Assignment 4: Gaussian Elimination and Integration
due: Feb. 27, 2024– 11:59 PM

GOAL: This is the final implementing Gaussian quadrature. In the last assignment you found the location of the grid point as zeros in Legendre polynomials. Here you determine the weights using a linear algebra routine called `Gaussian Elimination`. Then we will a few integrals to see how (amazingly) well it converges to the correct answer. The last part in **Exercise #4** on GPUs using OpenACC will be done with lots of help in class and the deadline for this may be extended if necessary. This will give you a taste of the amazing speed of GPUs vs CPUs.

I Background

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 warm up 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 Gaussian integration with N terms in the sum to the trapezoid rule from Assignment 3 with $h = 1/(2N)$.

In the **third part** you are going to push to multidimensional integrals and one 4d integral to run in parallel using OpenACC.

I.1 Warm up on 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, `N`.

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 N);
```

where

- `N` is the *dimension* of the problem, that is, the number of equations and variables.
- `A` is a `N` by `N` 2D array which contains the coefficients of a system of equations.
- `b` is a 1D array of length `N` 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 N = 3;
A = new double*[N];
for (i=0;i<N;i++) { A[i] = new double[N]; }
b = new double[N];

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,N);
for (i=0;i<N;i++) { printf("%f ", b[i]); }
delete[] b;
for (i = 0; i < N; i++) { delete[] A[i]; }
delete[] A;
```

which should print out

```
1.0 4.0 -2.0
```

Another fun example to also test `doubles` precision arithmetic. Mathematica can give the **exact** solution to

$$x/16 + 3y + z = 2$$
$$x/5 + y + 2z = 1/4$$
$$5x - 3y + z = 13 \tag{2}$$

as rationals

$$\{x- > 1000/317, y- > 1011/1268, z- > -(747/1268)\} \tag{3}$$

Why is the solution 2 rational. By the way it is not too hard introduce a `rational` class (e.g. data type) on C++ do this. Fun but of course Mathematica is all set with this (see code in **HW4_code** at https://github.com/brower/EC526_2024)

The deliverables for this exercise are:

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

II Coding Exercise #1: Applied to Uniform Grid

Ok suppose we are not as smart as Gauss – **who is!**. (This is a warm up for the Gaussian integral next.) Lets to the integral

$$\int_{-1}^{1} f(x)dx \simeq \sum_{j=1}^{N} w_j f(x_j) \tag{4}$$

on a uniform grid of N-points (or $N-1$ intervals since I am assuming that end points $x_1 = -1$ and $x_N = 1$ are included!) of width $h = 2/(N-1)$

$$x_j = -1, -1+h, -1+2h, \cdots 1-(N-2)h, 1 \tag{5}$$

That is the grid pints are

$$x_j = -1 + 2(j-1)/(N-1) \quad \text{for} \quad j = 1, 2, 3, \cdots N \tag{6}$$

Since the positions of the points are fixed we can only demand exact answers for N terms: $1, x, x^2, \cdots x^{N-1}$. JUST the same algebrix step of full Gaussian integration except you don't have to precompute the x's. are

$$\sum_{j=1}^{N} (x_j)^{i-1} w_j = \int_{-1}^{1} (x_j)^i = (1-(-1)^i)/i) \tag{7}$$

or the equation to solve

$$\sum_{j=1}^{N} A[i][j]w[j] = b[i] \quad \text{where} \quad i = 1, \cdots, N \tag{8}$$

where have defined the matrix and vector

$$A[i][j] = (-1 + 2(j-1)/(N-1))^{i-1} \quad \text{and} \quad b[i] = (1-(-1)^i)/i) \tag{9}$$

SEE mathematica notebook on GitHUB for the exact values of the weights! It turns out the even N is easiery to do beause there is no point at $x = 0$ and Mathematica can't figure out how to compute $x^0 = 1$ for $x = 0$. The general N case neeeds to add this the first row b hand. Then $n = 1$ it is the Reimann itegral rule. For $n = 2$ the trapazoid rule for $n = 3$ simplson rule. But you can keep inventing higher order rules for any N. Pretty good even if you are not as smart as Gauss.

II.1 Coding Exercise #2: Applied to 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:

$$\int_{-1}^{1} f(x)dx \simeq \sum_{i=1}^{N} w_i f(x_i) \tag{10}$$

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 #3.

Here we find the wights w_i for each zero using Gaussian elimination to do Gaussian quadrature. (Yes! Everything seems to have Gauss' name on it! He even invented the FFT.) The weights must obey the conditions,

$$w_1 + w_2 + \cdots + w_N = \int_{-1}^{1} x^0 dx \tag{11}$$

$$x_1 w_1 + x_2 w_2 + \cdots + x_N w_N = \int_{-1}^{1} x^1 dx \tag{12}$$

$$\vdots \tag{13}$$

$$x_1^{N-1} w_1 + x_2^{N-1} w_2 + \cdots + x_N^{N-1} w_N = \int_{-1}^{1} x^{N-1} dx \tag{14}$$

So that the weights $w_1, w_2, \cdots w_N$ are found by solving the N linear equations:

$$\sum_{j=1}^{N} A[i][j] w[j] = b[i] \quad \text{where} \quad i = 1, \cdots N \tag{15}$$

where have defined the matrix and vector

$$A[i][j] = x_j^{i-1} \quad \text{and} \quad b[i] = \int_{-1}^{1} x^{i-1} dx = (1-(-1)^i)/i \tag{16}$$

for $i, j = 1, 2, \cdots N$.

You will write a program `gauss_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 32. 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`

II.2 Coding Exercise #4. Comparing Uniform vs to Gausssian Grid Integrals

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. Modify the integration program to again evaluate

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

but this time use Gaussian integration for $N = 2, 4, 8. \cdots, s32$ and plot error in comparison with the trapezoidal rule. You will want to keep the Trapezoid rule method to be able to make comparisons between Gaussian integration for N point with Trapezoidal rule for $2N$ points.

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 32. 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. . 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 $1/(x^2 + 1)$.

Next extend this to multi-dimensional integrals (The next problem we will do some really hard multiple integral using GPUs and parallelization with OpenACC!)

As a quick refresher, a multiple dimensional integrals

$$\iint_{\Box} g(x,y)dA = \int_{-1}^{1} dy \int_{-1}^{1} dx\ g(x,y) \tag{18}$$

Applying Gaussian integration to each integral is straight forward,

$$\int_{-1}^{1} dy \int_{-1}^{1} dx\ g(x,y) \approx \int_{-1}^{1} dy \sum_{i=1}^{N} w_i g(x_i, y) \tag{19}$$

$$\approx \sum_{j=1}^{N}\sum_{i=1}^{N} w_i w_j g(x_i, y_j) \tag{20}$$

This is just a nested loop for a 1d integrals over the x-coordinate and the y-coordinated. Note that the weights w_i and the value of x_i and y_j are exactly the same as in the one dimensional example. This approximation generalizes trivially and dimension on a "hyper-cubic" cell.

II.3 Integrating a few multiple double Integrals

This problem should seem familiar: we're going to integrate a few problems on the interval $[-1, 1]$, except in multiple dimensions. Write a main program `test_integrate_2d.c` that performs the following three integrals using Gaussian integration for $N = 2$ to 24 in the sum:

$$\int_{-1}^{1} dy \int_{-1}^{1} dx\, [x^8 + y^8 + (y-1)^3(x-3)^5] = ? \tag{21}$$

$$\int_{-1}^{1} dy \int_{-1}^{1} dx\, \sqrt{x^2 - y^2 + 2} = ? \tag{22}$$

$$\int_{-1}^{1} dy \int_{-1}^{1} dx\, e^{-x^2 - \frac{y^2}{8}} \cos(\pi x) \sin(\frac{\pi}{8} y) = ? \tag{23}$$

You can test you code against Mathematica. (Don't be shy about getting help with Mathematica!)

III Exercise #5 Using OpenACC with Gaussian integration

Next, let's try a 4 dimensional integral:

$$P(m) = \int_{-1}^{1} dp_0 \int_{-1}^{1} dp_1 \int_{-1}^{1} dp_2 \int_{-1}^{1} dp_3 \frac{1}{\left(\sin^2(\pi p_0/2) + \sin^2(\pi p_1/2) + \sin^2(\pi p_2/2) + \sin^2(\pi p_3/2) + m^2\right)^2} \tag{24}$$

What is this integral anyway? Glad you asked, although there is no need to know this. It is called a one loop Feynman diagram on a 4d hyper-cubic lattice. It is related to the probability in quantum mechanics of two particles (of mass m), starting at the same point, performing a random walk in space and time, and ending up eventually together at the same place at the same time! To get this random walk into this cute integral, we first put all of the space-time on an infinite 4D lattice and let the particle hop from point to point. Using the magic of Fourier transform we convert to velocities $v_i = p_i/m$ and energy $E = p_0$. Lots of applications in engineering materials and physics need to do such integrals. Each time you add one particle you get another 4 dimensional integral. These are real computational challenges which is a very active research area.

As a first step, integrate this for $m = 1$. Now if you want an accurate result from Gaussian integration there are a lot of sums, e.g. $N = 16$ implies $16^4 = 65536$ terms. You'll want to parallelize this integral. Performing an integral is an example of a *reduction*, where we are summing a series of numbers. Reductions can be finicky in parallel code due to race conditions—in a naïve implementation, multiple threads can try to accumulate a sum variable at the same "time". Thankfully, OpenACC can handle this. See the instruction on GitHub for running OpenACC at the SCC with a Makefile that you can copy and modify for this program.

For a four dimensional integral, you should have four nested `for` loops. Include an appropriate `pragma` to parallelize the outer loop. Try this for a range of m: for large m, the probability of them ever meeting is small, while for lighter particles, there is a much higher chance they'll meet... and the integral becomes more difficult. Scan over m from $\frac{1}{\sqrt{10}}$ to 10 in steps of $10^{-1/6}$ (so 10 different values). You'll see some inconsistency as a function of N for very small m—what's the reason for this? Don't be worried about it, ultimately. While it's not perfect, you should see a scaling $P(m) \sim -\log(m)$ at small m: make a plot showing this. Remember to set your axes properly to make it clear!

Write a main program `test_integrate_feynman.cpp` where you perform this integral in parallel, and create a plot `integral_scaling` (or whatever appropriate extension your `gnuplot` or `Mathematica` supports) showing the log behavior described above. The program should be compiles by a makefile called `makeACC` which is run with the command `make -k -f makeACC` and placed in your submission file HW4 on the SCC with the `integral_scaling`

By the way it is easy to see what the answer is as we take $m^2 \to \infty$. The leading term is

$$P(m) \simeq 16/m^4 \ .$$

Why?

Setting

$$x = (\sin^2(\pi p_0/2) + \sin^2(\pi p_1/2) + \sin^2(\pi p_2/2) + \sin^2(\pi p_3/2))/m^2 \ ,$$

you can expand the denominator as

$$(1/m^4)/(1+x)^2 = (1/m^4)(1+x)^{-2} \simeq (1/m^4)(1 - 2x + 3x^2 - 4x^3 + 5x^4 \cdots) \ .$$

The next term is a easy integral over sin squares so you can write this down as well. You will find that at large m Gaussian Quadratures is a lot better than near $m^2 = 0$. Why?