Homework 10 example

When you use the ODE solvers in MATLAB and then write scripts to automatically produce the plots, you use several pieces of software tied neatly together by a common environment. Because you won't always be programming in MATLAB, it seems worthwhile to point out that you can do the same types of things in a more general UNIX environment: you can solve ODEs using someone else's solvers, write scripts to produce nice plots of the results, and automate all the processes that go into producing a final document. Because everything is automated, it's easy for me to fix a problem in the code (for example) and re-generate the correct output, graphics, and final report just by typing `make`. While this sort of automation is not particularly tied to scientific computing, it makes sense to me to remind you that you can do this type of thing. Think of it as a part of your general computing education.

Using CVODE

CVODE is part of the Sundials package: `https://computation.llnl.gov/casc/sundials/main.html`. It solves both stiff and non-stiff problems using variable-order multistep methods. The code comes with documentation and examples. Using CVODE to solve an ODE is more complicated than using a MATLAB function, but it is not so bad. Except for a set of supporting vector routines, the library basically stands alone, and the documentation is pretty good. In the program below, we solve the pendulum equation using CVODE.

```
/* cvpendulum.c -- pendulum demo
 * dbindel, Apr 2009.
 *
 * Adapted from  demo/cvode/serial/cdenx.c
 * by Scott Cohen, Alan Hindmarsh, and Radu Serban.
 *
 * For more info, see CVODE web page:
 *   https://computation.llnl.gov/casc/sundials/main.html
 */

#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>

#include <cvode/cvode.h>
#include <nvector/nvector_serial.h>
#include <cvode/cvode_dense.h>
#include <sundials/sundials_dense.h>
#include <sundials/sundials_types.h>


static int f(realtype t, N_Vector y, N_Vector ydot, void *f_data)
{
    realtype theta = NV_Ith_S(y,0);
    realtype omega = NV_Ith_S(y,1);
    realtype omegap = -sin(theta);

    NV_Ith_S(ydot,0) = omega;
    NV_Ith_S(ydot,1) = omegap;

    return 0;
}


int main(int argc, char** argv)
{
    int N = 200;
    realtype T0 = 0;
    realtype Tfinal = 10;
    realtype theta0 = atof(argv[1]);
    realtype reltol = 1e-6;
    realtype abstol = 1e-8;
    realtype t;
    int flag, k;
    N_Vector y = NULL;
    void* cvode_mem = NULL;

    /* Create serial vector of length NEQ for I.C. */
    y = N_VNew_Serial(2);
```

```
    NV_Ith_S(y,0) = theta0;
    NV_Ith_S(y,1) = 0;

    /* Set up solver */
    cvode_mem = CVodeCreate(CV_ADAMS, CV_FUNCTIONAL);
    if (cvode_mem == 0) {
        fprintf(stderr, "Error in CVodeMalloc: could not allocate\n");
        return -1;
    }

    /* Call CVodeMalloc to initialize the integrator memory */
    flag = CVodeMalloc(cvode_mem, f, T0, y, CV_SS, reltol, &abstol);
    if (flag < 0) {
        fprintf(stderr, "Error in CVodeMalloc: %d\n", flag);
        return -1;
    }

    /* In loop, call CVode, print results, and test for error. */
    for (k = 1; k < N; ++k) {
        realtype tout = k*Tfinal/N;
        if (CVode(cvode_mem, tout, y, &t, CV_NORMAL) < 0) {
            fprintf(stderr, "Error in CVode: %d\n", flag);
            return -1;
        }
        printf("%g %.16e %.16e\n", t, NV_Ith_S(y,0), NV_Ith_S(y,1));
    }

    N_VDestroy_Serial(y);     /* Free y vector */
    CVodeFree(&cvode_mem);    /* Free integrator memory */

    return(0);
}
```

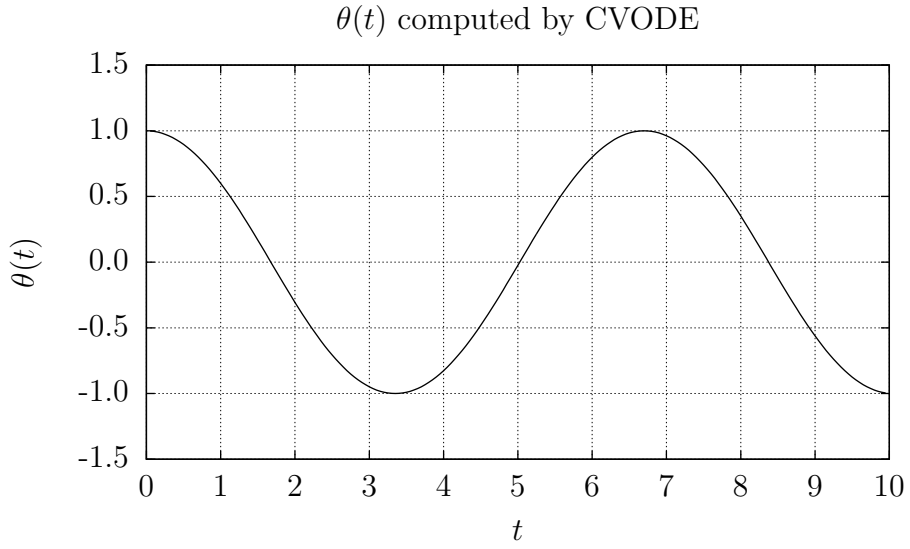

Figure 1: Output of CVODE

Producing the graphics

`gnuplot` is capable of producing some very spiffy-looking graphics. If you have a UNIX-ish system, including Cygwin, you can probably get `gnuplot` for it. In our case, we illustrate the graphics by plotting the computed pendulum angle (Figure 1) and the energy error (Figure 2) for the CVODE solution starting from initial conditions $\theta_0 = 1$.

```
# Gnuplot script to produce plots of pendulum behavior

energy(theta, thetap) = thetap*thetap/2 - cos(theta)
E0 = energy(theta0,0)

# Plot solution
set terminal epslatex size 5in,3in
set output "pendulum_plot.tex"
set grid
set format y "%.1f"
set format x "%.0f"
set title  "$\\theta(t)$ computed by CVODE"
```

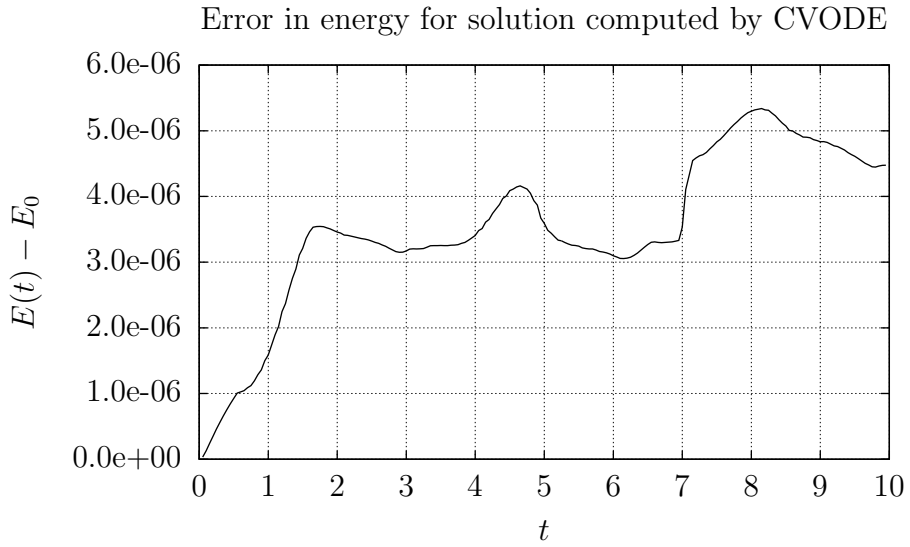

Figure 2: Energy error from CVODE

```
set xlabel "$t$"
set ylabel "$\\theta(t)$"
set nokey
m = 2
plot "pendulum.txt" using 1:2 with lines lw m

# Plot energy error
set output "energy_plot.tex"
set format y "%.1e"
set title  "Error in energy for solution computed by CVODE"
set xlabel "$t$"
set ylabel "$E(t)-E_0$"
plot "pendulum.txt" using ($1):(energy($2,$3)-E0) with lines lw m
```

Automating everything

The UNIX `make` command is a marvelous tool not only for building software, but also for building documents. This `Makefile` describes rules for compiling `cvpendulum.c` (assuming CVODE is installed under `/usr/local`), running it to produce `pendulum.txt`, running a `gnuplot` script to produce plots of the solution and the energy error shown above, and recompiling the whole document.

```
# Makefile for pendulum example

THETA0=1
LIBS=   /usr/local/lib/libsundials_cvode.a \
        /usr/local/lib/libsundials_nvecserial.a

hw10ex.pdf: hw10ex.tex pendulum_plot.pdf energy_plot.pdf

energy_plot.eps pendulum_plot.eps: pendulum.gnuplot pendulum.txt
        echo "theta0=$(THETA0)" | gnuplot "-" pendulum.gnuplot

pendulum.txt: cvpendulum.exe Makefile
        ./cvpendulum.exe $(THETA0) > pendulum.txt

cvpendulum.exe: cvpendulum.c
        gcc -Wall cvpendulum.c -o cvpendulum.exe \
                /usr/local/lib/libsundials_cvode.a \
                /usr/local/lib/libsundials_nvecserial.a

%.pdf: %.eps
        epstopdf $<

%.pdf: %.tex
        pdflatex $<
        pdflatex $<

%.x: %.cc
        g++ $< -o $@
```

```
clean:
        rm -f *.aux *.log *.out
        rm -f cvpendulum.exe
        rm -f energy_plot.*
        rm -f pendulum_plot.*
        rm -f pendulum.txt

realclean: clean
        rm -f hw10ex.pdf
```