Mathematical Analysis of Stiff and Non-Stiff Initial Value Problems of Ordinary Differential Equation Using Matlab

*D. Omale, P.B. Ojih, M.O. Ogwo

Abstract - Many important and complex systems from different fields of sciences are modeled using differential equations. Due to the complexity of these systems, analytical methods are often difficult or impossible to implement for such problems and so numerical methods are the way out. The advent of computer applications like MATLAB starting from the mid 20th century has made a drastic improvement in numerical solutions for differential equations. In this work, we present the solvers in MATLAB for obtaining numerical solution for initial value problems of ODEs – ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb. Six problems are solved, three of which are stiff and three non-stiff using the relevant MATLAB solvers and the solutions are presented.

Index Terms – Advent of computer application, Analytic approach, Differential equation, Dynamic, Matlab, Matrix laboratory, Non-Stiff, Stiff,

……♦……

1. INTRODUCTION

The dynamic behavior of systems is an interesting and important subject of study for scientists, Marcel B. Finan. (2012). Mechanical systems involve time related change in position and speed, electric systems involve change in current and voltage with time as well as several other systems from fields like engineering, economics, social sciences, biology, business and so on. These systems involving change are represented mathematically using differential equations. Differential equations are equations involving the derivatives of a dependent variable with respect to one or two independent variables.

Differential equations can be classified as ordinary or partial. A differential equation is ordinary if the unknown function is dependent only on a single variable. If the unknown function is dependent on multiple independent variables and the equation involves its partial derivatives, then the equation is a partial differential equation. Marcel, (2012).

Differential equations are also classified according to their order. The order of a differential equation is the highest derivative in the equation. A differential equation that has the second derivative as the highest derivative is said to be of order 2. The highest power of the highest derivative in a differential equation is the degree of the equation.

In physics, Newton's Second Law, Navier Stokes Equations, Cauchy-Riemman Equations, Schrodinger Equations are all well known differential equations. The Lotka-Voltera Equations, Verhulst Equations and Replicator dynamics in biology as well as the exogenous growth model and Malthusian growth model in Economics are all represented by differential equations.

The solution of a differential equation is a value of the dependent variable in the equation that satisfies the equation at all points of the solution domain. The solution of a differential equation at a point is the value of the dependent variable at that point. Solutions to differential equations can be categorized in three broad sections. The analytic approach of solution, the qualitative approach and the numerical approach.

The analytic approach seeks to provide an explicit solution to the differential equation. Many important equations are impossible to solve using this method. The qualitative approach does not provide explicit solutions; rather it uses geometry to provide an overview of the behavior of the model. The qualitative approach yields solutions in form of direction fields, solution curves and phase plots. This method may be used to validate an analytic or numeric result. The numerical method provides approximate values of the solutions to the differential equation. The numerical method starts with an initial value of the variable and then uses the equations to figure out the changes in this variable over a brief time and continues to compute approximations of the solutions until the end of the desired solution interval.

Differential equations along with a specified value of the unknown function at a given point in the domain of the solution are an Initial Value Problem. This specified value is the initial condition. In many important cases of differential equations, analytic solutions are difficult or impossible to obtain and time consuming. Eric, (2013). Although numerical solutions are approximations, the error of approximation is often acceptable and numerical solutions give birth to algorithms that are used to design computer simulated solutions.

A major development in the study of numerical methods is the introduction of modern computers for the calculation of functions in the mid 20th century. Wikipedia, (2014). Until this time, numerical methods often depended on hand interpolation in large printed tables. These tables contained data such as interpolation points and function coefficients up to 16 decimal places or more and were used to obtain very good numerical estimates of functions. Although these same algorithms continue to be part of the base of software algorithms for obtaining numerical solutions, the way computer represents and processes numbers gives rise to inexact results from the programs. Inexactness arises for different reasons like the number of decimal places in which results are retrieved and the number of steps required to the final solution, nevertheless, the use of computers and computer applications in numerical methods has become an established part of general numerical analysis with the development of many numerical computing applications such as MATLAB, S-PLUS, LabView, FreeMat, Scilab, GNU Octave and their associated speed and performance in obtaining solutions.

2. OVERVIEW OF MATLAB

MATLAB which stands for Matrix Laboratory is a high-level language and interactive computer environment developed by MathWorks for numerical computation, visualization, and programming. Using MATLAB, you can analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions enable you to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java.

You can use MATLAB for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.

3. STIFFNESS OF ORDINARY DIFFERENTIAL EQUATIONS

Stiff ordinary differential equations arise frequently in the study of chemical kinetics, electrical circuits, vibrations, control systems and so on. It is a difficult and important concept in the study of differential equations. Stiffness has no generally accepted definition but several attempts had been made at defining the concept. Some of the definitions are:

1. A stiff problem is one for which no solution component is unstable (no eigenvalue of the Jacobian matrix has a real part which is at all large and positive) and at least some component is very stable (at least one eigenvalue has a real part which is large and negative.
2. A problem is stiff if the solution being sought varies slowly but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.
3. Stiffness occurs when some components decay more rapidly than others.
4. The linear system of differential equations

$$\frac{du}{dt} = \boldsymbol{A}u(t)\ t \in [0,T]$$

$$u(0) = u_0$$

is stiff when the eigenvalues λ_k $(k = 1,2,\dots,m)$ of the constant coefficient matrix $\boldsymbol{A}$ of the system have the following properties:

5. $\mathrm{Re}\lambda_k < 0$ for each $k = 1,2,\dots,m$ (all its eigenvalues have negative real parts.

   The number, S defined as, $S = \frac{max_k|Re\lambda_k|}{min_k|Re\lambda_k|}$ is large, i.e. $S \gg 1$. S is called the stiffness number.

6. A problem is stiff if explicit methods fail to provide solutions or works extremely slowly.

According to Shampine and Thompson as reported in Aliyu et al (2014), the best way to detect stiffness is to try one of the solvers intended for non-stiff systems. If it is unsatisfactory, then the problem may be stiff and effective stiff solvers should be employed. We shall employ this method of detecting stiffness in this work.

The objective of this work is to present how MATLAB solvers are applied in providing numerical solutions for initial value problems of ordinary differential equations.

4. MATLAB ODE SOLVERS

There are several inbuilt solvers for differential equations in MATLAB. Eight (8) of these solvers are applicable for initial value ODE problems. They are: ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb, ode15i. These ODE solvers are designed to handle three types of first order ODEs:

1. Explicit ODEs of the form $y' = f(t, y)$
2. Linearly implicit ODEs of the form $\boldsymbol{M}(t, y).y' = f(t, y)$, where $\boldsymbol{M}(t, y)$ is a matrix.
3. Fully implicit ODEs of the form $f(t, y, y') = 0$

In this work, we shall consider explicit ODEs only.

Initial value problems are categorized as stiff and non-stiff, Shampine, et al (2003) and this classification is important in selecting the solvers to use in MATLAB. Stiffness is a subtle, complicated and important concept in numerical solutions of ordinary differential equations. A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly and so the numerical method must take small steps to obtain satisfactory results. Stiffness also forms a basis for the classification of ODE solvers in MATLAB. Some solvers are designed for stiff problems and some others for non-stiff problems. Stiffness is an efficiency issue. If non-stiff methods are used to solve stiff IVPs, the solver will either take an extremely long time to supply a solution or supply an inaccurate solution or fail to supply any solution at all. All stiff problems are difficult for solvers that are not designed for them and this is why it is necessary to select an appropriate solver when using MATLAB ODE solvers. MATLAB documentation recommends that ode45 is the best solver to give the first try for most problems. If ode45 is slow in solving the problem or failed to solve the problem, then stiffness is suspected and you can then go ahead to try ode15s.

Ode45

ode45 is a sophisticated built-in MATLAB function that gives very accurate solutions. Ode45 is based on a simultaneous implementation of an explicit fourth and fifth order Runge-Kutta formula called the Dormand-Prince pair. It is a one-step solver. This is the first solver to be tried for most problems. It is designed for non-stiff problems. Ode45 can use long step size and so the default is to compute solution values at four points equally spaced within the span of each natural step.

The Dormand-Prince pair is an explicit method and a member of the Runge-Kutta family of solvers. The method employs function evaluations to calculate fourth and fifth order accurate solutions. The difference between these solutions is then taken to be the error of the fourth order solution.

Ode23

This solver is designed for solving non-stiff problems. It's method is based on the 2nd and 3rd Order Runge-Kutta pair called the Bogacki-Shampine method. Ode23 is less expensive than ode45 in that it requires less computation steps than ode45. But it is of a lower order, although it may be more efficient at crude tolerances and in the presence of mild stiffness. Ode23 is a one-step solver.

The Bogacki-Shampine method is a Runge-Kutta method of order 3 with four stages proposed by Przemyslaw Bogacki and Lawrence F. Shampine in 1989. It uses three function evaluations per step. It has embedded second order method which is used to implement adaptive step size for the method.

Ode113

Ode113 is a multi-step variable order method which uses Adams–Bashforth–Moulton predictor-correctors of order 1 to 13. It may be more efficient than ode45 at stringent tolerances and when the ODE problem is particularly expensive to evaluate. It is designed for non-stiff problems.

Ode 15s

Ode15s a variable order solver whose algorithm is based on the numerical differentiation formulas (NDFs) and optionally along with the backward differentiation formulas (BDFs) which is also called Gear's method. This is a multi-step solver that is designed for stiff problems. This is the next recommended solver if ode45 fails or is too slow.

The Backward Differentiation Formula is a family of implicit methods for the numerical integration of ordinary differential equations. They are linear multi-step methods and for a given function and time, they approximate the derivative of that function using information from already computed times. BDF methods are implicit and, as such, require the solution of nonlinear equations at each step.

Ode23s

This is based on a modified Rosenbrock formula of order 3 and 2 with error control designed for stiff systems. It advances from y_n to y_{n+1} with the second order method and controls the local error by taking the difference between the third and second order numerical solutions. This is a one step method and

can solve some kinds of stiff problems for which ode15s is not effective.

Ode23t

The algorithm implemented in this solver is the trapezoidal rule along with a free interpolant. This solver is designed for only moderately stiff problems and when the solution required should be without damping.

Ode23tb

Is an implementation of TR-Backward Differentiation Formula 2. This is an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order 2. This method may be used when crude error tolerance is required to solve stiff systems.

Ode15i

This solver is designed based on a variable order method for fully implicit differential equations. This is the only solver that is designed for fully implicit differential equations.

A Summary of MATLAB ODE Solvers

Solver	Kind of Problem	Base Algorithm
Ode45	Non-stiff differential equations	Runge-Kutta
Ode23	Non-stiff differential equations	Runge-Kutta
Ode113	Non-stiff differential equations	Adams-Bashfort-Moulton
Ode15s	Stiff differential equations	Numerical Differentiation Formulas (Backward Differentiation Formulas)
Ode23s	Stiff differential equations	Rosenbrock
Ode23t	Moderately stiff differential equations	Trapezoidal Rule
Ode23tb	Stiff differential equations	TR-BDF2
Ode15i	Fully implicit differential equations	BDFs

5. MATLAB ODE SOLVER SYNTAX

Apart from the ode15i solver, all other ODE solvers in MATLAB have the same syntax. This makes it easy to apply different methods to the same problem. The basic syntax for all the solvers is

$$[t, y] = solver(f, tspan, y0, options)$$

Where $solver$ is the ode solver method, e.g. ode45, ode23 and others.

The output arguments are:

$\boldsymbol{t}$	Column vector of time points
$\boldsymbol{y}$	Solution array

The input arguments are:

6. Optional Parameters.

The default integration properties for the ODE solvers are selected to handle common problems. It is possible to use optional parameters to improve ODE solvers performance. This will override the default integration properties of the solver.

The optional properties that can be set are in several categories. Some of them are Error Control Properties, Solver Output Properties, Step-Size Properties, Event Location Property, Jacobian Matrix Properties.

On the error control properties, each MATLAB ODE solver is designed in a manner that the estimated local error in each component of the solution satisfies a given error test. For a single equation the estimated local error in passing from $y(t_n)$ to $y(t_{n+1})$, call it $e(t_n)$, is to satisfy $|e(t_n)| \leq max\{AbsTol, RelTol \cdot |y(t_n)|\}$. The error tolerances $AbsTol$ and $RelTol$ can be specified by running the MATLAB program $odeset$; when left unspecified, the default tolerances are $AbsTol = 10^{-6}, RelTol = 10^{-3}$.

The step size properties specify the size of the first step the solver should use and the upper bound on the step size that the solver can use. The property that specifies the initial step size is called $InitialStep$

and the property that specifies the upper bound on solver step size is called $MaxStep$. To set optional parameter properties, we use the $odeset$ function and the syntax is as shown below:

$$options = odeset('name1', value1, 'name2', value\ 2, \dots)$$

'name1' refers to the property name that has the specified value 'value1'. Any property that is unspecified retains its default value.

For example, to set the initial step size, we have

$$options = odeset('InitialStep', 0.0001)$$

7. PROCEDURE FOR SOLVING FIRST ORDER ODEs USING MATLAB SOLVERS

1. **Code the Problem**
   In this step, you develop an m-file to contain the given function. For the general problem of the form $y' = f(t, y)$, the code will look like this:
   $function\ dydt = p1(t, y)$
   $dydt = f(t, y);$
   end

   For example, consider the problem, $y' = 2t + 1$, the code for the problem is:
   $function\ dydt = p1(t, y)$
   $dydt = 2 * t + 1;$
   end
   The names $dydt\ and\ p1$ in the code above do not play any role in the definition of the function. Any name can be used as long as they conform with MATLAB naming rules for functions and m-files.
2. **Apply a Solver to the Problem**
   Make a decision of the solver you intend to use. Then call the solver using the syntax in Section 3.4.1 and pass the function, the time interval you desire for your solution and the initial condition vector to it. For example, to apply ode45 solver to the problem described by $p1$ in step 1 above on the time interval between 0 and 10, with initial condition $y(0) = 0$, the following code is typed into the command window:
   $tspan = [0, 10];$
   $y0 = 0$
   $[t, y] = ode45(@p1, tspan, y0)$
3. **Retrieve the Solver Output**

   You can simply use the $plot$ command to view the solver output. i.e. $plot(t, y)$

   You can also display the results by using the $disp$ command i.e. $disp([t, y])$

   Phase portrait of the solution can also be retrieved.

PROCEDURE FOR SOLVING HIGHER ORDER ODEs USING MATLAB SOLVERS

1. **Rewrite the second order ODE as a system of first order ODEs.**
   For the nth-order ODE: $y^{(n)} = f\left(t, y, y', \dots, y^{(n-1)}\right)$
   Set $y_1 = y,\ y_2 = y',\ \dots, y_n = y^{(n-1)}$
   The result of this substitution will be an equivalent system of n first order ODEs

$$y_1' = y_2$$
$$y_2' = y_3$$
$$\vdots$$
$$y_n' = f(t, y_1, y_2, \dots, y_n)$$

   For example, consider the second order ODE $y'' = f(t,\ y,\ y')$
   Set, $y_1 = y$ and $y_2 = y'$
   This implies that, $y_1' = y' = y_2$ and $y_2' = y'' = f(t, y_1, y_2)$
   So, we obtain the system of first order ODEs,

$$y_1' = y_2$$
$$y_2' = f(t, y_1, y_2)$$

2. **Code the system of First Order ODEs.**
   Develop an m-file containing the system of first order ODEs obtained from splitting the higher order ODE in question. For the general second order ODE in step 1 above, the code will look like this:
   $function\ dydt = p2(t, y)$
   $dydt(1) = y(2);$
   $dydt(2) = f(t, y_1, y_2);$
   $dydt = [dydt(1);\ dydt(2)];$
   end
3. **Apply a Solver to the Problem.**
   You use the solver syntax to call the solver you want to use as in the case of the first order ODE.
4. **Retrieve the Solver Output**
   You can simply use the $plot$ command to view the solver output. i.e. $plot(t, y)$

   You can also display the results by using the $disp$ command i.e. $disp([t, y])$

   Phase portrait of the solution can also be retrieved.

APPLICATION OF MATLAB ODE SOLVERS TO NON-STIFF INITIAL VALUE PROBLEMS OF ODEs.

In this section, we apply the relevant solvers to three non-stiff problems (Problems 1, 2 and 3) and three stiff problems (Problems 4, 5 and 6). Solution graphs and Phase Portraits of the systems are also presented.

Problem 1:

The state variable equations of a continuous system describing the dynamic behaviour of the system is given by

$$y_1' = 5y_2$$

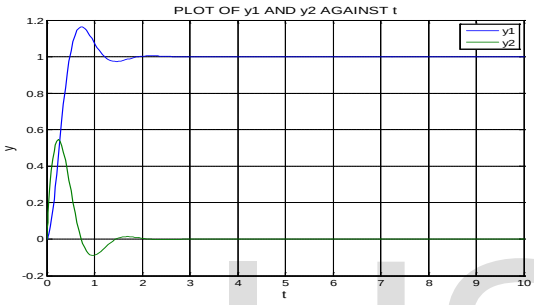
$$y_2' = -5y_1 - 5y_2 + 5$$

$$y_1(0) = y_2(0) = 0$$

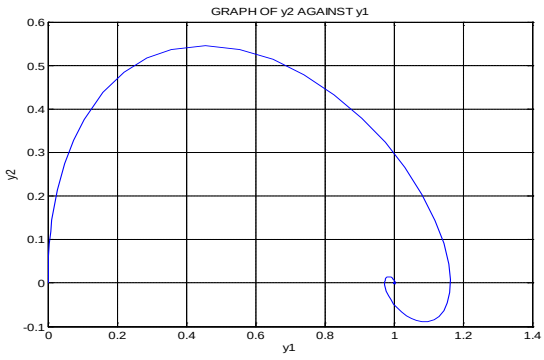
Develop a MATLAB program that plots y_1 *and* y_2 on the same graph for $t = 0$ *to* 10. obtain the phase portrait of the system.

Source: Exercises from Ogbonnaya (2008)

Solution: Using ode45 to solve the problem:



Phase Portrait of the System



Problem 2.

Lotka-Volterra Equations

The populations of two species, a prey denoted by y_1 and predator denoted by y_2, can be modeled by a system of ODEs:

$$y_1' = by_1 - cy_1y_2$$

$$y_2' = -dy_2 + cy_1y_2$$

due to Lotka and Volterra.

The parameters b and d govern the birth rate of prey and death rate of predators, respectively, and the parameter c governs the interaction of the two populations. With the parameter values $b = 1, d = 10$, and $c = 1$, and initial conditions $y_1(0) = 0.5$ and $y_2(0) = 1$

(the populations are normalized, and we treat them as continuous variables), use MATLAB function ode45 to solve this system numerically, integrating to $t = 10$. Plot each of the two populations as a function of time, and on a separate graph plot the trajectory of the point $(y_1(t), y_2(t))$ in the plane.

Source: Heath (1997)

Solution:

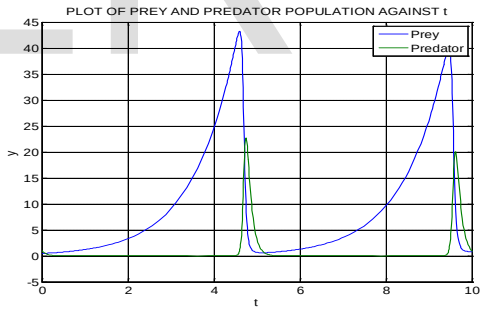
Our system of ODEs now become

$$y_1' = y_1 - y_1y_2$$

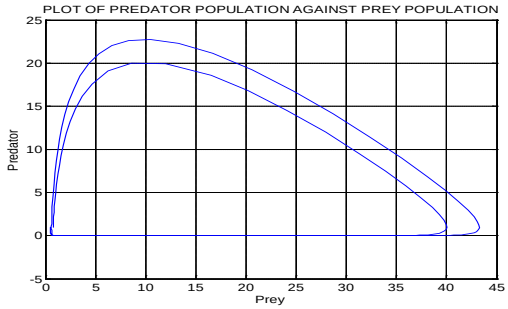
$$y_2' = -10y_2 + y_1y_2$$

$$y_1(0) = 0.5\,,\ y_2(0) = 1$$

Implementing ode45 for the problem:



Phase Portrait of the System



Problem 3:

Lorentz Equations

The Lorentz equations are used to simulate the convection of a layer of fluid of infinite horizontal extent heated from below. The model is a simplified version of the heating of the atmosphere. The equations are obtained by expanding the terms for temperature and pressure involved in the problem with their Fourier series expansion and simplifying the expansion to the first three modes represented by the variables x, y, and z. The resulting system of equations is

$$\frac{dx}{dt} = \sigma(-x + y)$$

$$\frac{dy}{dt} = rx - y - xz$$

$$\frac{dz}{dt} = xy - bz$$

where

σ, r, and b are constants that result from combining physical parameters of the problem. Solve the Lorentz equations for the following combination of parameters:

$$\sigma = 10, r = 75, b = 2.666, x(0) = 1, y(0) = 1, z(0) = 1$$

Plot the signals $x - vs - t, y - vs - t, z - vs - t$, as well as the phase portraits $x - vs - y, x - vs - z$, and $y - vs - z$.

Source: Gilberto (2004)

Solution:

Our system of ODEs now become:

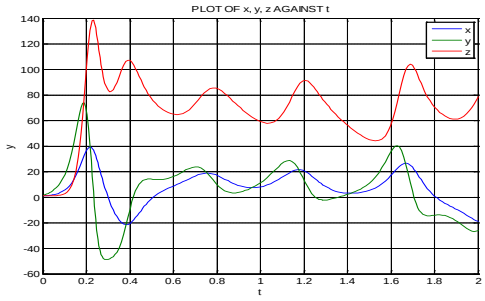
$$\frac{dx}{dt} = 10(-x + y)$$

$$\frac{dy}{dt} = 75x - y - xz$$

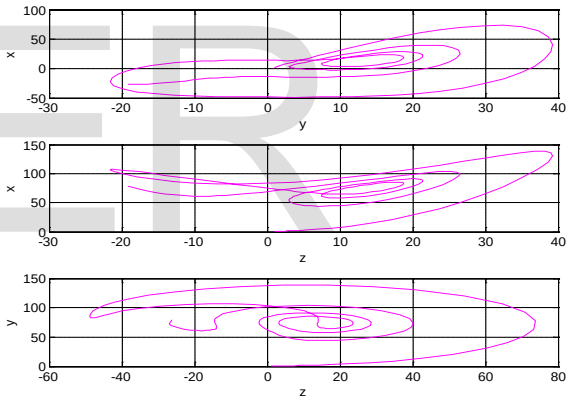
$$\frac{dz}{dt} = xy - 2.666z$$

$$x(0) = 1, y(0) = 1, z(0) = 1$$

Solving the problem using ode45:



Phase Portraits of the System



Comparison of ode45, ode23 and ode113 for Problems 1, 2, and 3

	Solver	Elapsed Time (s)	Successful Steps	Failed Attempts	Function Evaluation
Problem 1	**Ode45**	0.020157	40	6	277
	Ode23	0.540800	87	9	289
	Ode113	0.489211	90	5	186
Problem 2	**Ode45**	0.033996	76	8	505
	Ode23	Failed			
	Ode113	Failed			
Problem 3	**Ode45**	0.025690	49	8	343
	Ode23	0.035048	145	14	478
	Ode113	0.054591	140	10	291

APPLICATION OF MATLAB ODE SOLVERS TO STIFF INITIAL VALUE PROBLEMS OF ODEs.

Problem 4.

Van der Pol System

The Van der Pol system is a non-linear oscillator. This is a classical stiff problem in the field of electrical engineering and was first reported in 1927 by Balthazar Van der Pol and his colleague Van der Mark**.** The mathematical representation of the system is:

$$y_1' = y_2$$

$$y_2' = \alpha(1 - y_1^2)y_2 - y_1$$

The stiffness can be controlled by changing the parameter α. $\alpha = 1000$ will result in a stiff case. We shall solve the system with the initial conditions $y_1(0) = 2$ and $y_2(0) = 0$ on the interval 0 to 3000.

Source: Yihai (2000)

Solution

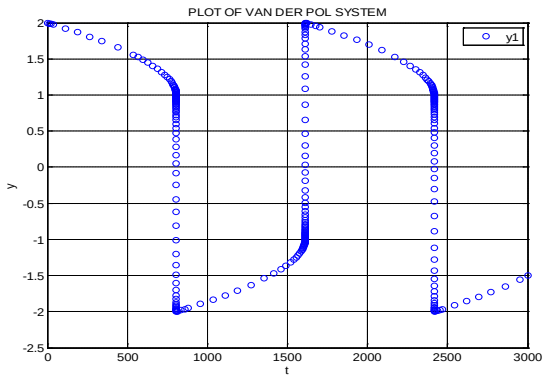
Our system of ODEs now become:

$$y_1' = y_2$$

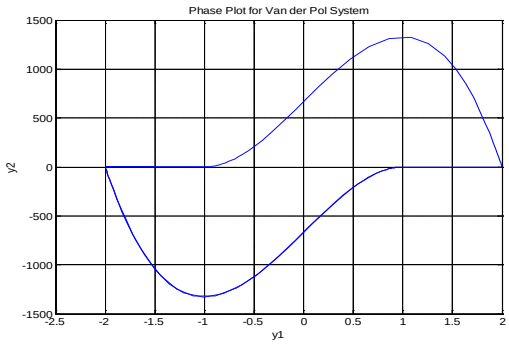
$$y_2' = 1000(1 - y_1^2)y_2 - y_1$$

$$y_1(0) = 2,\ y_2(0) = 0$$

Solving with ode15s:



Phase Portrait of the System



Problem 5:

The Robertson's Problem

Consider the system below on the interval $0 \leq t \leq 5000$

$$y_1' = -.04y_1 + 10^4 y_2 y_3$$

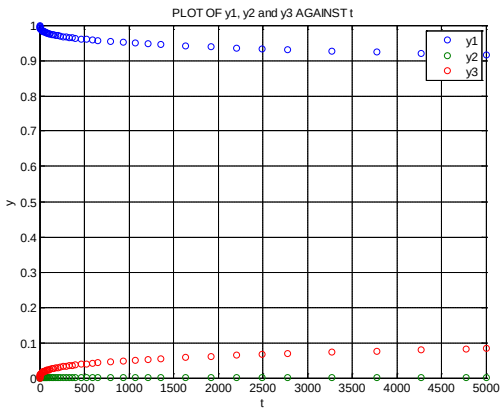
$$y_2' = .04y_1 - 10^4 y_2 y_3 - 3.10^7 y_2^2$$

$$y_3' = 3.10^7 y_2^2$$

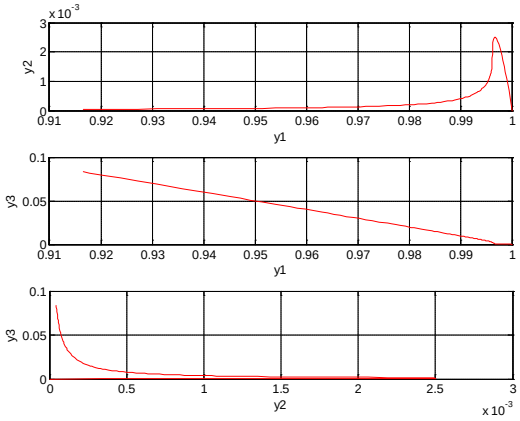
$$y_1(0) = 1, y_2(0) = 0, y_3(0) = 0$$

Solution:

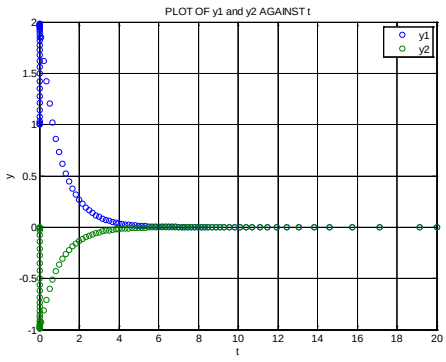
Solving with ode15s:



Phase Portraits of the System



Solution: Solving with ode15s



Problem 6

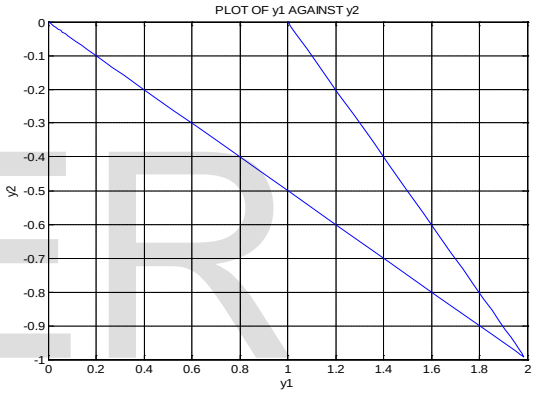
Consider the system of stiff differential equations on the interval $0 \leq t \leq 20$

$$y_1' = 998y_1 + 1998y_2$$

$$y_2' = -999y_1 - 1999y_2$$

$$y_1(0) = 1, \; y_2(0) = 0$$

Phase Portrait of the Solution



Comparison of ode15s, ode23s and ode23t for Problems 4, 5, and 6

	Solver	Elapsed Time (s)	Successful Steps	Failed Attempts	Functions Evaluated	Linear Systems Solved
Problem 4	**Ode15s**	0.424216	591	225	1883	1747
	Ode23s	0.549822	743	17	3751	2280
	Ode23t	0.520989	776	94	2121	2012
Problem 5	**Ode15s**	0.067788	107	11	261	227
	Ode23s	0.705399	93	2	565	285
	Ode23t	0.520620	99	8	284	250
Problem 6	**Ode15s**	0.049062	90	1	153	149
	Ode23s	0.053143	68	0	342	204
	Ode23t	0.066586	108	0	206	202

CONCLUSION

The use of MATLAB is very effective in proffering numerical solution to initial value problems of ordinary differential equations as seen in this work. Calculations that could run through pages and days are obtained in a matter of seconds. Accuracy is also maintained. Further studies on the use of optional parameters to enhance performance of solvers is recommended.

REFERENCES

Aliyu B. K., Osheku C. A., Funmilayo A. A. and Musa J.I. (2014) Identifying Stiff Ordinary Differential Equations and Problem Solving Environments (PSEs). *Journal of Scientific Research & Reports*. 3(11): 1430 - 1448

Bilokon P., Amen S., Brinley Codd A., Fofaria M., Shah T. (2004). *Numerical Solutions of Differential Equations.* Retrieved March 3, 2014 from www.oc.ic.ac.uk/~pb401/DE/archive/report/report.pdf

Butcher J.C. (2000) Numerical Methods for Ordinary Differential Equations in the 20th Century. *Journal of Computational and Applied Mathematics* 125:1-29

Coddington Earl A. and Levinson Norman (1955) *Theory of Ordinary Differential Equation. (*London, McGraw-Hill Book Company, Inc.)

David Houcque. (2014). *Applications of MATLAB: Ordinary Differential Equations (ODE).* Retrieved March 30, 2014 from www.math.unipd.it/~alvise/CS_2008/ODE/MFILES/ode.pdf

Didier Gonze. (2013). *Numerical Methods for Ordinary Differential Equations*. Retrieved March 10, 2014 from http://homepages.ulb.ac.be/~dgonze/TEACHING/numerics.pdf

Gilberto E. Urroz. (2004). *Examples of Initial-Value ODE Problems.* Retrieved April 5, 2014 from ocw.usu.edu/Civil_and_Environmental_Engineering/Numerical_Methods_in_Civil_Engineering/ODEsExamples.pdf

James Blanchard. (1998). *Ordinary Differential Equations: Initial Value Problems.* Retrieved March 19, 2014 from http://homepages.cae.wisc.edu/~blanchar/eps/ivp/ivp.html

Kendall Atkinson, Weimin Han, David Stewart (2009) Numerical Solutions of Ordinary Differential Equations. (New Jersey, John Wiley & Sons, Inc.)

Shampine L. F., Gladwell I., Thompson S. (2003) *Solving ODEs in MATLAB.* (New York, Cambridge University Press)

Marcel B. Finan. (2012). *A first Course in Elementary Differential Equations.* Retrieved March 30, 2014 from www.math.umass.edu/~gardner/m331/diffq1book.pdf

Mathworks. (2014). MATLAB, *The Language of Technical Computing.* Retrieved April 3, 2014 from http://www.mathworks.com/products/matlab/index.html?sec=applications

Ogbonnaya, I. O. (2008) *Introduction to MATLAB/SIMULUNK for Engineers and Scientists.* (Enugu, John Jacob's Classic Publishers Ltd.)

Ogunride R. Bosede, Fadugba S. Emmanuel, Okunlola J. Temitayo (2012) On Some Numerical Methods For Solving Initial Value Problems In Ordinary Differential Equations. *IOSR Journal of Mathematics*. 1(3):25-31

Paul's Online Math Notes. (2014). *Differential Equations-Notes.* Retrieved March 13, 2014 from tutorial.math.lamar.edu/Classes/DE/Definitions.aspx

Yihai Yu. (2000) *Stiff Problems in Numerical Simulation of Biochemical and Gene Regulatory Networks.* Retrieved April 5, 2014

From www.getd.libs.uga.edu/pdfs/yu_yihai_200408_ms.pdf

Robert E. Terrell (2014). *Lecture Note on Differential Equations.* Retrieved March 9, 2014 from http://www.math.cornell.edu/~bterrell/dn.pdf

Ryuichi Ashino, Michihiro Nagae and Remi Vaillancourt (2000) Behind and Beyond MATLAB ODE Suite. *Computers and Mathematics with Applications*. 40:491-512

Sharaban Thohura & Azad Rahman (2013) Numerical Approach for Solving Stiff Differential Equations: A Comparative Study. *Global Journal of Science Frontier Research Mathematics and Decision Sciences.* 13(6): 6 - 18

Steven T. Karris (2007) *Numerical Analysis Using MATLAB and Excel* (California: Orchard Publications)

Toshinori Kimura. (2009). *On Dormand-Prince Method.* Retrieved April 27, 2014 from http://depa.fquim.unam.mx/amyd/archivero/DormandPrince_19856.pdf

Wikibooks. (2014). *MATLAB Programming.* Retrieved April 3, 2014 from http://en.wikiboos.org/wiki/MATLAB_Programming

Wikipedia. (2014). *Bogacki Shampine Method.* Retrieved April 27, 2014 from http://en.wikipedia.org/w/index.php?title=Bogacki-Shampine_method

Wikipedia. (2014). *Differential Equation.* Retrieved February 20, 2014 from http://en.wikipedia.org/wiki/Differential_equation

Wikipedia.(2014). *Initial Value Problem.* Retrieved February 20, 2014 from http://en.wikipedia.org/wiki/Initial_Value_Problem