I Assignment 4: Gaussian Elimination and Integration

due: March 2, 2018 – 11:59 PM

GOAL: After laying the groundwork, it's now time to integrate functions in Assignment #4. We've covered the last necessary topic, `Gaussian Elimination`, to construct arbitrarily accurate Gaussian quadratures. Sec 5 in Lecture Notes online at Github.

This problem set is split into two parts.

In the first part, you will write your own code to perform Gaussian elimination, extended to include row pivoting to help with numerical stability. After a few warmup examples to test your code, you'll use it to find the weights for arbitrary order Gaussian quadrature. This depends on the function `int getLegendreZero(double* zero, double* a, int n)` from the last problem set, if you couldn't complete that problem please let us know! The zeros of the first 64 or so Legendre polynomials can be found on the web, so if need be you can proceed using those in the short run. https://pomax.github.io/bezierinfo/legendre-gauss.html

In the second part, you'll put all of the pieces together and write your own numerical integration code. You'll integrate a few functions we supply on the interval $[-1, 1]$ comparing trapezoidal rule to Gaussian integration with N terms in the sum—this last part depends on the first part of this problem set!

I.1 Coding Exercise #1: Gaussian Elimination.

Gaussian elimination is a general algorithm to systematically solve systems of equations. In the lecture notes, we gave a blow by blow example for 3 unknowns:

$$\begin{aligned} -x + 2y - 5z &= 17 \\ 2x + y + 3z &= 0 \\ 4x - 3y + z &= -10 \end{aligned} \tag{1}$$

This form lends itself nicely to multi-dimensional arrays. The left hand side can be stored as a 3x3 two-dimensional array, and the right hand side can be stored as a one-dimensional array of length 3. These are commonly referred to as A and b, respectively. These operations can all be done in place in A and b (with a temporary variable for interchanging rows when you pivot, and with the assumption that you're not worried about saving the original matrix A). When the algorithm is done, b contains the solution!

Your task for this problem to solve a system of linear equations using Gaussian elimination. The routine needs three inputs:

- A 2D array `A`,
- A 1D array `b`,

- the dimension of the matrix and the array, `dim`.

As noted above, your algorithm should be done in place, destroying the original contents of `A` and `b`. At the end of the algorithm, `b` will contain the solution of the linear system. A benefit of an **in place** algorithm is that you avoid allocating additional memory and carrying around additional references. While this isn't an issue with problems this small, in **Big Data** applications spurious copies of data can be a killer. In this small data example you many want to save one copy of `A` and `b` outside the function to have the original data for future reference and testing. The function you need to program is:

```
int gaussianElimination(double** A, double* b, int dim);
```

where

- `dim` is the *dimension* of the problem, that is, the number of equations and variables.
- `A` is a `dim` by `dim` 2D array which contains the coefficients of a system of equations.
- `b` is a 1D array of length `dim` which contains the right hand side of a system of equations.

For example, for the system of equations in Eq. 1, one might write the code to form `A` and `b`, call the gaussian elimination function, and print the results as:

```
int i;
double** A;
double* b;
int dim = 3;
A = new double*[dim];
for (i=0;i<dim;i++) { A[i] = new double[dim]; }
b = new double[dim];

A[0][0] = -1; A[0][1] =  2; A[0][2] = -5; b[0] = 17;
A[1][0] =  2; A[1][1] =  1; A[1][2] =  3; b[1] = 0;
A[2][0] =  4; A[2][1] = -3; A[2][2] =  1; b[2] = -10;
gaussian_elimination(A,b,dim);
for (i=0;i<dim;i++) { printf("%f ", b[i]); }
delete[] b;
for (i = 0; i < dim; i++) { delete[] A[i]; }
delete[] A;
```

which should print out

```
1.0 4.0 -2.0
```

Of course, this should work for any other system of equations. Try, for example,

$$x + 2y + 3z + 4t = 1 \tag{2}$$
$$4x + 8y + 6z + 7t = 2 \tag{3}$$
$$7x + 8y + 10z + 11t = 3 \tag{4}$$
$$x + y + z + 5t = 4 \tag{5}$$

which will require you to pivot. Put tests for both of these systems in a file `test_gauss_elim.c`.

The deliverables for this exercise are:

- Your own code file `test_gauss_elim.c` as defined above with the function `gaussian_elimination`.

I.2 Applying Elimination to find weights in Gaussian Quadrature

You should refer to our lecture notes on numerical integration for our base discussion of approximating integrals from -1 to 1 in the following form:

$$\int_{-1}^{1} f(x)dx \simeq \sum_{i=1}^{N} w_i f(x_i) \tag{6}$$

We learned that we needed N points to integrate a polynomial of degree x^{2N-1} exactly. More importantly, we discussed how the points x_i, $i = 1, 2, ..., N$ exactly coincided with the zeroes of the Legendre polynomial $P_N(x)$, which we found in Problems Set #2.

As an example, remember, for $N = 1$, $P_1(x) = x$, which has one zero $x_1 = 0$. We can find the corresponding weight w_1 by looking at the constant function, or trivially x^0:

$$\int_{-1}^{1} 1dx = w_1 1 \tag{7}$$
$$2 = w_1 \tag{8}$$

Which tells us we can exactly integrate a polynomial of degree $x^{2N-1} = x^1$ with the form:

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1) = 2f(0) \tag{9}$$

For $N = 2$, $P_2(x) = \frac{3}{2}x^2 - \frac{1}{2}$, which has zeroes $x_1 = -\sqrt{\frac{1}{3}}, x_2 = \sqrt{\frac{1}{3}}$. We can find the corresponding weights w_1, w_2 by looking at the constant function and the linear function:

$$\int_{-1}^{1} 1dx = w_1 1 + w_2 1 \qquad \Longrightarrow 2 = w_1 + w_2 \tag{10}$$
$$\int_{-1}^{1} xdx = w_1 x_1 + w_2 x_2 \qquad \Longrightarrow 0 = -\sqrt{\frac{1}{3}} w_1 + \sqrt{\frac{1}{3}} w_2 \tag{11}$$

This is a system of equations in w_1 and w_2! It can be solved to give $w_1 = w_2 = 1$.

This can be generalized for higher order Gaussian quadrature. In general, for any N, given the zeroes $x_1, x_2, \cdots, x_N$ of $P_N(x)$, we can consider the following system of equations:

$$\int_{-1}^{1} x^0 = w_1 + w_2 + \cdots + w_N \tag{12}$$

$$\int_{-1}^{1} x^1 = x_1 w_1 + x_2 w_2 + \cdots + x_N w_N \tag{13}$$

$$\vdots \tag{14}$$

$$\int_{-1}^{1} x^{N-1} = x_1^{N-1} w_1 + x_2^{N-1} w_2 + \cdots + x_N^{N-1} w_N \tag{15}$$

where

$$\int_{-1}^{1} x^k = \begin{cases} \frac{2}{k+1}, & k \text{ even} \\ 0, & k \text{ odd} \end{cases} \tag{16}$$

You will write a program `guass_quad_weight.c` which, given a value `N` that you ask the user for, prints out the values of the zeroes x_i and the weights w_i. You should reuse the program `getZeros.c` from the previous assignment to find the zeroes, x_i, of the Legendre polynomial $P_N(x)$. You can then set up the system of equations above and use the function `gaussianElimination` that you wrote in the previous part of the problem to compute the weights, w_i.

As an example of how the code may work, let's consider asking for the zeroes and weights for $N = 3$. User input is given on lines beginning with >.

```
> ./gauss_quad_weight
What value of N?
> 3
The zeroes and weights are given by:
x_i                 w_i
-0.774596669241483 +0.555555555555559
+0.000000000000000 +0.888888888888889
+0.774596669241483 +0.555555555555559
```

You can assume we will never ask for `N` larger than 20. You should take it upon yourself to check your answers against Wikipedia:
`https://en.wikipedia.org/wiki/Gaussian_quadrature#Gauss.E2.80.93Legendre_quadrature`.

The deliverables for this exercise are:

- Your own code file `gauss_quad_weight.c` as defined above. Be sure to print the zeroes and weights with at least 15 digits of precision. Your code should make use of the following functions from previous problems:
  - `getLegendreCoeff`
  - `getLegendreZero`
  - `gaussianElimination`

I.3 Coding Exercise #2: Integrate a few functions

This problem set is a little top-heavy: with all of the work you've done, it's now time to put the pieces together and integrate a few functions on the interval $[-1, 1]$. This depends heavily on the code you wrote in the previous problem, so start early and ask questions if you have them! In this problem just write a main program `test_integrate.c` that performs the following three integrals using the Trapezoidal rule and Gaussian integration for $N = 2$ to 15 in the sum.

$$\int_{-1}^{1} x^8 \, dx = ?$$
$$\int_{-1}^{1} \cos(\pi x/2) \, dx = ?$$
$$\int_{-1}^{1} \frac{1}{x^2+1} \, dx = ? \tag{17}$$

You may want to try other functions, but only for your own enjoyment. (A really crazy example that may well fail numerically is $\int_{-1}^{1} \cos(1/x) \, dx$, although the value is -0.168821901119148 according to Mathematica.)

We repeat the formula for the Trapezoidal rule. For N equal segments n the interval -1 and 1 with $N+1$ points, $x_i = -1 + 2i/N$ for $i = 0, 1, \cdots N$. ($x_0 = -1$, $x_N = 1$.) In this case the formula is

$$\int_{-1}^{1} f(x)dx \simeq \sum_{i=0}^{N-1} h \frac{f(x_i) + f(x_{i+1})}{2} \tag{18}$$

where $h = 2/N$.

While you are required to submit the code, the important deliverable is a plot of the relative error between the Gaussian quadrature approximate integral and the "exact" answer given by Mathematica:

- `Integrate[x^8, {x, -1, 1}]]` `N[Integrate[x^8, {x, -1, 1}],15]`
- `Integrate[Cos[Pi x/2], {x, -1, 1}]` `N[Integrate[Cos[PI x/2], {x, -1, 1}],15]`
- `Integrate[1/(x^2 + 1), {x, -1, 1}]` `N[Integrate[1/(x^2 + 1), {x, -1, 1}],15]`

as a function of the number of points N. (Don't confuse this N with the `N` in the Mathematica commands—the latter forces Mathematica to print a numerical solution! You may share these exact result with other students.) Remember, the relative error is defined as (exact − approximate)/exact.

See the Mathematica Notebook on `github` in `Referencecode/n04Integration` that does this for $f(x) = \sin(x)$ on the interval $x \in [0, 1]$. You can play with this BUT in this problem you must submit your own C code that does these integrals.

The deliverables for this exercise are:

- Your own code file `test_integrate.c` as defined above: give the integrals from -1 to 1 of x^8, $\cos(\pi x/2)$, $1/(x^2+1)$ using the Trapezoidal rule and Gaussian quadrature with $N = 2$ to 15. Be sure to print the integrals with 15 digits of precision. Your code should make use of the following functions from previous problems:
  - `getLegendreCoeff`
  - `getLegendreZero`
  - `gaussianElimination`
- Three plots: One for each integral that gives relative errors of integration against the exact answer as a function N for the Trapezoidal and the Gaussian quadrature. Use `gnuplot` to make the figures. Be sure to include labels on the axes. Use the naming convention:
  - `x8err.pdf` for the integral of x^8.
  - `cosPIxerr.pdf` for the integral of $\cos(\pi x/2)$.
  - `x2p1inverr.pdf` for the integral of $1/(x^2+1)$.

Making use of Makefile

Since we are now writing several functions, it maybe time to start to use a Makefile to compile and link separate source files. This is **optional** for this assignment but if you wish to start doing this look at the simple example on github **MakefileExample.zip** to get started. Using a Makefile will be described in class. You will find this a great labor saving device!

I.4 Submitting Your Assignment

This assignment is due at 11:59 pm on Friday , March 2. Please e-mail a **tarball** containing the assignment to the class e-mail, bualghpc@gmail.com. Include your name in the tarball filename.

If you're not familiar with tar, here's a sample instruction that perhaps I would use:

```
tar -cvf evan_weinberg_asgn4.tar [directory with files]
```

You may want to include other files. **Do NOT include a compiled executable!** That's a dangerous, unsafe practice to get into.