XPPWEB: PORTING XPP TO THE WEB

by

Derek Pawlush

Bachelor of Arts, Franklin & Marshall College, 2015

Submitted to the Graduate Faculty of

the Dietrich School of Arts & Sciences in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2018

UNIVERSITY OF PITTSBURGH

THE DIETRICH SCHOOL OF ARTS & SCIENCES

This thesis was presented

by

Derek Pawlush

It was defended on

December 6th 2018

and approved by

Bard Ermentrout, Distinguished University Professor

Jonathon Rubin, Professor and Department Chair

Brent Doiron, Professor

Thesis Advisor: Bard Ermentrout, Distinguished University Professor

XPPWEB: PORTING XPP TO THE WEB

Derek Pawlush, M.S.

University of Pittsburgh, 2018

Ordinary differential equations (ODEs) arise in many areas of mathematics, especially in mathematical modeling and engineering problems. Naturally, solving these types of problems is of great interest due to the practical implications involved. Solving a system of ODEs is often very complicated and in a multitude of instances there is no explicit solution to the system. However, by coupling numerical methods to solve a system of ODEs and dynamical systems techniques centered around visualization, we may gain a meaningful understanding of how the system behaves. XPP is a native software for numerically solving and plotting differential equations [1]. The aim of this project is to extend XPP and create a web application, XPPWeb, with an emphasis on an intuitive user interface and robust plotting features. Case studies of relevant and interesting systems of equations are also included to exhibit the capabilities of XPPWeb. The application will be accessible at https://xppweb.math.pitt.edu/.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

Solving ordinary differential equations (ODEs) is pivotal in many branches and practical applications of applied mathematics. Similar to normal algebraic equations, we want to solve ODEs for their dependent variables. However, more often than not, an analytic solution is nonexistent. In these cases, we need to utilize computers and numerical mathematics to numerically solve ODEs. There are many software applications that compute solutions to ODEs; in this project, we create a web application, XPPWeb, for the XPP software package[1] [1]. While offering all the core features of XPP, we provide a slick and painless user experience that software applications often times lack.

XPP is an excellent and incredibly flexible tool when working with differential equations. However a web based solution provides its own flexibility. In a web application there is no need to download or install any software—all that is required is a browser. This decouples our software from both specific types of device (i.e. phone, laptop, etc.), and operating systems. A user can simply go to the site and use the application, without the worries associated with correct installation and prerequisite software. This ease and lack of any virtually any commitment to the application makes the web based software extremely desirable and available.

XPP uses a distinct file format for ODEs. We offer file upload and editing areas for the XPP veterans who are familiar with XPP's file format, while (at the same time) offering a simple equation input area which the application uses to generate and run an ODE file for the user after the equations are entered. By providing both options, we are not alienating any new users who may be intimidated by having to learn the syntax of the ODE file, in addition to not rendering XPP users' files useless in XPPWeb. This also offers an opportunity to

[1] http://www.math.pitt.edu/~bard/xpp/xpp.html

slowly bridge the knowledge gap for new XPP users.

Matlab, Maple, and Mathematica are commercial software packages that include the option for web based ODE solving and plotting. While these programs offer more options, they tend to be expensive, whereas XPPWeb is a free application. Matlab, Maple, and Mathematica all require learning to program in their respective syntax. Many commands are needed to create desired plots, while XPPWeb requires as little as a file upload or typed equation, and a button click to generate plots. The whole application is streamlined and focused on solely solving and plotting ODEs. The opinionated design of the application produces swift results for the user. However, the lack of generality does have its drawbacks if something slightly outside the set of features is required.

2.0 BACKGROUND

2.1 MATHEMATICS

2.1.1 ODEs

An ordinary differential equation is a mathematical equation that relates some function of one independent variable with its derivatives, more formally,

$$x^{(n)} = F(t, x, x', ..., x^{(n-1)}). \tag{2.1}$$

Higher order differential equations can be written as a system of first order differential equations so (2.1) becomes,

$$\begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_n' \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ F(t, x_1, x_2, ..., x_{n-1}) \end{bmatrix} [5]. \tag{2.2}$$

In order to be able to solve for a particular solution, we require an initial condition of the dependent variable(s). There are algebraic techniques for solving simpler ODEs, but for more complicated and especially for nonlinear ODEs, many cannot be explicitly solved. In those cases we have to use numerical mathematics approaches to solve the equations.

2.1.2 Numerical Methods

To illustrate common numerical methods, we first examine Euler's method. Euler's method is a first order one step numerical method for solving ODEs. It is an iterative approach that starts at the initial condition and steps forward following the function defined in the ODE. Consider the ODE,

$$x' = f(t, x), \text{ where } x(t_0) = x_0. \tag{2.3}$$

Then Euler's method is defined as:

$$x_{n+1} = x_n + hf(t_n, x_n) \tag{2.4}$$

where h is the step size [4]. Euler's method is the simplest and often used as a basis for other numerical ODE solvers. XPP (and likewise XPPWeb) offers the option to choose between different solvers but uses Runge-Kutta by default. The Runge-Kutta method is defined as:

$$x_{n+1} = x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \text{ where} \tag{2.5}$$

$$k_1 = f(t_n, x_n) \tag{2.6}$$

$$k_2 = f(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}) \tag{2.7}$$

$$k_3 = f(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}) \tag{2.8}$$

$$k_4 = f(t_n + h, x_n + k_3). \tag{2.9}$$

While the performance of different numerical solvers generally depends on the differential equation at hand, Runge-Kutta is the standard ODE solver in many numerical packages.

2.1.3 Dynamical Systems

A system of ODEs in which time is the independent variable is commonly referred to as a dynamical system. Evolution over time is instrinsic to studying physical and biological systems. Accordingly, dynamical systems are an excellent mathematical tool for analysis and modeling in these domains. In this section, we outline some of the techniques and definitions in dynamical systems to clarify the complex workings of a particular system. Consider the dynamical system,

$$x' = f(x), \text{ where } x(t_0) = x_0, \tag{2.10}$$

where $t \in \mathbb{R}, x \in \mathbb{R}^n$, and $f \in C^k(M, \mathbb{R}^n), k \geq 1$, and M is an open subset of $\mathbb{R}$.

We define a trajectory, $\phi(t)$ where $\phi(0) = x_0$, as a particular solution to a dynamical system. Evolving a trajectory forward and backwards in time forms an individual solution to the dynamical system given the initial condition x_0 [5].

The direction field for the dynamical system is defined as the slope of the tangent to each point of the solution to the system. The formula for the direction field is given in the definition of the dynamical system as $f(x)$. By evenly sampling the direction field and drawing the vectors, we obtain an easily understood tracing of the various trajectories starting at different initial conditions.

An important tool in studying dynamical systems is the phase plane. The phase plane is a planar depiction of the direction field that is typically between two spatial variables. To illustrate phase plane definitions let us use the dynamical system,

$$x' = f(x, y) \tag{2.11}$$

$$y' = g(x, y) \tag{2.12}$$

where $x(t_0) = x_0$, $y(t_0) = y_0$, for $t, x, y \in \mathbb{R}$, and $f, g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The first definition we will use is the nullcline. The x and y nullclines are defined as,

$$f(x, y) = 0 \tag{2.13}$$

$$g(x, y) = 0 \tag{2.14}$$

, respectively. Since, $x' = f(x, y)$, by definition, along the line (2.13) there will be no change in the x direction. Similar logic follows for the (2.14) in the y direction. Because x and y nullclines do not change in their respective direction, drawing the direction field vectors at those points is trivial. It directly follows that at any intersections of equations (2.13) and (2.14), there is no vector going from that point. This intersection is known as a fixed point, or equilibrium point. For our example, we define an equilibrium point as $(x_e, y_e) \in \mathbb{R}$ such that (x_e, y_e) satisfies both (2.13) and (2.14) simultaneously. Equilibrium points have a notion of stability and that stability heavily governs the direction field, and by extension, the solutions, local to the fixed point. For brevity let us define,

$$\boldsymbol{v} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.15}$$

and

$$\boldsymbol{F}(\boldsymbol{v}) = \begin{bmatrix} f(x, y) \\ g(x, y) \end{bmatrix}. \tag{2.16}$$

Then a stable fixed point, $\boldsymbol{v_e}$, is defined that if for any neighborhood $U(\boldsymbol{v_e})$, there exists another neighborhood $V(\boldsymbol{v_e}) \subseteq U(\boldsymbol{v_e})$ such that any solution starting in $V(\boldsymbol{v_e})$ stays in $U(\boldsymbol{v_e})$ $\forall t \geq 0$ [5]. It has been shown that if the real part of the eigenvalues of the Jacobian of $\boldsymbol{F}(\boldsymbol{v_e})$ are all negative, then the equilibrium point is asymptotically stable. And if the eigenvalues are all positive, then the equilibrium point is asymptotically unstable.

2.2 TECHNOLOGY

2.2.1 Client-Server Model

The web application architecture follows a standard client-server model. In the client-server model, there are centralized servers that run code and do the bulk of the computation and clients, typically through web browsers, consume the services offered by the server[2]. In the server side of the model, we utilize a web server that acts a proxy server between the client

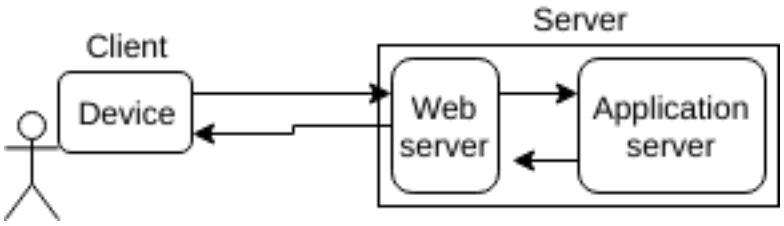

Figure 1: Client-Server Model

and the application server. The web server handles serving static content to the client and proxies other requests back to the application server that the web server is not meant to handle.

In the typical use case, the user will enter the URL into a web browser. The request for the web application will go through the Internet and be fielded by the web server. The web server will return HTML and JavaScript code for the client to run on the browser. Then throughout the user's interaction with the application, requests will be sent from the client back to the web server. The web server will then forward those requests back to the application server for processing. After processing, the application server will send the resulting data back to the web server which will forward the response back to the client.

2.2.2 RESTful Web Services

REpresentation State Transfer (REST) is a standard used for making an HTTP request for a service from a server. RESTful web services use uniform resource identifiers (URIs) along with an HTTP (GET, POST, PUT, DELETE) operation to define what service is being requested. For example, while the user is interacting with the application, clicking a button to generate plots for an ODE file will send a POST request with URI,
`https://xppweb.math.pitt.edu/rest/generatePlots`, to the server. The server side application will know how to interpret that request and return the plot data to the client in the response, accordingly. Using RESTful web services gives us something that is descriptive of the requested resource, lightweight, and very simple to consume from the client side.

3.0 METHODS

In this section we outline the specific technological implementations selected in order to create XPPWeb. The XPPWeb can be broken down into a two different pieces. The client side application which is the code that runs on the user's browser, handling the user interaction and plotting, while the server side application that does the processing and calculations.

3.1 CLIENT SIDE

3.1.1 React

The client side of the application is written entirely in JavaScript, heavily utilizing the JavaScript library React. The React library is used to build user interfaces in a declarative and component based manor. The client application is organized as a tree structure of small components cascading down from a single root component. This component based approach promotes code re-usability and separating concerns involved in different user interface aspects to its own component. In XPPWeb, the React client side application consumes RESTful web services to request data and processing from the server side.

3.1.2 Plotly

Plotly is an open source JavaScript plotting library. All the plots are created on the client side through Plotly. After React receives the plotting data in the response from the web service, React will feed the data to Plotly to create the different plots displayed in XPPWeb. A major advantage for having the client side, rather than the server side, generate the plots

is responsiveness. By using Plotly we take advantage of many built in features, such as zooming and panning without having to generate the plots every time we want a slight variant. This decreases loading time on more complicated plots.

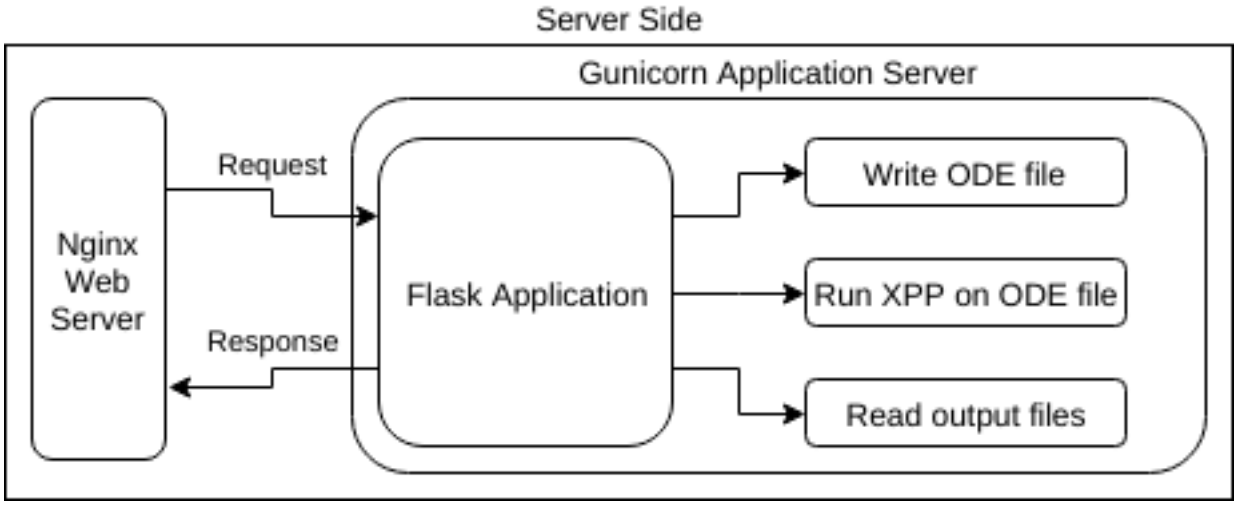

Figure 2: Server side diagram

3.2 SERVER SIDE

3.2.1 Nginx Web Server

Nginx is a web server we use to serve static content as well as act as a reverse proxy. When a user goes to access XPPWeb, Nginx initially serves the React client side application to the user. Then, as a proxy server, Nginx receives RESTful web services and forwards them onto the application server where the calculations happen.

3.2.2 Gunicorn

Gunicorn is a production grade Python HTTP server for unix that we use as the application server. The server side Python application runs on Gunicorn.

3.2.3 Flask

Flask is a Python microframework for web applications. Flask requires a minimal amount of configuration and code in order to implement a web application. We use Flask to setup

the RESTful web services for the client to consume. By utilizing Flask, we only need to consider the implementation for the functions that the web service calls. These operations are include to reading and writing text files, as well as running the XPP binary.

3.2.4 XPP

The compiled XPP binary acts as the engine of the XPPWeb. During the user's session inside the Python Flask application, we write ODE files in the XPP required format. Then Python's system libraries run the XPP binary using the ODE file. The resulting data files are read from Python and returned as part of the response to the client to be rendered in plots.

4.0 LOCAL INSTALLATION AND DEVELOPMENT

The code for XPPWeb is stored in the GitLab repository, https://gitlab.com/dpawlush/xppweb. The code may be run locally by downloading the xpp_web_local_run_only.zip file from the GitLab repository. To further develope the application, the entire repository must be downloaded. The entire repository is significantly larger due to the large amount of dependencies and other installations required for the JavaScript development environment. All development was done on a machine running Linux Mint 18.3 Cinnamon 64-bit. However, most Linux systems should be able to install and run the software required for both local hosting as well as development.

With the intention of running the application locally, or just Python development, the user may follow these steps:

1. Install python3.5
2. Python3 packages required to run:
    a. flask
    b. flask_cors
    c. paste
    d. scheduler
    e. subprocess
    f. asynchio
    g. os
    h. json
3. Download xpp_web_local_run_only.zip from GitLab repository, https://gitlab.com/dpawlush/xppweb

4. Extract xpp_web folder
5. Navigate to xpp_web folder and run python3 xpp_web.py
6. Visit http://127.0.0.1:5000/ to access local application

If the user plans to develop JavaScript code for the application, the user may follow the steps above with the exception of step 3. The user must pull down the entire repository rather than only the xpp_web_local_run_only.tar. Now the user may follow these steps:

1. Install nodejs version 8.11.2
2. Install npm version 6.4.1
3. Once those are installed, navigate to xpp_web/react folder
4. Once changes have been made to the JavaScript, then the user can run 'npm run build' in the command line to build the new JavaScript bundle
5. Revisiting http://127.0.0.1:5000/ should show the reflected changes now

5.0 CASE STUDIES

In the following two case studies we demo some functionality and capabilities of XPPWeb.

5.1 LORENZ SYSTEM

The famous 3 dimensional Lorenz System is described with the equations,

$$\frac{dx}{dt} = \sigma(y - x) \tag{5.1}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{5.2}$$

$$\frac{dz}{dt} = xy - \beta z \tag{5.3}$$

where $x_0 = y_0 = z_0 = 1$, and $\sigma = 10, \rho = 28, \beta = \frac{8}{3}$ [3].

The Lorenz Equations were derived as a system to model convection in the Earth's atmosphere. They are a primary example of a chaotic system for certain parameters and initial conditions. We generate a few different figures from XPPWeb to show some of the many capabilities. In figure 3, we show the plot of the solution, $z(t)$. Then in figure 4, we show both the $x(t)$ v $y(t)$ and $x(t)$ v $z(t)$ plotted on the same set of axes. Finally in figure 5, we show a 3 dimensional plot of $x(t)$ v $y(t)$ v $z(t)$.

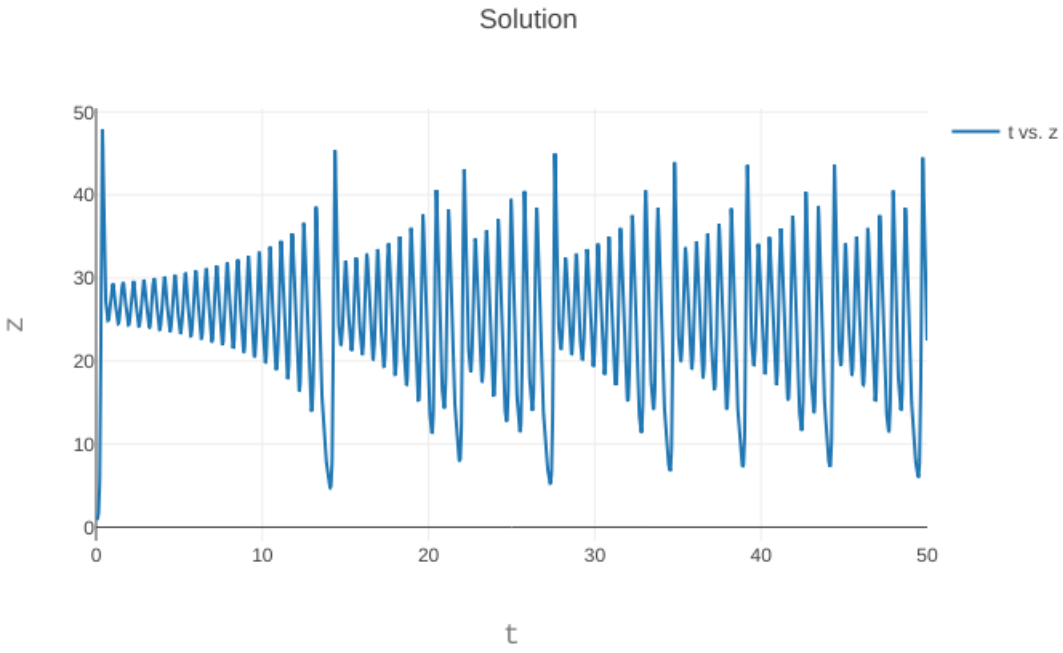

Figure 3: Lorenz System: Plot of $z(t)$ v t

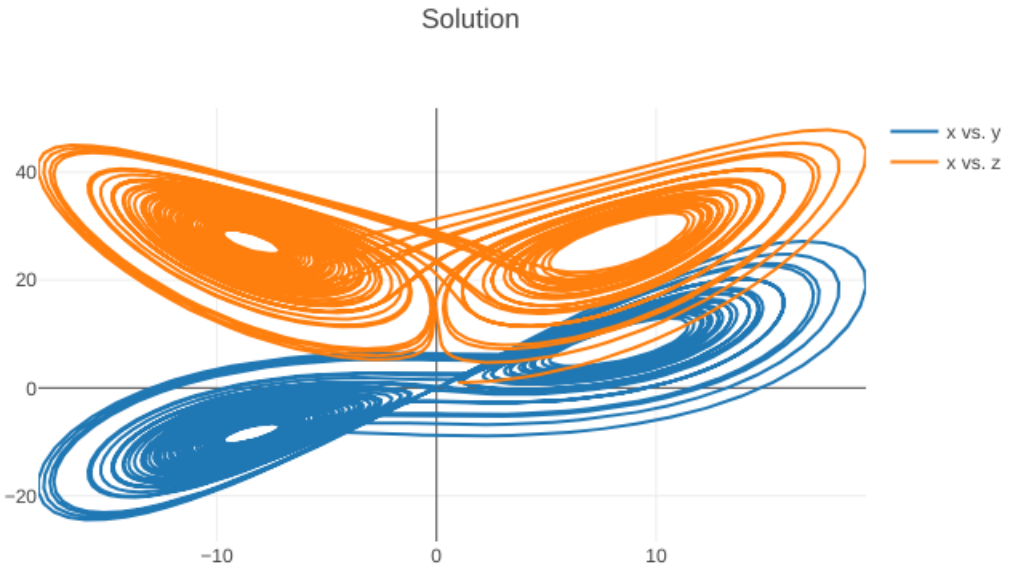

Figure 4: Lorenz System: Plot of $x(t)$ v $y(t)$ and $x(t)$ v $z(t)$ overlayed

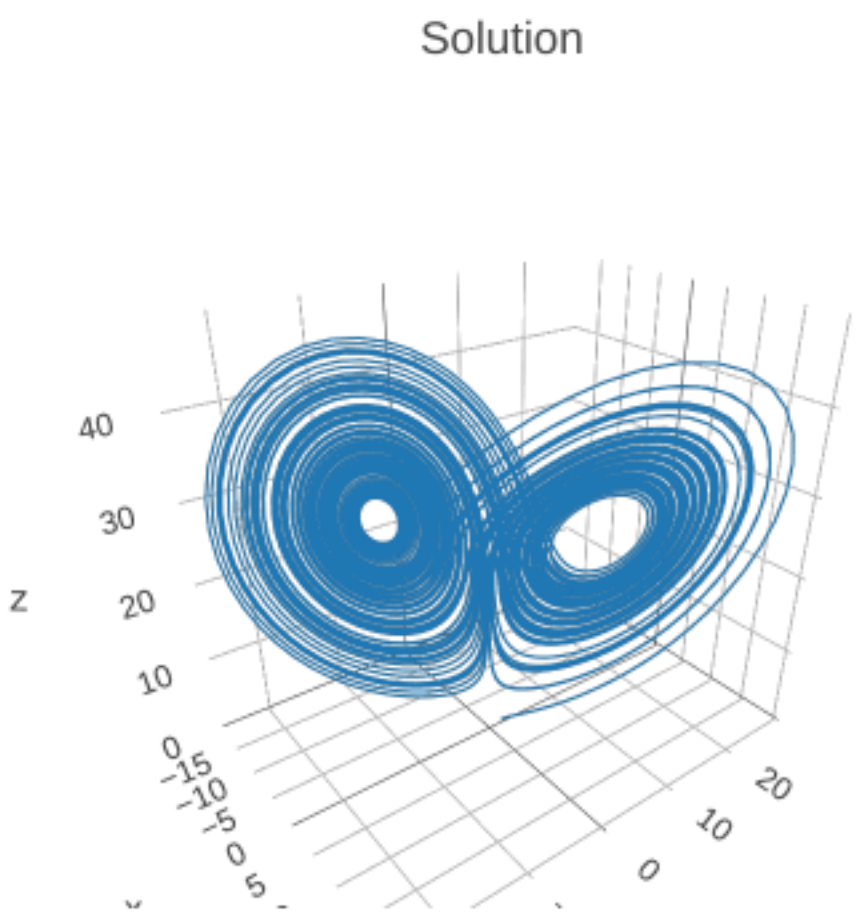

Figure 5: Lorenz System: 3 dimensional plot of $x(t)$ v $y(t)$ v $z(t)$

5.2 LOTKA-VOLTERRA EQUATIONS

The Lotka-Volterra, or more commonly know as the predator-prey model is defined as,

$$\frac{du}{dt} = \alpha u - \beta uv \tag{5.4}$$

$$\frac{dv}{dt} = \delta uv - \gamma v \tag{5.5}$$

where $u_0 = v_0 = 1$, and $\alpha = \frac{2}{3}, \beta = \frac{4}{3}, \gamma = 1, \delta = 1$.

In figure 6, we show a full screenshot of the application while plotting overlayed solutions to the Lotka-Volterra equations. We are also able to display the phase planes for the Lotka-Volterra model, illustrated in Figures 7 and 8. Figure 7 contains the u and v nullclines and the direction field, while Figure 8 shows off the capability for the user to generate trajectories by clicking on a point on the phase plane. The capacity to generate trajectories on-click is also available for on the solution panel of the application.

Figure 6: Lotka-Volterra Equations: Overlay of solutions in full screenshot of application

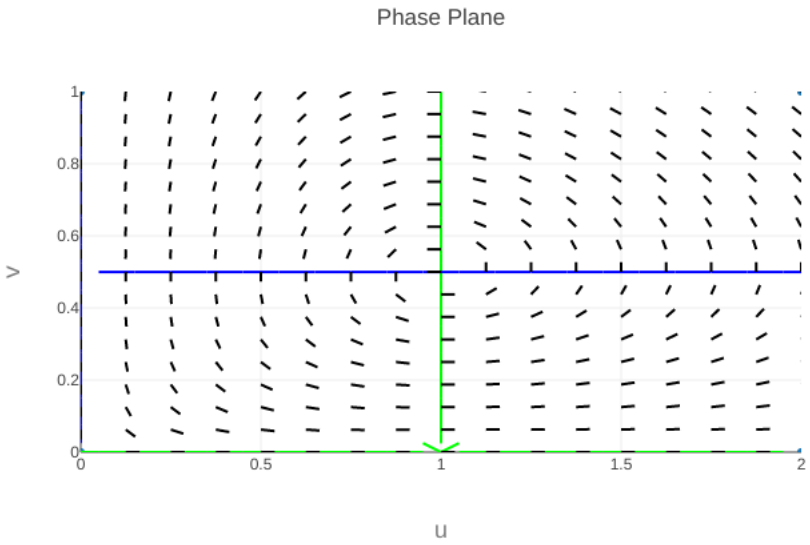

Figure 7: Lotka-Volterra Equations: Phase Plane view without any user generated trajectories

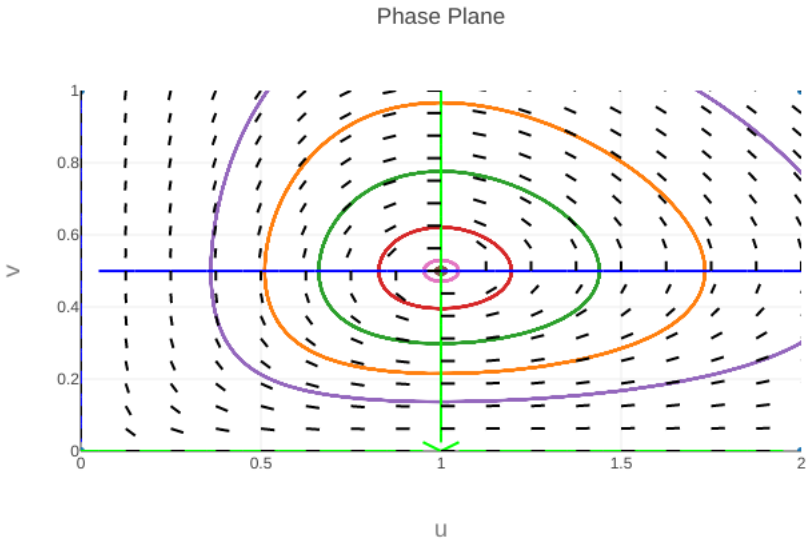

Figure 8: Lotka-Volterra Equations: Phase Plane view with user generated trajectories

6.0 CONCLUSION

XPPWeb offers a comparable and, in some cases, more streamlined experience compared to many other ODE solving and plotting software. In today's multitude of mathematical web applications, contemporary and cutting-edge technology choices and design ensure XPPWeb is a relevant option. A very robust and simple user interface allows new users and XPP veterans alike to appreciate the services offered by XPPWeb. Additionally, XPPWeb mirrors the core features of the native XPP application and admits future developments. As demonstrated in the case studies of two popular systems, we see the ease and opportunity to use XPPWeb as an aid in both the classroom and in academia. Some ideas for future extension include animation of solutions, and improved validation and error handling.

BIBLIOGRAPHY

[1] B. Ermentrout. *Simulating, analyzing, and animating dynamical systems: a guide to XPPAUT for researchers and students*, volume 14. Siam, 2002.

[2] J. F. Kurose and K. W. Ross. *Computer networking: a top-down approach*, volume 4. Addison Wesley Boston, 2009.

[3] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963.

[4] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010.

[5] G. Teschl. *Ordinary differential equations and dynamical systems*, volume 140. American Mathematical Soc., 2012.