Simulink: An Efficient Method for Solving Differential Equations.

C. Mabenga

Abstract— In this paper, we present an efficient graphical method for solving differential equations known as Simulink. The method is fast, simple and offers us the opportunity to spend more time exploring the behaviour of the solutions rather than in solving the differential equation as it is often the case with analytical and numerical methods. The differential equation is represented by means of a model diagram which comprises of three sections namely: the input, computational and output sections. A convenient approach in which the model diagram is drawn using the differential equation parameters instead of their values is considered. Three approaches for drawing the computational section of the model diagram namely; the differential equation approach, state-space approach and transfer function approach are studied. The steps followed starting from drawing the model diagram; simulation and display of the solutions are outlined. Ways of creating simulation values in the MATLAB workspace for further use and analysis are discussed. As an illustration, a mass-spring system equation driven by a sinusoidal force is solved using Simulink. Numerical results obtained are given.

Index Terms— Differential Equations, Simulink, Model Diagram, Mass Spring System

1 INTRODUCTION

Many real-world problems in the field of engineering and science are modeled using differential equations (DEs). Obtaining an analytical solution for many of these des is often a difficult or even an impossible task. In some cases even if an analytical solution exists, it is usually too complicated to interpret or involve integrals that can be evaluated only using numerical integration methods. It is in this sense that numerical methods are often regarded as a powerful alternative tool for solving DEs modeling practical problems [1, 2, 3]. However, numerical methods usually require lots of computations and function evaluations, thus time consuming. Computers are often used to speed up the process, but they require code writing and programming skills [2, 3]. In this paper, we present an effective graphical method for solving des called Simulink. Simulink which stands for SIMULATION and Link is a very powerful, simple and fast method that uses MATLAB as its computational engine [4, 5, 6]. Furthermore, Simulink offers us the opportunity to spend more time exploring the behaviour of the solutions rather than in solving the de as it is often the case with analytical and numerical methods [5]. In addition, Simulink requires no computation and function evaluations form the user nor programming skills and can solve all initial value problems of any order: linear or nonlinear, homogeneous or inhomogeneous with constant or variable coefficients. Simulink also offers a variety of forcing functions such as sinusoidal, step, logarithmic, exponential, wave functions and many others. In Simulink the de is represented by means of a model diagram which comprises mainly of three sections namely; the input, computational and output sections with solution is obtained by simulating the model diagram over time. The model diagram is drawn using blocks from the Simulink block libraries. A block is precisely seen as a device that performs a specific task on the input to produce an output [5, 6, 7]. Drawing a model diagram is simple, easy, takes little time and requires no special skills. To be precise, Simulink turns our computer into a laboratory for modeling systems allowing us to draw models as we would with pencil and paper [6]. Changes to DE's parameters values can be made and the solution obtained in no time.

In this paper, we consider a convenient approach in which the model diagram is drawn using parameters instead of their values. The model diagram is kept fixed once drawn and the parameter values passed to it using a MATLAB file or MATLAB prompt >>. Changes to parameter values are made in the file instead of model blocks thus making studying the behaviour of the solution as parameters are varied more efficient.

2. SECTIONS OF A MODEL DIAGRAM

The three sections of the model diagram are such that the input and output sections are for inputting source functions and displaying simulated solutions, respectively. The computational section is the main section of the model diagram and this is where the DE is solved. A complete model diagram is such that components of the three sections are connected together by means of interconnections [5]. Blocks

for drawing a model diagram for solving DEs are obtained from four main block libraries, namely; the *sources*, *sinks*, *continuous* and *Math Operations* libraries. A block is precisely seen as a device that performs a specific task on the input to produce an output [5, 6, 7].

The sources library comprises of blocks for signal generating devices and source functions. Examples include the constant (for inputting constants), step, sinusoidal, function blocks just to mention a few. The sinks library comprises of blocks for displaying simulated solutions. The four mainly used blocks include the *scope*, *To Workspace*, *To File* and *Out1* blocks. The continuous library comprises of blocks used to compute the solution of the DE. Three commonly used blocks are the *integrator*, *state-space* and *transfer function* blocks. The Math Operations library comprises of blocks used for computing arithmetic operations. Examples include the product, sum, quotient, difference, gain, trigonometric functions and math function blocks such as powers, surds, exponential and logarithmic functions [5, 7].

2.1 Drawing the Computational Section of a Model Diagram

We consider three approaches used for drawing the computational section of a model diagram.

2.1.1 Using the Differential Equation Approach

This is the most commonly used and straightforward approach in which the solution is obtained by integrating the DE using integrator blocks. An integrator block is considered as something that integrates the derivative of a signal to recover the signal [5, 7]. The highest order derivative of the DE is made the subject and the resulting equation is integrated to obtain the solution. For an n^{th} order DE, n integrator blocks are needed. The first block integrates the n^{th} derivative to obtain the n^{th-1} derivative and so on with the last one integrating the first order derivative to obtain the solution. The major setback with this approach is that it is space consuming and hence becomes inconvenient for higher order DEs. Convenient approaches such as using the state-space and transfer function of the DE are often preferred for higher order DEs.

2.1.2 Using the State-Space Approach

In this approach, an n^{th} order DE is converted to a system of n first order DEs having the matrix form

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}. \qquad (1)$$

System (1) is referred to as the state-space form of the DE. The first and second equations are called the state and output equations, respectively. The vectors x, and y are the state and the output vectors, respectively. The matrices A, B, C, and D are called the state, input, output and transition (direct action from input to output) matrices, respectively [7]. The computational section of the model diagram is drawn using the already available state-space block thus saving space [5].

2.1.3 Using the Transfer Function Approach

The DE is transformed into its transfer function form which is the ratio of the Laplace transform of the forcing function over the Laplace transform of the DE [5, 7]. It is given by $H(s) = \frac{X(s)}{F(s)}$ where $X(s)$, and $F(s)$ are the Laplace transforms of the DE and forcing function, respectively. The computational section is then drawn using the available transfer function block [5].

Like the state-space approach, the computational section is put in one block thus saving space. The major setback with this approach is that it only applies to zero initial conditions [5, 7].

A schematic diagram for solving DEs in Simulink is given in Figure 1

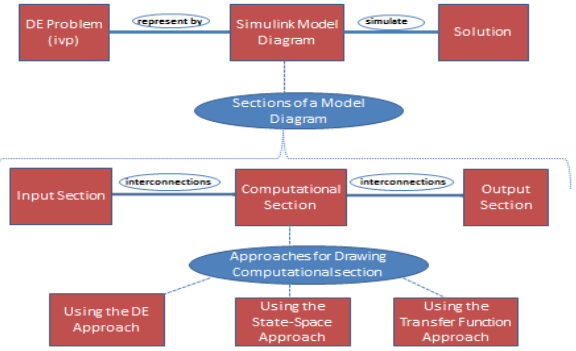

Figure 1. Simulink Schematic Diagram for solving a DE

3 STEPS FOLLOWED: SOLVING A DE IN SIMULINK

Steps followed when solving a DE in Simulink are generalized as: Accessing Simulink, Drawing model diagram Simulating model diagram, and Visualizing simulation results.

3.1 Step 1: Accessing Simulink

- Open MATLAB.

• Open Simulink by double clicking on the Simulink icon on the MATLAB menu. Alternatively, type ***Simulink*** at the MATLAB prompt >> followed by ***Enter***.

3.2 Step 2: Drawing the Model Diagram

• Decide on the approach to use to draw the computational section and the output block(s) to use. This is done outside Simulink.

• Open a new model by double clicking on the ***new model*** icon.

• Copy and drag all the required blocks from their respective block libraries to the new model.

• Align and orient blocks for connection. Right click on the block and use the ***flip block*** option from ***format*** to change the block's orientation.

• To connect two blocks left click and drag from the input port of one block to the output port of another block or vice versa. To connect a signal line to a block, left click on the block input port and drag to the signal line. A solid black line with a bold arrow at the end symbolizes connection success while a red broken line implies connection failure.

• Reset block parameter values. The simulation start and stop times, solver and its type can also be reset using ***Model Configuration Parameters*** from ***simulation*** from the model menu. The solver is the numerical method used to solve the DE problem. Simulink has two types of solvers being the fixed-step and variable-step. A fixed-step solver uses a constant step size throughout the simulation time while a variable-step solver varies the step size during simulation, reducing the step size to increase accuracy when a model's states are changing rapidly and increasing the step size to avoid taking unnecessary steps when the model's states are changing slowly [5]. By default, MATLAB versions such as R2011b and earlier ones set the simulation start and stop times as0, and 10 seconds, respectively and use the variable-step solver*ode*45. Latest versions such as R2016a use the default solver ***auto***. Auto solver means that Simulink chooses a suitable solver for the model and sets the step size to maximum [5]. Other parameters such as the step size and tolerance error can also be reset.

3.3 Step 3: Simulating the Model Diagram

The model diagram is saved first before simulation. Two file formats ***.mdl***, and ***.slx*** are used with the latter introduced in MATLAB R2012a and is the default file format for MATLAB R2012b and latest versions [5].

• Save the model as ***'model-name'.mdl*** or ***'model-name'.slx***.

• Simulate the model using the ***start simulation*** button in

the model menu. The model can also be simulated using an m-file or MATLAB prompt >> using the command ***sim('model-name').***

3.4 Step 4: Visualizing Simulation Results

The scope block is the commonly used block**.**

• For MATLAB R2011b and earlier versions, double click on the ***scope block*** to produce a figure of simulation results. In latest versions such as MATLAB 2016a, the scope block produces simulation results figures automatically when the model is simulated.

3.5 Creating Simulation Data in MATLAB Workspace

Often we wish to create simulation results in the MATLAB workspace for further use and analysis. This can be done in two ways. The first one is importing simulation data values from the scope block. The second one is using other output displaying blocks such as the ***To Workspace***, ***To File*** and *Out1* blocks. These blocks automatically create data values in the workspace when the model is run. The created data values are accessed using the MATLAB prompt >> or an m-file by the command ***sim('model-name')*** [5].

3.5.1 Importing Data from the Scope Block

On the scope figure toolboxes select ***Parameters***, followed by ***Data History*** and ***save data to workspace.*** Edit the ***variable-name*** (default variable name is ***ScopeData***) and change ***save format*** to ***Structure With Time*** for models saved using the .mdl format. For models saved using the .slx format on the ***scope figure toolbox*** select ***Configuration Properties*** followed by ***Data Logging*** and ***log data to workspace*** [5]. The simulation times and their corresponding solution values are obtained using ***t = ScopeData.time*** and ***y = variable-name.signals.values,*** respectively.

3.5.2 Using the To Workspace, To File and Out1 Blocks

The process of accessing simulation data created using the To Workspace, To File and Out1 blocks for models saved using the .mdl format is given in Table 1.

Block	Changes on block parameters	Output values
To Workspace	•Edit variable name (default variable name: ***simout***)	• y = variable-name.signals.values
To File	•Change file name from ***untitled.mat*** to name of m-file •Change save format to ***Timeseries***	• y = simout
Out1	• None	• y = yout

Table 1. Accessing simulation values for .mdl models

For models saved using the .slx format a similar process is followed. For the To Workspace block, in addition we change the save format of the block from ***Timeseries*** to ***Structure With Time***. For the ***To File*** and ***Out1*** blocks the only change is that the output is obtained using ***y = simout.signals.values*** and ***y = yout.signals.values,*** respectively. The simulation times values are obtained using ***t = tout*** for all the three blocks.

4 Solving the Mass Spring System Using Simulink

We consider a damped driven mass spring system given by the second order linear constant coefficients differential equation

$$m\ddot{x}(t) + c\,\dot{x}(t) + k\,x(t) = F(t), x(0) = x\_0, \dot{x}(0) = v\_0 \qquad (2)$$

where m is the mass, $x(t)$ the displacement, $c \geq 0$ the damping constant, $k > 0$ the spring constant, and $F(t)$ the driving force.

4.1 Computational Section for the System

4.1.1 Using the Differential Equation Approach

Making the highest order derivative of the DE (2) the subject we obtain $\ddot{x}(t) = \frac{1}{m}[F(t) - c\dot{x}(t) - kx(t)]$.

Two integrator blocks are needed with the first one integrating $\ddot{x}(t)$ to obtain $\dot{x}(t)$ and the second one $\dot{x}(t)$ to obtain the solution $x(t)$.

4.1.2 Using the State-Space Approach

The DE (2) has order two so we introduce two new variablesx_1, andx_2. Letting$x_1 = x$, $x_2 = \dot{x}$ and differentiating with respect to time, t gives the system of first order DEs

$$\begin{cases} \dot{x_1} = \dot{x} = x_2, \\ \dot{x_2} = \ddot{x} = \frac{1}{m}[F(t) - kx_1 - cx_2] \end{cases} \qquad (3)$$

having the matrix form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-k}{m} & \frac{-c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t) \qquad (4)$$

which simplifies to the state eqution $\dot{x} = Ax + Bu$ with $u = F(t)$. For the output equation given by $y = Cx + Du$ we have different matrices C depending on the output desired. If the output is the displacement, x, then $y = x_1$giving $C = [1\ 0]$. If the output is the velocity,$\dot{x}$, then $y = x_2$ giving $C = [0\ 1]$. If the output is both the displacement and velocity, the matrix C is given by $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. We assume no direct action of the input on the output and take matrix D as zero.

4.1.3 Using the Transfer Function Approach

Taking the Laplace transforms of (2) both sides and using zero initial conditions gives the transfer function as

$$H(s) = \frac{X(s)}{F(s)} = \frac{1}{ms^2 + cs + k} \qquad (5)$$

A model diagram for the system (2) with the computational section drawn using the three approaches is given in Figure 2.

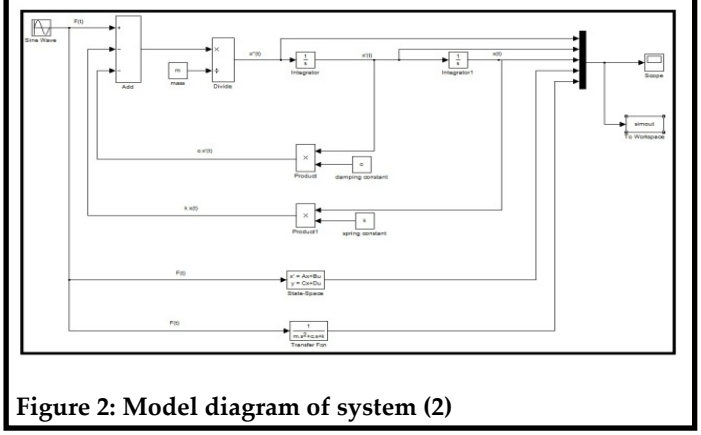

Figure 2: Model diagram of system (2)

The model diagram is drawn using parameters instead of their values, saved as ***MassSpringModel.mdl*** and simulated using an m-file with the solver ode23s. The file is given in Appendix

A. The scope and To Workspace blocks are used to display the results with the latter having block parameters set as described in Table 1.

5. NUMERICAL RESULTS

A mass spring system with parameter values: $m = 1\,kg,\ k = 3\,N/m, c = 4\,Ns/m, x(0) = 0.5m, \dot{x}(0) = 1\,m/s$ and $F(t) = 5\cos(2t)$ is considered. Simulink does not have the cosine function block so the sine function block is used with the source function expressed as $F(t)5\sin\left(2t + \frac{\pi}{2}\right)$. The state-space, transfer function and source function block parameters are set as: in Figure 3.

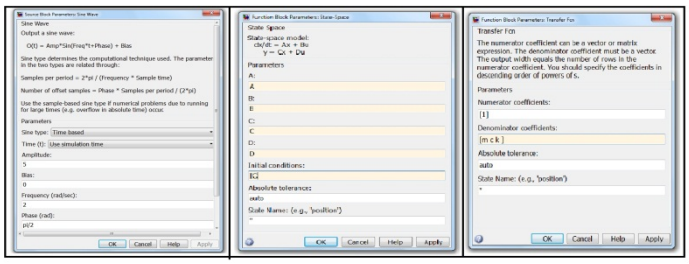

Figure 3. Block Parameters for $F(t)$, state-space and transfer function

The results obtained by simulating the model are given in Figure 4.

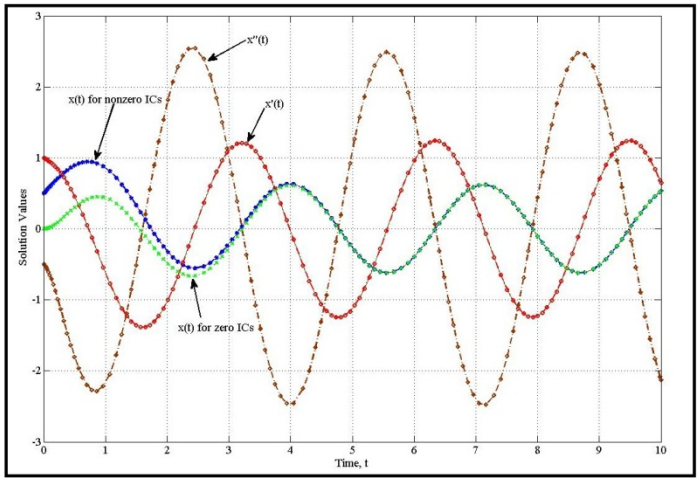

Figure 4. Displacement, velocity and acceleration results

6. CONCLUSIONS

In this work we have demonstrated that the model diagram of a DE problem can be drawn using parameters and simulated using an m-file with simulated results created in the MATLAB workspace and plotted using the file. The Simulink method is convenient as for the mass spring system the displacement, velocity and acceleration results can be obtained at the same time with ease. The acceleration results are only obtained by using the DE approach while the transfer function approach only gives the displacement. The results produced by the plot command using the m-file are identical to those from the model diagram scope block.

APPENDIX A

The m-file used to simulate the model is given as:

```
%M-file for Simulating mass spring system mx''(t) + %cx'(t) +
kx(t) = 5cos(2t) model
clear all %clears all saved variable values
clc %clears the workspace
m=1;c = 4;k = 3;x_0 = 0.5;v_0 = 1;%system parameters
F_0=5;w=2;phi = pi/2;% F(t)=F_0 sin(wt+phi) values
A=[0 1;-k/m -c/m];B=[0;1/m];C=[1 0; 0 1];D=[0 0]';
IC=[0.5 1]';% State space block parameter values
sim('MassSpringModel');% simulating model diagram
t=tout;%To Workspace block simulation times
y=simout.signals.values;%To Workspace simulation values
plot(t,y)%plotting simualtion results
xlabel('Time, t')
ylabel('Solution Values')
grid on
```

REFERENCES

[1] F. Kenmogne. Generalizing of Differential Transform Methods for Solving Nonlinear Differential Equations, *Journal of Applied and Computational Mathematics*, 4 (1) 2015.

[2] M. K. Ring and N. A. Mody, *Numerical and Statistical Methods for Bioengineering: Application in MATLAB*, Cambridge University Press, UK, 2011.

[3] K. Atkinson, W. Han and D. Stewart, *Numerical Solutions of Ordinary Differential Equations*, John Wiley and Sons, Inc, New Jersey, USA, 2009.

[4] D. Xue and Y. Chen, Solving *Applied Mathematical Problems With MATLAB*, Chapman and Hall/CRC, New York, USA, 2009.

[5] MATLAB *website: www.mathworks.com*.

[6] W. J. Palm III, *Introduction to MATLAB for Engineers*, Third Edition, McGraw-Hill, New York, USA, 2011.

[7] E. M. Davis, *Numerical Methods and Modeling for Chemical Engineers*, John Wiley and Sons, Inc, USA, 1984.

[8] B. R. Hunt, R. L Lipsman, J. E. Osborn and J. M. Rosenberg, *Differential Equations With MATLAB*, Second Edition, John Wiley and Sons, Inc, USA, 2005.

International Journal of Scientific & Engineering Research Volume 9, Issue 10, October-2018 1739
ISSN 2229-5518