluate analytical solutions of ODEs via specific examples and demonstrate how to use the

built-in functions of the Symbolic Math Toolbox[1] and MuPAD[2] notebooks. We put more emphasis on the Symbolic Math Toolbox's command syntaxes rather than MuPAD notebooks. The reason for this is that in future releases of MATLAB, a technical support for MuPAD notebooks will removed.

There are a number of numerical methods, including Euler (forward, backward, modified), Heun, the midpoint rule, Runge-Kutta, Runge-Kutta-Gill, Adams-Bashforth, Milne, Adams-Moulton, Taylor series, and trapezoidal rule methods. Some of these methods are explicit, and others are implicit. To demonstrate how to employ these methods, we will first describe their formulations concisely, and then we will work on their implementation algorithms for writing scripts (programs) explicitly. We do not attempt to derive any of the formulations used in these numerical methods, and there are many literature sources [2, 3, 4, 5] explaining the theoretical aspects of these methods.

In solving an IVP with numerical methods, we first start from an initial point (initial conditions) and then take a step (equal step size or varying step size) forward in time to compute numerical solutions. Some of the previously named numerical methods (e.g., Euler's methods) are single-step methods, and others (Runge-Kutta, Adams-Bashforth, Milne, Adams-Moulton, Taylor series) are multistep methods. Single-step methods refer to only one previous point and its derivative to determine the current value. Other methods, such as Runge-Kutta methods, take some intermediate steps to obtain a higher-order step and then drop off values before taking the next step. Unlike single-step methods, multistep methods keep and use values from the previous steps instead of discarding them. In this way, multistep methods link a few previously obtained values (solutions) and derivative values. All of these methods, such as the single-step and multistep methods, are assessed based on their accuracy and efficiency in terms of computation time and resources (e.g., machine time) spent to compute numerical solutions for specific types of IVPs of ODEs.

[1]Symbolic Math Toolbox is a registered trademark of The MathWorks Inc.

[2]MuPAD is a registered trademark of The MathWorks Inc.

Analytical Solutions of ODEs

The Symbolic MATH Toolbox (or MuPAD notebooks) has several functions capable of evaluating analytical solutions of many analytically solvable ODEs. There are two commands (built-in functions)—`dsolve()` and `ilaplace/laplace`—with which analytical solutions of some ODEs can be evaluated.

Note that in this section we demonstrate—via a few examples of first- and second-order ODEs and systems of coupled differential equations—how to compute analytical solutions of ODEs.

dsolve()

`dsolve()` is an ODE solver tool to compute an analytical (or general) solution of any given ODE in MATLAB. `dsolve()` can be used with the following general syntaxes:

```
Solution = dsolve(equation)
Solution = dsolve(equation, conditions)
Solution = dsolve(equation, conditions, Name, Value)
[y1,...,yN] = dsolve(equations)
[y1,...,yN] = dsolve(equations, conditions)
[y1,...,yN] = dsolve(equations, conditions, Name, Value)
```

Example 4

Given an ODE, $\dot{y} + 2ty^2 = 0$. Note that initial or boundary conditions are not specified. Here is the command by which we can compute a general analytical solution of the given example.

```
>> y_solution=dsolve('Dy=-2*y^2*t')

Y_solution=-1/(C3-t^2)
```

Note that `C3` is defined from the initial or boundary conditions of the given ODE.

There is an alternative command. The given problem with newer versions of MATLAB (starting with MATLAB 2012) can be solved by using the following command syntax:

```
>>syms y(t); y_sol=dsolve(diff(y) == -2*y^2*t)
 y_sol =
         0
 -1/(-t^2 + C3)
```

Example 5

Given an ODE, $\dot{y} + 2ty^2 = 0$, with the initial condition $y(0) = 0.5$.

```
>> Solution=dsolve('Dy=-2*y^2*t', 'y(0)=0.5')
Solution =
 1/(t^2 + 2)
```

Here is the alternative command syntax for newer versions of MATLAB:

```
>> syms y(t); Solution=dsolve(diff(y) == -2*y^2*t, y(0)==0.5)
Solution =
 1/(t^2 + 2)
```

Here is another alternative syntax for newer versions of MATLAB:

```
>> syms y(t) t; Dy=diff(y, t); Equation = Dy ==-2*y^2*t; IC=y(0)==0.5;
Solution=dsolve(Equation, IC); pretty(Solution)
 1
------
 2
t  + 2
```

The resulting analytical solution is in a symbolic formulation and thus can be plotted (Figure 1-2) with `ezplot` or the recommended `fplot`. (`ezplot` will not be supported in future releases of MATLAB.)

```
>> fplot(Solution, [-5, 5], 'r--o'), grid on
>>title('Solution of: $$ \frac{dy}{dt}+2y^2t=0, y_0 = 0.5 $$',
'interpreter', 'latex')
>> xlabel('\it t'), ylabel '\it Solution, y(t)'
```

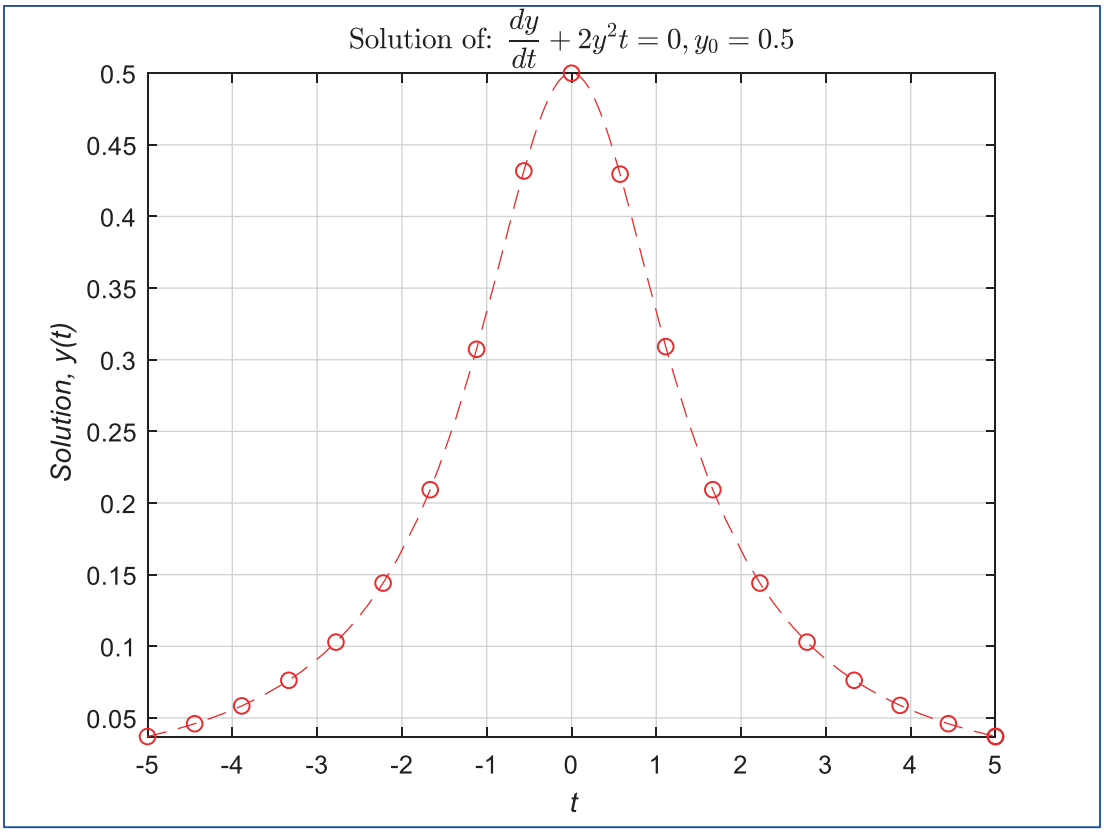

Figure 1-2. *Analytical solution of* $\dot{y}+2ty^2=0, y(0)=0.5$

Numerical values of the resulting analytical solution (equation) can be computed by vectorizing (parameterizing) the resulting symbolic formulation (solution), as shown here:

```
>> ysol=vectorize(solution)
ysol =
1./(t.^2 + 2)
>> t=(-5:.1:5); ysol_values=eval(ysol);
```

An alternative way of computing numerical solution values is to use `fplot`, as shown here:

```
[t, yt]=fplot(Solution, [-5, 5]);
```

Example 6

Given $\dot{y}+kty^2=0, y(0)=0.5$. Note that this exercise has one unspecified parameter, *k*.

```
>> syms k
>> solution=dsolve('Dy=-k*y^2*t', 'y(0)=0.5')
solution =
1/((k*t^2)/2 + 2)
```

Here is the alternative command syntax:

```
>> syms y(t) k; solution=dsolve(diff(y) == -k*y^2*t, y(0)==0.5)
solution =
1/((k*t^2)/2 + 2)
```

Example 7

Given $\dot{y}-|y|e^t=2, y(0)=2$. Let's solve this exercise in a MuPAD note. Figure 1-3 shows the commands used to compute an analytical solution for this exercise.

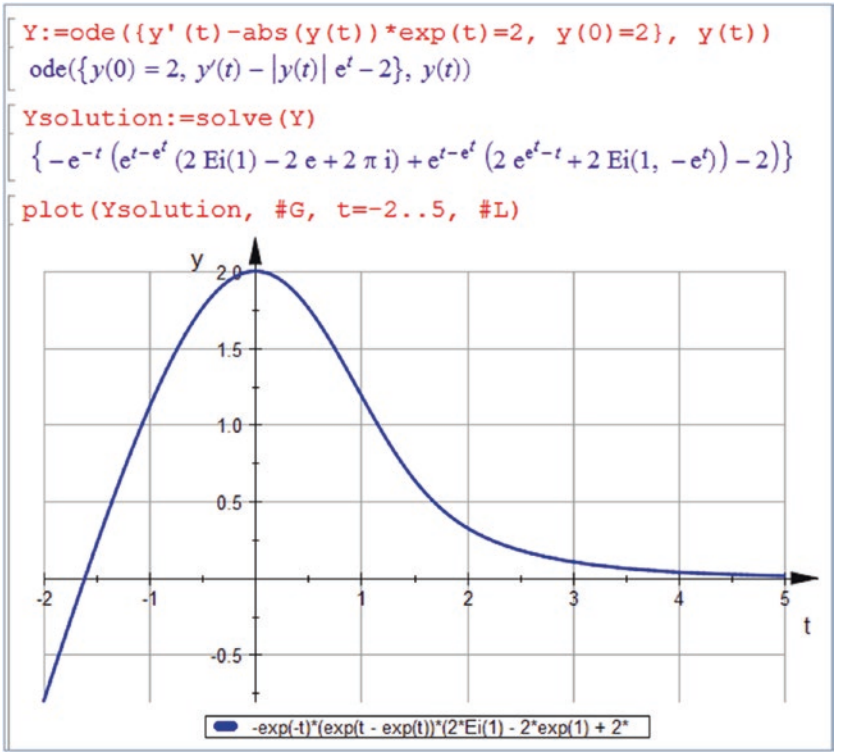

Figure 1-3. *MuPAD commands to solve* $\dot{y}-|y|e^t=2, y(0)=2$ *analytically*

Note The options in `dsolve()` need to be set appropriately depending on the problem type. For MATLAB 2008–2010 or earlier versions, you should set `IgnoreAnalyticConstraints` to `none` to obtain all or any possible solutions.

Here's an example:

```
solution=dsolve('Dy=-k*y^2*t', 'y(0)=0.5', ... 'IgnoreAnalyticConstraints',
'none')
```

Note For MATLAB 2012 or newer versions, you should set `IgnoreAnalyticConstraints` to `false` to get all the possible correct answers for all the argument values. Otherwise, `dsolve()` may output an incorrect answer because of its pre-algebraic simplifications.
