9. 780.20 Session 9

a. Follow-ups to Session 8 and earlier

- Secure Shell Hint. When using ssh, the -Y option (note that the “Y” must be uppercase) is usually what you want. For example,
  
  `ssh -Y furnstah@fox.mps.ohio-state.edu`

  Using this option will give permission to all X-windows programs (such as gv or emacs or nedit) to display on the computer you are using. If you don’t use this option, some (but not all) programs will give a cryptic error and not display at all.

- Follow-up to Streams. In the `filename_test.cpp` code, we created stringstream objects and used them to create filenames to open. In doing so, we had an intermediate step where we defined a string `filename2`:

  ```cpp
  // next, create a string with the stringstream class
  ostringstream filename_stream; // declare a stringstream object
  int i = 3;
  // you can load the string stream just like output streams
  filename_stream << "test_stream" << i << ".out";
  // use .str() to convert to a string
  string filename2 = filename_stream.str();

  ofstream file2; // now for a filename
  file2.open (filename2.c_str()); // use .c_str() to convert to a char *
  ```

  We used the intermediate string `filename2` because we wanted to first convert `filename_stream` to a string, and then convert the string to a C-style string (using `.c_str()`). We can skip the intermediate step and do it all at once with `.str().c_str()`:

  ```cpp
  // next, create a string with the stringstream class
  ostringstream filename_stream; // declare a stringstream object
  int i = 3;
  // you can load the string stream just like output streams
  filename_stream << "test_stream" << i << ".out";

  ofstream file2; // now for a filename
  file2.open (filename_stream.str().c_str()); // convert to a char *
  ```

- “Visualization of the Pendulum’s Dynamics.” The handout with this heading shows the progression of phase-space trajectories and Poincaré sections for a pendulum as the value of the driving force amplitude is varied. The idea of a Poincaré section is to plot a point in phase space once every period of the external force, $2\pi/(\text{external frequency})$. The resulting
pattern gives information about the periodicity of the signal (or indicates chaos). The green points in the gnuplot plots from Session 8 form a Poincaré section. You will also find the Mathematica version in the nonlinear.nb notebook in Session 9. The figure caption to Fig. 3.4 in the handout indicates the key characteristics of each plot.

Figures 3.8 and 3.9 show another analysis tool: the power spectrum. The power spectrum is found by taking a Fourier transform (FFT) of the “signal” (e.g., the angle of the pendulum as a function of time). In Fig. 3.8, we see discrete frequencies, while in Fig. 3.9 we see a continuous distribution, which is an indicator of chaos. In the nonlinear.nb and pendulum.nb Mathematica notebooks, you’ll be able to create power spectra. Try to find chaos!

b. Bash Shell

The shell is the command-line user interface to the Linux kernel. It starts up when you log into a machine (or bring up a terminal window) and lets you run programs and interact with the computer hardware. You can find out the shell you logged on into by checking the SHELL environment variable. Type

```
echo $SHELL
```

at the prompt. The result is most likely to be /bin/bash or /bin/tcsh. The first one is the “Bourne Again Shell” or bash (which is a descendent of the Bourne shell written by S.R. Bourne) and the second one is is the TC-Shell or tcsh (which started at Carnegie-Mellon University but was then further developed at Ohio State).

In Session 9, we’ll look at a few useful features of bash. Later we’ll come back and discuss environment variables more so that we can set up our computer to use a different compiler from g++.

c. Using Rsync for Backups

Here is a quote from the rsync web page (at http://samba.anu.edu.au/rsync/):

rsync is a file transfer program for Unix systems. rsync uses the ‘rsync algorithm’ which provides a very fast method for bringing remote files into sync. It does this by sending just the differences in the files across the link, without requiring that both sets of files are present at one of the ends of the link beforehand. Some features of rsync include

- can update whole directory trees and filesystems
- optionally preserves symbolic links, hard links, file ownership, permissions, devices and times
- internal pipelining reduces latency for multiple files
- can use rsh, ssh or direct sockets as the transport
- supports anonymous rsync which is ideal for mirroring
There are many options when invoking rsync. In Session 9, two sets of options useful for backing-up and mirroring files to another computer are given (in the form of aliases in a .bashrc file). A “mirror” means an exact copy on the destination computer of the files on the source computer (this usually means the files in a given directory and its subdirectories). This means that any files on the destination computer that are not on the source computer are deleted. A “back-up” omits this last part, so all of the source files will be reproduced on the destination computer, but there may be additional files.

You will find an explanation of how rsync works (theory and implementation), plus links to the original technical report and Andrew Tridgell’s PhD thesis, at http://samba.anu.edu.au/rsync/how-rsync-works.html

d. Adaptive Differential Equation Solvers

In general, it’s best to adjust the step size $h$ in solving a differential equation because the optimal size will vary for different parts of the function. A routine that varies the step size automatically to keep the local error under control is called “adaptive”. We’ll try out the adaptive routines from GSL in this session with the ode_test.cpp code. This code is based on the example included with the GSL reference manual.

The test program will solve the Van der Pol oscillator, which is defined by the equation

$$\frac{d^2x}{dt^2} + \mu \frac{dx}{dt}(x^2-1) + x = 0,$$  

(9.1)

where $\mu$ is the only parameter. To specify a solution, we give initial values for $x$ and $v = dx/dt$ (which we call $x_0$ and $v_0$, respectively). We’ll take $\mu = 2$ with a variety of initial conditions.

Choosing different initial conditions means starting at different points in a phase space plot ($v$ vs. $x$). In Session 9, you’ll start at three different initial conditions. You should find that the phase-space trajectories all end up on the same curve. (Does this work for any initial conditions?) According to the Landau-Paez discussion, this is called an “isolated attractor” (the phase-space trajectories are attracted to the universal curve).

This is just one example. You are invited to explore further!

e. Interpolation vs. Data Fitting

In this session, we will look at interpolation, which is sometimes confused with data fitting. Our basic interpolation problem will be to take a table of function values $\{x_i, y_i = f(x_i)\}$ for which an analytic form is not available, and estimate $f(x)$ for $x \neq x_i$ (for interpolation, $x$ should be between two of the $x_i$’s, otherwise it is extrapolation, which is much harder). Here are some possible applications for interpolation:

- we want to calculate $\int_a^b [f(x)]^2 \, dx$ using Gaussian quadrature;
• we want the derivative (or 2nd derivative) of the tabulated function \( f \);
• we want to solve an ode involving \( f \) using a GSL routine.

There is an important assumption in all of these applications: the values of \( f \) should not be noisey (although they invariably have round-off errors). An example of a noisy function \( f \) would be experimental data. If you want to interpolate a noisy function, it’s usually best to first fit a curve to the data and then to interpolate on the fit function.

If we assume the values \( y_i \) are not noisy, then between \( x_i \) and \( x_{i+1} \), \( f(x) \) should look like a polynomial, if the points are spaced closely enough. (How close is close enough? See if you can answer this after going through this section.) But what polynomial should we use? The GSL provides several easily interchangable interpolation methods, which you’ll compare for a simple application. The handout “Using GSL Interpolation Functions” takes you through the steps needed to use the GSL interpolation functions.

Lagrange interpolation fits an \((n-1)\)th degree polynomial to \( f(x_i) \) for \( n \) values of \( x_i \). That is, the polynomial is constructed to exactly pass through those \( n \) points; if we call this polynomial \( P_N(x_i) \), then
\[
P_N(x_i) = f(x_i) = y_i, \quad i = 0, 1, \ldots, N.
\] (9.2)
The formula for \( N = 1 \) (linear interpolation) is:
\[
P_1(x) = \frac{x - x_0}{x_1 - x_0} y_1 + \frac{x - x_1}{x_0 - x_1} y_0,
\] (9.3)
while for \( N = 3 \) (a parabolic approximation) we have:
\[
P_3(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0.
\] (9.4)
The general formula is
\[
P_N(x) = \sum_{i=0}^{N} \prod_{k \neq i} \frac{x - x_k}{x_i - x_k} y_i.
\] (9.5)
You might think that applying this to \( n \) points with \( N = n \) would be optimal. It is not! You should only apply polynomial interpolation to a relatively small region (you’ll see why in the Session 9 example!).

A spline function is built from polynomial pieces defined on subintervals of the entire interval for the function. The most commonly used spline is the cubic spline. The idea in this case is to fit \( f(x) \) in the interval \([x_i, x_{i+1}]\) with a cubic polynomial
\[
f_i(x) = f_i + f_i^{(1)}(x - x_i) + \frac{1}{2} f_i^{(2)}(x - x_i)^2 + \frac{1}{6} f_i^{(3)}(x - x_i)^3,
\] (9.6)
with the requirement that the function \( f(x_i) \) is reproduced at all of the \( x_i \) and the first and second derivatives be continuous with the next interval. Thus the function and the first and second derivatives are continuous through the entire interval. We still need boundary conditions for \( f^{(2)} \) at the endpoints. A “natural” spline chooses \( f^{(2)}(a) = f^{(2)}(b) = 0 \). The spline coefficients are determined by a GSL library routine. In the process, we get approximations to the first and second derivatives for free.
f. References


[2] M. Hjorth-Jensen, *Computational Physics*. These are notes from a course offered at the University of Oslo. See the 780.20 webpage for links.