Assignment 7: Multigrid and FFT due: April 2, 2019 – 11:59 PM

GOAL: FFT (fast Fourier transform) and MG (multigrid) are recursive methods that are the bedrock of fast algorithms in both computer science tasks and numerical methods for high performance computing. They work magically but to appreciate them you just need to code them. **Their magic is easier to see in practice than to explain: Learn by doing!**

1 Review Background

Let's start again with the simple iterative solvers Jacobi, Gauss Seidel and Red/Black (or even/odd) iterations in 1D. We want to combare them with multigrid. Remember all these methods must give the same solution when they converge, so you can use one method to debug the next. Also in 1D the **exact** solution is known even on a grid with spacing h, so there is no confusion of what the answer is. **(Using a simple exact solution to debug a code is perhaps the most important debugging tool used by expert computational engineers and scientists but too often not taught in class rooms.)** The 1D Laplace equation for a function $\phi(x)$ of a real variable $x \in [a, b]$ and put it on a grid with spacing h is:

$$-\frac{d^2\phi(x)}{dx^2} = b(x) \rightarrow -\frac{2\phi[i] - \phi[i-1] - \phi[i+1]}{h^2} = b[i] \tag{1}$$

Let's take this to model temperature in the cold night with a heater in the midle of the 1D room. The walls are set to temperatures T_1 and T_2. For simplicity let's take $a = -1, b = 1$ with a grid from $i = -N, \cdots N$ with $h = 1/N$. Let's set the walls to absolute zero $T_1 = 0$ and $T_2 = 0$ with a single heater in the middle of the room! (I guess you are in outer space in a linear room. The matrix `A T = b` equation is

$$T[i] - \frac{1}{2}(T[i-1] + T[i+1]]) = h(\alpha/2)\delta_{x,0} \tag{2}$$

with $T[N] = T[-N] = 0$. Actually we know the solution!

$$T[i] = (N - |i|)(h\alpha/2) \quad \text{for} \quad x = ih \tag{3}$$

Since this is exact, we can even look at for $h \rightarrow 0$.

$$-\frac{d^2T(x)}{dx^2} = \alpha\delta(x) \tag{4}$$

whatever "$\delta(x)$" means. The rule is the integral over $\delta(x)$ is 1. Let's use this as the definition. (Physicists and engineers cheat this way all the time. Actually it is not cheating it is smart. [1])

Now with $h = 1/N$ our solution is $T = h(N - |i|)(\alpha/2) \rightarrow (1 - |x|)(\alpha/2)$ is independent of h so it is also the continuum solution when we take $h \rightarrow 0$ or $N \rightarrow \infty$. It has zero second derivative for all

[1] See comment in in the very interesting book *The history of Pi* by Petr Beckman that "What especially outranged the mathematicians was not so much that electrical engineers continued to use it (e.g. delta functions) but that it would almost always supply the correct result". This book is fun, short and really worth reading: https://www.amazon.com/History-Pi-Petr-Beckmann/dp/0312381859

$x \neq 0$. What happens at $x = 0$? Lets check the solution by comparing the LHS and RHS of Eq. 4 near $x = 0$,

$$\begin{aligned} \text{LHS} &= -\int_{-\epsilon}^{\epsilon} \frac{d^2T(x)}{dx^2} = -\frac{dT(x)}{dx}|_{-\epsilon}^{\epsilon} = (1+1)\alpha/2 \\ \text{RHS} &= \int_{-\epsilon}^{\epsilon} \alpha\delta(x) = \alpha \end{aligned} \tag{5}$$

for $\epsilon > 0$.

1.1 Background on Coding Problem

This is the real part—it's time to turn the simple program you wrote in the previous problem set into a Multigrid program. The project will extend this to more interesting application in 2D of your choice. (As a start see the 2D code `mg.c` on github.)

For a first Multigrid exercise it is annoying to have boundary conditions The problem we started with has fixed boundary conditions so the value of $T[i]$ at $i = 0$ and $i = 2N$ are not variables. So there are really are $2N - 1$ free variables. We would like that to have 2^n variables to do the Multigrid or FFT so let's be creative but with $2N - 1 = 2^n$ this is impossible.

There are lots of methods to deal with boundary condition BUT to make life easy (always a good idea at first) let's turn this into a periodic problem. This is easy. Just double the size of the problem and make it odd with reflection around the boundaries.

To do this take double the orginal $2N + 1$ points to $N_0 = 4N + 2 - 2 = 4N$ points and make the system anti-periodic. (We have - 2 because when you joint the two parts the $T = 0$ ends are identified.) Namely put a positive source at $i = N$ again and a negative source at $i = -N$ which is equivalent mod N_0 to a negative source at $i = 3N$ (i.e. $-NmodN_0 = 3N$). Now the solution automatically will vanish at $i = 0$ and $i = 2N$ by symmetry so we can solve this problem with multigrid and FFTs on a periodic grid of size $N_0 = 4N$.

2 Coding Exercise #1: Recursive Multigrid Solver

Ok with this method have periodic problem to compare Multigrid and the FFT solution, let's simplify the notation.

Let N be (as usual) the total numer of grid points and now place the positive charges at $(N/4)$ and the negative charge at $3(N/4)$. So the 1D periodic equation is

$$T[i] - \frac{1}{2}(T[(i-1+N_0)\%N] + T[(i+1)\%N]) + h^2m^2T[i] = h\delta_{i,N/4} - h\delta_{i,3(N/4)} \tag{6}$$

for $i = 0, \cdots, N-1$ and we assume powers of 2 ($N = 2^n$). m^2 is a small number that will not change the solution very much. In your code you can set $h = 1$. If the Multigrid is really working you can take it almost to zero.

The problem is to program the matrix problem in Eq. 6 this with multigrid and compare the iteration number for large N and scaling with respect to N relative to the single level algorithms above. For simplicity, you can just use Jacobi iterations. The code is recursive and needs 3 basic routines as described above and in class. The top level has $h = 1$ and $N = 2^n$ grid points. This is called level 0. Below it are levels with spacing $2^{level}h$ and $N/2^{level}$ grid points. There should be at least 3 functions, something like the following:

```
Proj(double *rHat, double r, int level);          // Project level to level +1
Inter(double *error,double *errorHat,int level);  // Interpolate level to level -1
Iterate(double *phi_new, *phi, *b , level);       // Iterate Once  on level
```

Now for the problem you should write a MG code for both 1D and 2D. In 2D the grid will have $L \times L = N$ points. **To simplify the exercise in 2D put just one point source in the middle at periodic boundary conditions in both x and y directions.** So you are solving the matrix problem:

$$\begin{aligned} T[i][j] &- \frac{1}{4}(T[(i-1+L)\%L]][j] + T[(i+1)\%L]][j] + T[i][(j-1+L)\%L]] + T[i][(j+1)\%L]) \\ &+ h^2 m^2 T[i]][j] = h^2 \delta_{i,L/2}\delta_{j,L/2} \end{aligned} \tag{7}$$

The program should stop iterating when each method has reached single precision convergence conditions:

$$\frac{\sqrt{\sum_{i=1}^{2N-1} r[i]*r[i]}}{\sqrt{\sum_{i=1}^{2N-1} b[i]*b[i]}} < 10^{-6} \quad \text{or} \quad \frac{1}{N}\sqrt{\sum_{i=1}^{2N-1} r[i]*r[i]} < 10^{-6} \tag{8}$$

Can use either. Note that for our point sources their strength is proportional to $h \sim 1/N$ so both stopping conditions scale the same way for large N.

The deliverables for this exercise are:

- At least one source file, `multigrid.cpp`, as described above. Don't forget a Makefile!
- Plots that shows the number of iterations in both your 1D and 2D code for as a function of m^2 for $m^2 = 1, 0.1, 0.01, 0.001$and 0.0001 for Jacobi iterations both with and without multigrid to a small residual like 10^{-6}. The number of grid points should be large so the linear dimension is at least $L = 1024$. Vary this until you get some nice plots for one large valus of L.
- For the smallest mass for both Jacobi and multigrid plot the residual as function of iteration and comment.

3 Coding Exercise #2:Solution by FFT

Solve this problem using an FFT and compare with the previous part. How do we do this? Ok it is convenient now to call the index x as an integer and set h = 1. This is common trick in code. Later

you can put back h with $x \to xh$, when you want to revert to the original problem.

First we re-write Eq. 6 in k-space. Fourrier transform is

$$\widetilde{T}[k] = \frac{1}{N}\sum_{x=0}^{N-1} e^{2\pi ikx/N} T[x] \tag{9}$$

If you substitute this into Eq. **??** you get,

$$(1-\cos(2\pi k/N))\widetilde{T}[k] + m^2\widetilde{T}[k] = \frac{1}{N}\sum_{x=0}^{N-1} e^{2\pi ikx/N}[h\delta_{x,N/4} - h\delta_{x,3(N/4)}] = 2i\sin(2\pi k/4) \tag{10}$$

This can be explicitly solve for $\widetilde{T}[k]]$ and the solution given by the reverse transformation,

$$T[x] = \sum_{k=0}^{N-1} e^{-2\pi ikx/N}\widetilde{T}[k] \tag{11}$$

No need iterate the matrix equation at all!

Similarly the 2D fourrier transform,

$$\widetilde{T}[k_x][k_y] = \frac{1}{L}\sum_{y=0}^{L-1} e^{2\pi ik_y y/L}\frac{1}{L}\sum_{x=0}^{L-1} e^{2\pi ik_x x/L} T[x][y] \tag{12}$$

of Eq. 7 gives,

$$(\ 1-\cos(2\pi k_x/L) - \cos(2\pi k_y/L)\ )\widetilde{T}[k_x][k_y] + m^2\widetilde{T}[k_x][k_y] = e^{i\pi}e^{i\pi} = 1 \tag{13}$$

In the lecture notes and on GitHub, we have a test code for the regular (slow) FT in detail mostly to show how to use complex variables in C. Here you only need to turn the FT into a FFT by introducing a recursive call to a function. For debugging you will want to do the inverse too. So add to the test code the functions

```
void FFT(Complex * Ftilde, Complex * F, Complex * omega, int N);
void FFTinv(Complex * F, Complex * Ftilde, Complex * omega, int N);
```

then apply them as set routine to this exercise.

Next generalize this problem for the 2D example in the multigrid examples with periodic boundary condition in both axes. This is not difficult because you can take Fourier transform on one axis and another one after the other.

The deliverables for this exercise are:

- At least one source file, `multigrid.cpp`, as described above. Don't forget a Makefile!
- For both 1D and 2D make a 2d plot of the solution for both the multigrid solution and the FFT solution picking a convenient (not too large grid size). The FFT should be "exact" up to round off. Take the difference of multigrid and FFT so see if the agree to round off.

COMMENT: Why did I do FFT and MG together. Because the transfer of data for N sites to N/2 is exact the same for both. You can write them once and use them for both FFT and MG. For example in MG: Input double T[N] for $x = 0, 1, 2...N-1$ with $N = 2^p$

$$x = n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^{p-1} n_{p-1} \tag{14}$$

where $p = \log_2(N)$ is the number of bits. (call it "x" or "n" who cares!)

How do we do bit divide an conquer? Project on to N/2 values:

$$\hat{x} = x/2 \quad , \quad \hat{x} = n_1 + 2n_2 + \cdots + 2^{p-1} n_{p-1} \tag{15}$$

Interpolate back to N values:

$$\text{even:} \quad x = 2\hat{x} \quad , \quad \text{odd:} \quad x = 2\hat{x} + 1 \tag{16}$$

So for example in MG the *Projection* routines above you project using

$$\hat{r}[\hat{x}] = (1/2)(r[2\hat{x}] + r[2\hat{x} + 1]) \tag{17}$$

and *Interpolate* routine uses,

$$e[2\hat{x}] = e[\hat{x}] \quad , \quad e[2\hat{x} + 1] = e[\hat{x}] \tag{18}$$

Same algebra here for FFT needed for

$$y_k = \mathcal{FT}_{N/2}[a_{2n} + \omega_N^k a_{2n+1}] \tag{19}$$

$$y_{k+N/2} = \mathcal{FT}_{N/2}[a_{2n} - \omega_N^k a_{2n+1}] \tag{20}$$

The basic difference is the FFT keep two copies for even/odd as it recurses so it losses no information and in one pass gets the exact tranform in $O(NlogN)$ time. The recursive discrete FFT is very much like multigrid. In fact for this case at fixed h, it is **exact** (with infinite precession arithmetic which is impossible of course), but it is not as efficient to a reasonable accuracy. More important the FFT, it can not be generalized to complex geometries and variable conductance. MG converges in $O(N)$ to fixed accuracy. Faster, more stable and more generally applicable.

4 Submitting Your Assignment

There kernel codes for MG and FFT on GitHub at `HW_Code_Samples/HW7`. A lot of the final code can be refactored from these sample. The main exercise it changing dimension for d =1 to d = 2 and changing boundary contions.

The solutions should be put in your CCS account at HW7 subdirectory.

This assignment is due at 11:59 pm on Friday, April 2, 2019.

Hard Copy write up: Also pass in a more detailed derivation of Eq. 10 and Eq. 13 filling in details to make it clear how these follow from the Fourier transform and why the inverse Fourier tranform solvers these 1D and 2D matrix problems.

4.1 Extra Credit and/or Beginning of a Project

There are many linear solvers – indeed a vast landscape. Some of these might mak ideal examples as an element of your project. One example that you can try as a quick extra credit in this Homework is the venerable *Conjugate Gradient Method*. This is a class of method s that use so called Krylov space. Look at Conjugate Gradient reference below. It easy to program and as extra credit compare its performance with red/black and Multigrid for the 1D Laplacian. If you want an extensive (but readable) set of notes on Conjugate Gradient see on GitHup the file: *painless-conjugate-gradient.pdf*.

One project if you are so inclined is to exercising solvers on a simple heat map or heat flow and scale them up at the SCC with MPI and/or use openACC to put them on a GPU. Keep it simple. Pick a particular geometry, boundary conditions, heat production and heat sink and a simple solver choice and parallelization method. You could search the web for input files from actual chips or other 2D surfaces.

Some Wikipedia References

1. Wikipedia, Jacobi method
2. Wikipedia, Gauss-Seidel method
3. Wikipedia,Multigrid method
4. Wikipedia,Conjugate Gradient
5. Wikipedia, Thermal management (electronics)