European Society of Computational Methods in Sciences and Engineering (ESCMSE)

Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM)
vol. 3, no. 1-2, 2008, pp. 151-165
ISSN 1790–8140

VFGEN: A Code Generation Tool[1]

Warren Weckesser[2]

University of Sydney
School of Mathematics and Statistics
Sydney, Australia

Received 1 October, 2007; accepted in revised form 19 October, 2007

Abstract: VFGEN is a computer program that creates computer code for a wide variety of numerical software packages. It generates code for ordinary differential equations and for delay-differential equations. Symbolic differentiation is used to generate Jacobians and higher derivatives. From a single definition of the user's equations, VFGEN can generate code for initial value problem solver libraries, numerical continuation and bifurcation analysis programs, and general purpose computing environments. VFGEN also provides specialized commands for extending a vector field with its variational equation, converting delay equations to finite dimensional approximations, and for generating C code to compute Taylor polynomial approximations (of any given order) to the solution to a differential equation.

Keywords: software, ODE solvers, code generation, bifurcation, continuation, delay-differential equation

Mathematics Subject Classification: 34-04, 34K28, 37-04, 37M20, 65-04, 65P30

1 Introduction

There are many excellent programs available for solving differential and delay-differential equations and for computing their bifurcations. VFGEN is a program that generates computer code for a wide variety of these programs. To understand what it does and why it does it, we present three related problems that confront a user of these programs.

- Each program requires the creation of some form of *system definition file*, in which the equations are defined using the specific language and syntax of the program. For some programs, the system definition file can be quite complicated. Defining a set of equations correctly can be a significant obstacle to using the program.
- After using one program to study a system, a user may decide that it would be useful to use another tool. For example, after using the initial value problem solver CVODE to compute solutions to a system of differential equations, the user may decide to study the bifurcations

[1]Published electronically March 31, 2008
[2]E-mail about VFGEN may be sent to `vfgen.help@gmail.com`

of the system using MatCont. The fact that the user has already created a system definition file for CVODE (in this case, a C file) will provide very little advantage when creating the MatCont system definition file (a MATLAB function file).

- Differential and delay-differential equations are generally implemented as a vector field in the system definition file. Because most numerical algorithms that involve vector fields use the Jacobian matrix of derivatives of the vector field, many programs allow for the Jacobian of the vector field to be included in the system definition file. The programs generally use finite differences to approximate the Jacobian matrix if it is not provided in the system definition file, but an implementation of the analytically derived Jacobian can improve the speed, accuracy and reliability of the program. Programs that compute bifurcations will use even higher derivatives. MatCont, for example, allows for the definition of Hessians and higher order derivatives in its system definition file.

  Despite the benefit of providing analytical formulas for derivatives, it is not unusual for a user to not provide them, and let the program use finite difference approximations. For many users, the benefit of providing analytical derivatives is outweighed by the effort required to implement them in the system definition file.

VFGEN is an attempt to solve these problems. In particular:

- VFGEN uses a simple system definition file, which we will refer to as the *vector field specification file* (or simply the *vector field file*), to define the differential or delay-differential equations. This file can be interpreted as a form of "universal system definition file". The format of this file is described in Section 2.

- From the definition in the vector field file, VFGEN can generate system definition files for a wide variety of programs. Version 2.2.0 of VFGEN generates code for the software systems listed in Table 1. Once the vector field file is created, the user can, for example, issue one command to create a system definition file in C for the CVODE library, and then issue another command to create a MATLAB system definition file for MatCont.

- VFGEN uses the C++ symbolic algebra library GiNaC [2] to generate Jacobians and higher derivatives automatically. The user defines only the vector field; there is no need to enter the Jacobian or any higher order derivatives.

VFGEN is a command-line program, run from a terminal. The typical command has the form

```
$ vfgen <command> <vector_field_file>
```

where `<command>` is one of the commands in the third column of Table 1, and `<vector_field_file>` is the name of a vector field file. (The dollar sign, $, is the terminal prompt; it is not part of the command.) A detailed description of all the VFGEN commands listed in Table 1 will not be given in this paper. Instead, some of the more common commands are demonstrated in three case studies presented in Section 4.

VFGEN provides a few specialized commands that are not associated with a particular software package:

- `delay2ode`: Given a delay-differential equation, this command creates a system of ordinary differential equations whose solutions provide approximations to the solution of the delay equation.

- `evf`: This command generates a new vector field in which the given vector field has been extended with its variational equations.

Software	*Description*	*VFGEN Command*
ADOL-C [13]	Automatic differentiation library.	`adolc`
AUTO [6]	Continuation program.	`auto`
CVODE [4, 15]	The ODE solver library in the SUNDIALS suite.	`cvode`
DDE23 [19]	MATLAB delay equation solver.	`dde23`
DDE-BIFTOOL [8, 9]	MATLAB program for the bifurcation analysis of DDEs.	`ddebiftool`
DDE_SOLVER [23]	FORTRAN library for solving DDEs.	`dde_solver`
DSTool [1]	Environment for dynamical systems.	`dstool`
GNU Scientific Library [12]	Includes libraries for solving ODEs.	`gsl`
LaTeX [17]	The text processor.	`latex`
MatCont [5]	A MATLAB program for continuation.	`matcont`
MATLAB [19]	A general purpose computing environment; includes ODE solver libraries.	`matlab`
Octave [7]	A general purpose computing environment; includes ODE solver libraries.	`octave`
PDDE-CONT [22]	A tool for continuing periodic solutions to DDEs.	`pddecont`
PyDSTool [3]	An environment for analyzing dynamical systems.	`pydstool`
PyGSL [11]	The Python binding of the GNU Scientific Library; includes ODE solver libraries.	`pygsl`
RADAU5 [14]	A Fortran ODE solver.	`radau5`
Scilab [21]	A general purpose computing environment; includes ODE solver libraries.	`scilab`
SciPy [16]	Scientific libraries in Python; includes ODE solver libraries.	`scipy`
XPP [10]	A tool for studying dynamical systems, including ODEs and DDEs.	`xpp`

Table 1: Software systems for which VFGEN (version 2.2.0) generates code.

- `taylor`: The `taylor` command creates a C function that computes the Taylor approximation of the solution of the differential equations.

These are described in more detail in Section 3.

2 The Vector Field Specification File

A vector field file is an XML document in which the constants, parameters, expressions, and variables of the vector field are defined. Here we present the basic elements used to define a vector field, along with some of their attributes. A complete description and more examples are available on the VFGEN web page.

We start with an example. The differential equations for a simple pendulum are

$$\begin{aligned} \dot{\theta} &= v \\ \dot{v} &= -\left(\frac{b}{mL^2}\right) v - \frac{g}{L}\sin\theta \end{aligned} \tag{1}$$

where θ is the angle of the pendulum (measured from the straight down position), v is the angular velocity, g is the acceleration due to gravity, m is the mass of the pendulum, L is the length of the pendulum, and b is the coefficient of friction. The total energy of the pendulum is

$$E = \frac{1}{2}mL^2v^2 - mgL\cos\theta. \tag{2}$$

The key elements of this vector field are the variables θ and v along with the formulas for their derivatives. These derivatives are expressed in terms of the variables and the parameters g, b, L and m. Each of these mathematical objects has a corresponding element in the XML vector field file. For example, the parameter b is defined with the **Parameter** element. The simplest version of a definition of this parameter is

```
<Parameter Name="b" />
```

This tells VFGEN that b is a parameter of the vector field that can be used in the formulas for the vector field. When VFGEN creates code for a program or library that allows parameters to be passed to, say, the vector field function, b will be one of the parameters included in the list of parameters.

A variable is defined with a **StateVariable** element. For example, the simplest definition of the variable v is

```
<StateVariable Name="v" Formula="-b*v/(m*L^2)-(g/L)*sin(theta)" />
```

This line tells VFGEN that v is a variable in the vector field; the **Formula** attribute defines $\frac{dv}{dt}$.

Some of the VFGEN commands can create code that computes auxiliary functions specified by the user. Such an auxiliary function is defined with a **Function** element. For example, the energy can be defined as

```
<Function Name="energy" Formula="m*L^2*v^2/2 - m*g*L*cos(theta)" />
```

To complete the definition of the vector field, all the elements are enclosed in a **VectorField** element; the **Name** attribute of this element defines the name of the vector field. Figure 1 shows the file `pendulum.vf`, a complete vector field file for the pendulum equations. Additional attributes (mostly self-explanatory) have been included.

```
1  <?xml version="1.0" ?>
2  <VectorField Name="pendulum" Description="Pendulum Vector Field">
3  <Parameter Name="g"   Description="gravitational constant" DefaultValue="9.81" />
4  <Parameter Name="b"   Description="friction constant"      DefaultValue="0.0" />
5  <Parameter Name="L"   Description="pendulum length"        DefaultValue="1.0" />
6  <Parameter Name="m"   Description="mass"                   DefaultValue="1.0" />
7  <StateVariable Name="theta"  Description="Angle, measured from straight down"
8                 Formula="v"   PeriodFrom="0"   PeriodTo="2*Pi"
9                 DefaultInitialCondition="Pi-0.01" />
10 <StateVariable Name="v"      Description="angular velocity"
11                Formula="-b*v/(m*L^2)-(g/L)*sin(theta)"
12                DefaultInitialCondition="0.0" />
13 <Function Name="energy"   Description="total energy (kinetic plus potential)"
14           Formula="m*L^2*v^2/2 - m*g*L*cos(theta)" />
15 </VectorField>
```

Figure 1: The vector field file `pendulum.vf` that defines the vector field given in equations (1). The line numbers on the left are for reference only; they are not part of the file. Line 1 is a header that is required in all XML documents. The **VectorField** element is begun on line 2, and completed on line 15. It contains four **Parameter** elements (lines 3–6), two **StateVariable** elements (lines 7–12), and a **Function** element (lines 13–14).

As a second example, we consider the Mackey-Glass delay equation [18]. A nondimensional form of the equation is

$$\frac{dx}{dt} = -x + \alpha \frac{x(t-\tau)}{1+x(t-\tau)^{10}} \tag{3}$$

Delays are specified in the vector field file formulas with the function $\text{delay}(e, h)$. This function refers to the value of the expression e at time $t - h$. For example, $\text{delay}(x, 1)$ means $x(t-1)$, and $\text{delay}(xy, \tau)$ means $x(t-\tau)y(t-\tau)$. The variable x in the Mackey-Glass equation may be defined as

```
<StateVariable Name="x" Formula="-x+alpha*delay(x,tau)/(1+delay(x,tau)^10)" />
```

VFGEN allows intermediate expressions to be defined and used in formulas. These are defined with the **Expression** element. A simple example is

```
<Expression Name="delayedx" Formula="delay(x,tau)" />
```

With this definition, the symbol `delayedx` can be used in formulas instead of `delay(x,tau)`. (A less trivial example of the use of **Expression** elements is shown in Figure 4.)

Figure 2 shows a vector field file for the Mackey-Glass equation. This file uses the **Expression** `delayedx` in the **Formula** for x. It also uses the optional **StateVariable** attributes **DefaultInitialCondition** and **DefaultHistory** to define the values of $x(t)$ for $t \leq 0$.

3 Specialized Commands: `evf`, `delay2ode`, `taylor`

The three VFGEN commands `evf`, `delay2ode`, and `taylor` are not associated with a separate software package. The first two transform a given vector field into a new vector field, and the third generates C code to compute the Taylor polynomial of a solution to a differential equation. These commands are discussed in more detail in this section.

```
1  <?xml version="1.0"?>
2  <VectorField Name="MackeyGlass">
3  <Parameter Name="alpha" DefaultValue="2" />
4  <Parameter Name="tau"   DefaultValue="1.0"  Description="Delay time" />
5  <Expression Name="delayedx"  Formula="delay(x,tau)"  Description="x(t-tau)" />
6  <StateVariable Name="x"  Formula="-x+alpha*delayedx/(1+delayedx^10)"
7                 DefaultInitialCondition="0.1" DefaultHistory="0.1" />
8  </VectorField>
```

Figure 2: The vector field file `mackeyglass.vf` for the Mackey-Glass equation (3). Note that the **Expression** called `delayedx` is defined in line 5 and used in the formula in line 6.

3.1 `evf`: Extending a Vector Field with its Variational Equation

The output of the `evf` command is a new vector field file in which the original vector field has been extended with its variational equations. That is, if the original system is

$$x'(t) = f(x), \quad x \in \mathbb{R}^n \tag{4}$$

then the vector field created by the `evf` command is the $2N$ dimensional system

$$\begin{aligned} x'(t) &= f(x) \\ \eta'(t) &= Df(x)\eta \end{aligned} \tag{5}$$

Often one is interested in how sensitive a solution is to a specific parameter. In this case, the variational equations of interest are

$$\begin{aligned} x'(t) &= f(x, \lambda) \\ \eta'(t) &= D_x f(x, \lambda)\eta + D_\lambda f(x, \lambda) \end{aligned} \tag{6}$$

where $\lambda \in \mathbb{R}$ is a parameter.

The VFGEN command `evf` transforms a given vector field file into either (5) or (6). The latter is created if the command option `par=lambda` is given, where `lambda` is the name of a **Parameter** in the vector field file.

Suppose, for example, we are interested in how a solution to the pendulum equation depends on the parameter b. The command

```
$ vfgen evf:par=b pendulum.vf > pendulum_evf.vf
```

creates the vector field file `pendulum_evf.vf`. This vector field has two additional variables, dtheta and dv. The differential equations for these variables are

$$\begin{aligned} \text{dtheta}' &= \text{dv} \\ \text{dv}' &= -\frac{g}{L}\cos(\theta)\text{dtheta} - \frac{b}{mL^2}\text{dv} - \frac{v}{mL^2} \end{aligned} \tag{7}$$

The only difference between `pendulum.vf` and `pendulum_evf.vf` is that the latter has two additional **StateVariable** elements:

```
<StateVariable Name="dtheta" Formula="dv"/>
<StateVariable Name="dv" Formula="(-L^(-1)*g*cos(theta))*dtheta + (-L^(-2)*b*m^(-1))*dv
                                                               + (-L^(-2)*v*m^(-1))"/>
```

Upon solving this system with initial conditions `dtheta(0)=0` and `dv(0)=0` (using, say, an ODE solver with code created by applying VFGEN to the extended vector field), the final values of `dtheta` and `dv` give the derivatives with respect to b of the final values of θ and v.

3.2 `delay2ode`: Approximations of the Delay Operator

Consider a given function $x(t)$, and define $y(t)$ by

$$y(t+\tau) = x(t) \tag{8}$$

Then we can (trivially) rewrite the delay equation

$$x' = f(t, x, x(t-\tau)) \tag{9}$$

as

$$x' = f(t, x, y) \tag{10}$$

The idea that we explore here is to develop an approximation for $y(t)$ that will convert this equation into a system of ordinary differential equations. We present the basic idea here; a more thorough analysis will be published elsewhere.

Since τ is not necessarily small, we cannot expect a truncated Taylor series expansion of y at t to provide a good approximation for $y(t+\tau)$. However, we can define a sequence of intermediate functions as follows:

$$\begin{aligned} y_1(t+\tau/N) &= x(t) \\ y_2(t+\tau/N) &= y_1(t) \\ &\vdots \\ y_N(t+\tau/N) &= y_{N-1}(t) \end{aligned} \tag{11}$$

for an integer $N \geq 1$. Note that $y_N(t) = y(t)$. Expand each function on the left in a Taylor series, and truncate at order p:

$$y_k(t) + y_k'(t)(\tau/N) + \frac{1}{2}y_k''(t)(\tau/N)^2 + \cdots + \frac{1}{p!}y_k^{(p)}(t)(\tau/N)^p \approx y_{k-1}(t) \tag{12}$$

This is an ordinary differential equation of order p for $y_k(t)$, which can be written as a system of p first order differential equations. In this way we approximate the delay equation (9) with a system of $pN+1$ ordinary differential equations, assuming x is scalar. More generally, if we have a system of m delay equations with r delays, the approximation will be a system of $m+rpN$ first order ordinary differential equations.

The VFGEN command `delay2ode` implements this transformation from a delay equation to a system of first order ordinary differential equations. An example of the use of the `delay2ode` command is given in Section 4.3, where it is applied to the Mackey-Glass delay equation.

3.3 `taylor`: Generating Taylor Polynomials

The idea of the Taylor method for approximating the solution to a system of differential equations is to expand the solution to

$$X'(t) = F(X), \quad X \in \mathbb{R}^m \tag{13}$$

in a Taylor series:

$$X(t+h) = X(t) + hX'(t) + \frac{h^2}{2}X''(t) + \frac{h^3}{6}X^{(3)}(t) + \ldots \tag{14}$$

To evaluate this expression, we need the derivatives $X^{(n)}$. The differential equation (13) provides $X'(t)$. To find $X''(t)$, we differentiate the differential equation with respect to t (t arguments are suppressed):

$$X'' = DF(X)[X'] \tag{15}$$

(We denote the r-linear differential $D^r F(X)$ acting on the vectors $V_1, ..., V_r$ as $D^r F(X)[V_1, ..., V_r]$.) Differentiate again to find $X^{(3)}$ and higher derivatives of X:

$$\begin{aligned} X^{(3)} &= D^2F(X)[X', X'] + DF(X)[X''] \\ X^{(4)} &= D^3F(X)[X', X', X'] + 3D^2F(X)[X'', X'] + DF(X)[X^{(3)}] \\ X^{(5)} &= D^4F(X)[X', X', X', X'] + 6D^3F(X)[X'', X', X'] + 3D^2F(X)[X'', X''] \\ &\quad + 4D^2F(X)[X^{(3)}, X'] + DF(X)[X^{(4)}] \end{aligned} \tag{16}$$

and so on.

The `taylor` command generates the C functions necessary to compute the Taylor polynomial for the solution to an ordinary differential equation. For example, the VFGEN command

```
$ vfgen taylor:order=7 pendulum.vf
```

creates C files in which the explicit functions for the r-linear differentials up to $r = 6$ are defined for the pendulum vector field. VFGEN also creates a function that computes all the derivatives of $X(t)$ up to order 7, and finally a function to evaluate the order 7 Taylor polynomial.

4 Case Studies

To demonstrate the use of VFGEN, we present three case studies. The first shows how VFGEN can be used to generate the C code required to solve the pendulum equations (1) with the CVODE library. In the second, the MATLAB program MatCont (actually CL_MatCont, the command-line version of MatCont) is used to do two parameter continuation of bifurcation points of the Morris-Lecar equations. In the last case study, code to solve the Mackey-Glass delay equation with the DDE_SOLVER Fortran function is generated, and solutions generated from finite-dimensional approximations to the equation generated with the `delay2ode` command are compared to the solution generated by DDE_SOLVER.

4.1 Case Study 1: Generating code for the pendulum equations

The equations for the pendulum and the vector field file `pendulum.vf` were given in Section 2. C files to be used with the CVODE library are easily created with the command

```
$ vfgen cvode:demo=yes,func=yes pendulum.vf
```

The options are specified by including a colon after the command, and then listing the options in the form `option=value`. In this example, the options `demo=yes` and `func=yes` have been given.

This command creates several C files. The file `pendulum_cv.c` contains the functions that implement the vector field and its Jacobian, along with a function that computes the value of each **Function** element in the vector field file. In this case, there is just one **Function**, the energy.

Because the option `demo=yes` was given, a file called `pendulum_cvdemo.c` is created. This file contains a complete main program that uses the functions defined in `pendulum_cv.c` and the CVODE library to solve the initial value problem. Initial conditions, parameter values, solver error tolerances, and the time interval can all be specified as command-line arguments to the program. The output consists of lines containing four numbers: t_k, θ_k, v_k and E_k (the energy is included in the output because the option `func=yes` was given).

After compiling the demonstration program (using yet another file created by VFGEN), the demonstration program is run with a command such as

```
$ ./pendulum_cvdemo v=0.0 theta=3.14 b=0.5 stoptime=10 > pend.dat
```

Any initial conditions or parameters not specified on the command line will be given the default values specified in the vector field file. A plot of the data in `pend.dat` is shown in Figure 3.

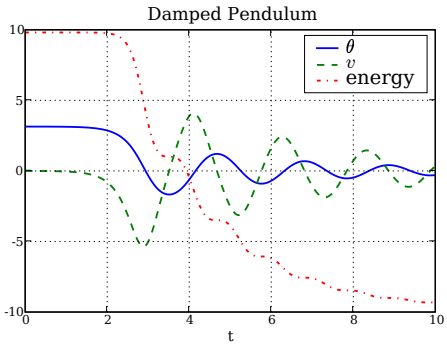

Figure 3: A solution to the pendulum equations (1) generated by the CVODE library using the code generated by VFGEN.

4.2 Case Study 2: Generating MatCont code for the Morris-Lecar equations

The Morris-Lecar equations are a two dimensional model for the voltage oscillations in the barnacle giant muscle fiber[20]:

$$\begin{aligned} C\frac{dV}{dt} &= I - g_{ca}m_\infty(V)(V - V_{ca}) - g_k w(V - V_k) - g_\ell(V - V_\ell) \\ \frac{dw}{dt} &= \phi\frac{w_\infty(V) - w}{\tau_w(V)} \end{aligned} \tag{17}$$

where

$$m_\infty(V) = \frac{1}{2}\left(1 + \tanh\left(\frac{V - V_1}{V_2}\right)\right), \quad w_\infty(V) = \frac{1}{2}\left(1 + \tanh\left(\frac{V - V_3}{V_4}\right)\right), \tag{18}$$

and

$$\tau_w(V) = \cosh\left(\frac{V - V_3}{2V_4}\right)^{-1} \tag{19}$$

Figure 4 shows a vector field file for the Morris-Lecar equations. The symbol `ic` is used instead of `i` or `I` because `i` is predefined to mean $\sqrt{-1}$ in software systems such as MATLAB or Scilab.

MatCont [5] is one of the few software tools that allows the user to provide Hessians and higher order derivatives of the vector field, so VFGEN is especially useful for MatCont users. A MATLAB file to be used with MatCont is created with the command

```
$ vfgen matcont MorrisLecar.vf
```

The MATLAB file is called `MorrisLecar.m`; this file contains the vector field function, the Jacobian, and higher order derivatives. This file is ready to be used with MatCont.

A MATLAB script that uses CL_MatCont to compute a two parameter bifurcation diagram for the Morris-Lecar equations is available on the VFGEN web page. The plot generated by this script is shown in Figure 5.

4.3 Case Study 3: The Mackey-Glass delay equation

The Mackey-Glass delay equation [18] was given in Section 2. Here we use VFGEN to create Fortran code to solve the equation with the DDE_SOLVER software.

The command

```
<?xml version="1.0" ?>
<VectorField Name="MorrisLecar">
<Parameter Name="gca" DefaultValue="5.5" />
<Parameter Name="gk"  DefaultValue="8.0" />
<Parameter Name="gl"  DefaultValue="2.0" />
<Parameter Name="vca" DefaultValue="115.0" />
<Parameter Name="vk"  DefaultValue="-84.0" />
<Parameter Name="vl"  DefaultValue="-55.0" />
<Parameter Name="c"   DefaultValue="20.0" />
<Parameter Name="phi" DefaultValue="0.22" />
<Parameter Name="ic"  DefaultValue="90.0" />
<Parameter Name="v1"  DefaultValue="-1.2" />
<Parameter Name="v2"  DefaultValue="18.0" />
<Parameter Name="v3"  DefaultValue="2.0" />
<Parameter Name="v4"  DefaultValue="30.0" />
<Expression Name="minf" Formula="(1/2)*(1+tanh((v-v1)/v2))" />
<Expression Name="winf" Formula="(1/2)*(1+tanh((v-v3)/v4))" />
<Expression Name="tauw" Formula="1/cosh((v-v3)/(2*v4))" />
<StateVariable Name="v"
                Formula="(1/c)*(ic-gca*minf*(v-vca)-gk*w*(v-vk)-gl*(v-vl))" />
<StateVariable Name="w"
                Formula="phi*(winf-w)/tauw" />
</VectorField>
```

Figure 4: Vector field file `MorrisLecar.vf`, for the Morris-Lecar equations.

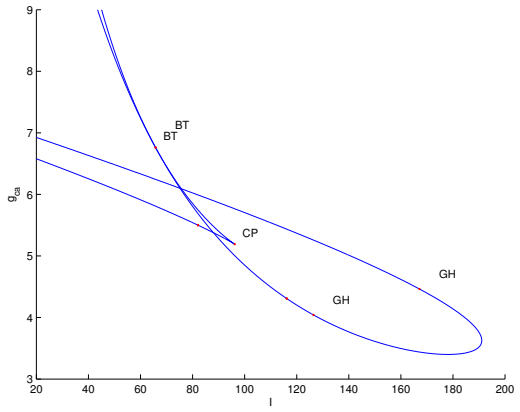

Figure 5: A two parameter bifurcation diagram computed with CL_MatCont, using the MatCont system definition file created by VFGEN. The loop is the curve of Hopf bifurcations; MatCont has found two generalized Hopf points (GH) and a Bogdanov-Takens point (BT) along this curve. The other curve is the curve of limit points; MatCont has identified a cusp (CP) and the same BT point. (The BT point is labeled twice because it was found on both curves.)

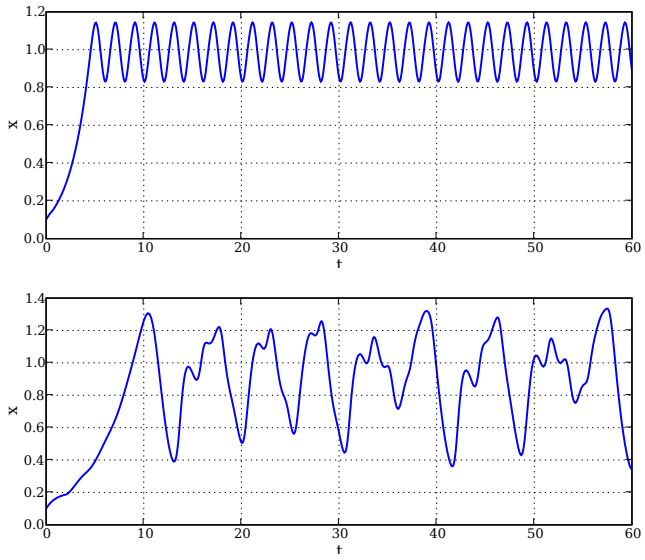

Figure 6: Solutions to the Mackey-Glass equation generated by the DDE_SOLVER software. In both cases, $\alpha = 2$ and $x(t) = 0.1$ for $t \leq 0$. *Top*: $\tau = 0.6$. *Bottom*: $\tau = 2.0$.

```
$ vfgen dde_solver:demo=yes MackeyGlass.vf
```

creates two files, `MackeyGlass.f90` and `MackeyGlass_demo.f90`. The differential equation and history for the variable x are defined in `MackeyGlass.f90`. Because the option `demo=yes` was given, a demonstration program was created in `MackeyGlass_demo.f90`. The demonstration program is compiled and run with the commands

```
$ gfortran -c -w dde_solver_m_unix.f90
$ gfortran MackeyGlass.f90 MackeyGlass_demo.f90 dde_solver_m_unix.o -o MackeyGlass_demo
$ ./MackeyGlass_demo > mg.dat
```

where the file `dde_solver_m_unix.f90` is the Fortran module containing the DDE_SOLVER code.

With the code created by VFGEN, we can easily recreate Figures 2(b) and 2(c) from the article by Mackey and Glass [18]. The solutions generated by DDE_SOLVER with $\tau = 0.6$ and $\tau = 2.0$ are shown in Figure 6. Other than the difference in time scale (recall that (3) is a nondimensional version of the equation), these plots agree with the plots given by Mackey and Glass.

Finally, we present a few plots that compare the solution to the delay equation to solutions to the ODEs created by the `delay2ode` command. A 28 dimensional system of ODEs was created with the command

```
$ vfgen delay2ode:N=9,p=3 MackeyGlass.vf > MackeyGlass_2ode.vf
```

VFGEN was then applied to `MackeyGlass_2ode.vf` to create an initial value problem solver using the CVODE library, and this was used to compute a solution to the same initial value problem as solved by DDE_SOLVER. This was repeated with $N = 16$ and $N = 27$, creating 49 and 82 dimensional systems, respectively. The results shown in Figure 7 demonstrate the convergence of the ODE solutions to the delay equation solution.

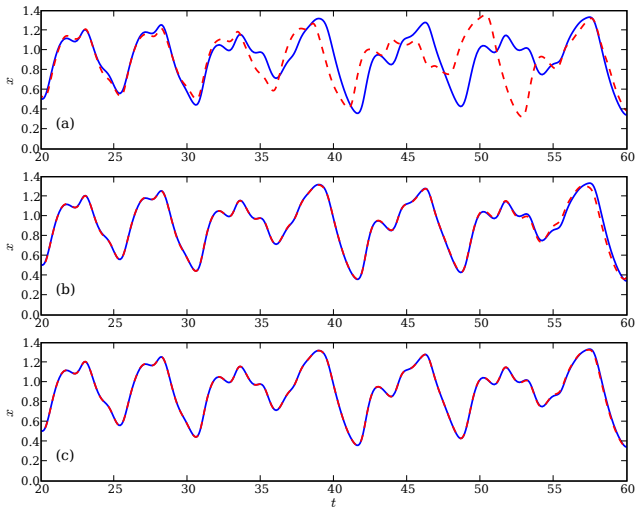

Figure 7: These plots demonstrate the convergence of the solutions to the ODEs generated by the `delay2ode` command to the solution of the delay equation as N is increased. In all three plots, the solid line is the solution to the Mackey-Glass equation obtained by the DDE_SOLVER software. The delay is $\tau = 2$, and the initial conditions are $x(t) = 0.1$ for $t \leq 0$. The dashed line is the solution to the finite dimensional approximation generated by the VFGEN `delay2ode` command, with $p = 3$. The values of N are (a) $N = 9$, (b) $N = 16$, and (c) $N = 27$.

5 Current Limitations and Future Development

A primary goal of VFGEN is to allow a user to easily use an assortment of computational tools. By generating the appropriate system definition files (including analytically derived Jacobians and higher derivatives) for a wide variety of programs, and in some cases providing demonstration programs, the current version of VFGEN has made significant progress towards that goal. But, not surprisingly there is more to be done.

VFGEN cannot handle vector variables, and this is a significant limitation. Support for vector variables is currently under development.

To define a vector field, one must create a vector field file in the XML format. While not difficult, it requires the user to remember syntactic details that are not relevant to the problem to be solved. Currently under development are two forms of user interface, one a stand-alone program and the other a web interface. These will allow the user to define a vector field and generate output without having to know anything about the underlying XML file.

In some cases, the system definition file created by VFGEN does not exploit the full capabilities of the program for which it is generated. For example, DDE-BIFTOOL can handle delay equations with state-dependent delays, but the current version of VFGEN (2.2.0) will only generate DDE-BIFTOOL code for equations with constant delays.

Finally, there are many more computational tools available than those listed in Table 1. Adding more commands to VFGEN and upgrading the existing commands is an ongoing process–VFGEN will never really be "finished".

6 Getting VFGEN

VFGEN can be downloaded from `www.warrenweckesser.net/vfgen`. This web page also provides online documentation of all the commands, and includes many more examples than presented here.

Acknowledgments

This paper was completed while the author was visiting the School of Mathematics and Statistics at the University of Sydney. I am grateful for their hospitality.

The editorial guidance of Larry Shampine is gratefully acknowledged. The suggestions made by Larry and by Skip Thompson greatly improved this paper.

References

[1] A. Back, J. Guckenheimer, M. R. Myers, F. J. Wicklin and P. A. Worfolk, DsTool: Computer assisted exploration of dynamical systems, *Notices Amer. Math. Soc.* 39(4) pp. 303-309 (1992). DSTool web page: `www.cam.cornell.edu/guckenheimer/dstool.html`

[2] C. Bauer, A. Frink and R. Kreckel, Introduction to the GiNaC framework for symbolic computation with the C++ programming language, *J. Symbolic Computation* 33, pp. 1–12 (2002). GiNaC web page: `www.ginac.de`

[3] R. Clewley, D. Lamar and E. Sherwood, PyDSTool software (in development at Cornell University). PyDSTool web pages: `sourceforge.net/projects/pydstool`, `www.cam.cornell.edu/~rclewley/cgi-bin/moin.cgi`

[4] S. D. Cohen and A. C. Hindmarsh, CVODE: A Stiff/Nonstiff ODE Solver in C, *Computers in Physics*, 10(2), pp. 138-143 (1996).

[5] A. Dhooge, W. Govaerts and Yu. A. Kuznetsov, MATCONT: A MATLAB package for numerical bifurcation analysis of ODEs, *ACM Transactions on Mathematical Software*, 29(2), pp. 141-164 (2003). MatCont web page: `www.matcont.ugent.be`

[6] E. J. Doedel, R. C. Paffenroth, A. R. Champneys, T. F. Fairgrieve, Yu. A. Kuznetsov, B. E. Oldeman, B. Sandstede and X. Wang, *AUTO 2000: Continuation and Bifurcation Software for Ordinary Differential Equations (with HomCont)*, reference manual (2006). AUTO web page: `indy.cs.concordia.ca/auto`

[7] J. W. Eaton, *GNU Octave Manual*, Network Theory, Ltd., Bristol, UK (2002). Web page: `www.gnu.org/software/octave`

[8] K. Engelborghs, T. Luzyanina and D. Roose, Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL, *ACM Transactions on Mathematical Software* 28(1), pp. 1-21 (2002). DDE-BIFTOOL web page: `arachne.cs.kuleuven.ac.be/~twr/research/software/delay/ddebiftool.shtml`

[9] K. Engelborghs, T. Luzyanina and G. Samaey, DDE-BIFTOOL v. 2.00: a Matlab package for bifurcation analysis of delay differential equations. Technical Report TW-330, Department of Computer Science, K.U.Leuven, Leuven, Belgium (2001).

[10] B. Ermentrout, *Simulating, Analyzing, and Animating Dynamical Systems: A Guide to XPPAUT for Researchers and Students*, SIAM (2002). XPP web page: `www.math.pitt.edu/~bard/xpp/xpp.html`

[11] A. Gädke, J. Küpper, S. Maret and P. Schnizer, *PyGSL Reference Manual*, `pygsl.sourceforge.net/reference/pygsl/` (2005). PyGSL web page: `pygsl.sourceforge.net`

[12] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth and F. Rossi, *GNU Scientific Library Reference Manual - Revised Second Edition (v1.8)*, Network Theory, Ltd., Bristol, UK (2006). GSL web page: `www.gsl.org`

[13] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther, ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++, *ACM TOMS* 22(2) pp. 131-167 (1996). ADOL-C web page: `www.math.tu-dresen.de/~adol-c`

[14] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems (second edition)*, Springer Series in Computational Mathematics 14, Springer-Verlag (1996). Web page: `www.unige.ch/~hairer/software.html`

[15] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers, *ACM Transactions on Mathematical Software*, 31(3), pp. 363-396 (2005). SUNDIALS/CVODE web page: `www.llnl.gov/CASC/sundials`

[16] E. Jones, T. Oliphant, P. Peter and others, SciPy: open source scientific tools for Python (2001–). SciPy web page: `www.scipy.org`

[17] L. Lamport, LaTeX: A document preparation system: User's guide and reference, 2nd edition, Addison-Wesley Professional, Reading, MA, USA (1994).

[18] M. C. Mackey and L. Glass, Oscillation and chaos in physiological control systems, *Science* 197, 287–289 (1977).

[19] MATLAB®, The Mathworks, Inc., Natick, MA, USA. MATLAB web page: `www.mathworks.com`

[20] C. Morris and H. Lecar, Voltage oscillations in the barnacle giant muscle fiber, *Biophys J.* 1981 July; 35(1):193–213.

[21] Scilab is a trademark of INRIA. Scilab web page: `www.scilab.org`

[22] R. Szalai, *PDDE-CONT: A continuation and bifurcation software for delay-differential equations*, online reference manual (2007). PDDE-CONT web page: `seis.bris.ac.uk/~rs1909/pdde`

[23] S. Thompson and L. F. Shampine, A friendly Fortran DDE solver, preprint (2004). DDE_SOLVER web page: `www.radford.edu/~thompson/ffddes`

[24] W. Weckesser, VFGEN: A Vector Field File Generator, `www.warrenweckesser.net/vfgen`