Last revised: 2006-Feb-06

Using GSL Interpolation Functions

This is a guide to using the Gnu Scientific Library (GSL) routines to do interpolation. In the following we'll assume we have a function defined by `npts` pairs of numbers, (x, y), stored in two arrays, `x_array` and `y_array`. We want to evaluate that function for *any* x using interpolation.

1. Include the `gsl_errno.h` and `gsl_spline.h` header files:
   ```
   #include <gsl/gsl_errno.h>
   #include <gsl/gsl_spline.h>
   ```

2. Allocate an interpolation accelerator and a `gsl_spline` object (these could be split into separate lines):
   ```
   gsl_interp_accel *my_accel_ptr
     = gsl_interp_accel_alloc ();
   gsl_spline *my_spline_ptr
     = gsl_spline_alloc (gsl_interp_cspline, npts);
   ```

   Some of the choices for the interpolation function (used as the first argument to `gsl_spline_alloc`) are:

   - `gsl_interp_linear` — Linear interpolation. Probably only useful to check the other interpolation functions.
   - `gsl_interp_polynomial` — Polynomial interpolation. Be careful with this one: only use it with a small number of points.
   - `gsl_interp_cspline` — Cubic spline. This one is highly recommended!
   - `gsl_interp_akima` — Akima spline. Use this to check cubic spline results.

3. Initialize the spline:
   ```
   gsl_spline_init (my_spline_ptr, x_array, y_array, npts);
   ```

4. Evaluate the spline at point `x` to obtain `y`, the first derivative `y_deriv`, and the second derivative `y_deriv2`:
   ```
   y = gsl_spline_eval (my_spline_ptr, x, my_accel_ptr);
   y_deriv = gsl_spline_eval_deriv (my_spline_ptr, x, my_accel_ptr);
   y_deriv2 = gsl_spline_eval_deriv2 (my_spline_ptr, x, my_accel_ptr);
   ```

5. Free the accelerator and `gsl_spline` object:
   ```
   gsl_spline_free (my_spline_ptr);
   gsl_interp_accel_free(my_accel_ptr);
   ```

Here is a simple demonstration program, with the steps from above numbered.

```
#include <iostream>             // cout and cin
#include <cmath>
using namespace std;
#include <gsl/gsl_errno.h>      // Step 1
#include <gsl/gsl_spline.h>     // Step 1

int main (void)
{
  const int NMAX = 300;   // maximum number of array points
  double x_array[NMAX], y_array[NMAX];

  // Step 2a: declare the accelerator and the spline object.
  gsl_interp_accel *accel_ptr;
  gsl_spline *spline_ptr;

  // Interpolate y = sin(x^2) from 0 to 2 with 20 points
  int npts = 20;
  for (int i = 0; i < npts; i++)
    {
      double x_temp = double(i) / 10.;
      x_array[i] = x_temp;
      y_array[i] = sin (x_temp * x_temp);
    }

  // Step 2b: allocate the accelerator and the spline object.
  accel_ptr = gsl_interp_accel_alloc ();
  spline_ptr = gsl_spline_alloc (gsl_interp_cspline, npts);  // cubic spline

  // Step 3: initialize the spline
  gsl_spline_init (spline_ptr, x_array, y_array, npts);

  // Step 4: evaluate the spline and/or derivatives as needed
  double x = 1.414;      // test point
  double y = gsl_spline_eval (spline_ptr, x, accel_ptr);
  double y_deriv = gsl_spline_eval_deriv (spline_ptr, x, accel_ptr);
  double y_deriv2 = gsl_spline_eval_deriv2 (spline_ptr, x, accel_ptr);
  cout << "x=" << x << ", y=" << y << ", y'=" << y_deriv << ", y''="
    << y_deriv2 << endl;

  // Step 5: free the accelerator and spline object
  gsl_spline_free (spline_ptr);
  gsl_interp_accel_free (accel_ptr);

  return (0);     // successful completion
}
```