Assignment 9: Multigrid versus FFT

due: April 23, 2020 – 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!** Amazingly MG is both much more general and sometime out perform the FFT as solver.

NOTE: The coding exercise requires basically modifying code provided and re-cycling programs form HW8. The basic MG code `mg.cpp` for 2D is already provided on GitHub in `HW9code` It solves the 2D problem with periodic boundary condition with point source and a mass term to avoid the singular constant solution. See on line Lecture notes and attend zoom class for backgroud on MG. For the FFT comparison use the basic programs from HW8. So the exercise is to understand and modify these elements. (It is even ok to use source code of an FFT from the web if you want for this HW9.)

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 experts 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 $h \to 0$.

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

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

2 Coding Exercise #1: Recursive Multigrid Solver

For your Multigrid exercise it is annoying to have fixed boundary conditions. So we use a trick to turm it into two copies with periodic boundaries! 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 $2N + 1$ 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 $N_0 = 4N$. The periodic problem has, for $doubleT[N_0]$, wrap around conditions $T[i + 1] = T[(i + 1)\%N_0]$ and $T[i - 1] = T[(i - 1 + N_0)\%N_0]$

$$T[i] - \frac{1}{2}[T[i-1] + T[i+1]] + h^2\frac{m^2}{2}T[i] = \alpha h^2\delta_{i,N} - \alpha h^2\delta_{i,3N+2} \tag{6}$$

for $i = 0, \cdots, N_0 - 1$ and we assume powers of 2 ($N_0 = 2^n$). 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.

[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"

The problem is to program 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 $N_0 = 2^n$ grid points. This is called level 0. Below it are levels with spacing $2^{level}h$ and $N_0/2^{level}$ 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 you may simplify the exercise still further with just one point source in the midle and periodic boundary conditions in both x and y directions. (For further reference possibly for you project see Sec 10.2 in class notes for fixed boundary conditions.) The program should stop iterating when each method has reached single precision convergence:

$$\frac{\sqrt{\sum_{i=1}^{2N-1} r[i] * r[i]}}{\sqrt{\sum_{i=1}^{2N-1} b[i] * b[i]}} < 10^{-6} \tag{7}$$

The deliverables for this exercise are:

- For simplicity provide two source file, `multigrid1D.cpp` and `multigrid2D.cpp`. 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 1024. Vary this until you get some nice plots.
- 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. Ok let's pretend we don't know the solution (this will be true when we go to 2D next). We could try a series in $cos(xn\pi/(2N))$ which satisfy the boundary condition:

$$T[x] = \sum_{k=1}^{N} a_k cos(xk\pi/(2N)) = a_1 \cos(x\pi/(2N)) + a_2 \cos(2x\pi/(2N)) + \cdots \tag{8}$$

An interesting feature of the solution is that the Fourier modes don't like this discontinuous derivative. See Gibbs phenomena and discussion in class: https://en.wikipedia.org/wiki/Gibbs_phenomenon Still the slow modes are the low k- terms as explained in the Lecture.

First we re-write our equation in k-space:

$$(1-\cos(2\pi k/N_0))\widetilde{T}[k] = \alpha h^2[e^{i2\pi kN/N_0} - e^{-i2\pi kN/N_0}] = 2i\alpha h^2 \sin(2\pi kN/N_0) \tag{9}$$

for for the transformed solution: $\widetilde{T}[k] = \sum_x e^{i2\pi kx/N_0} T[x])$ Then we solve for $T[k]$ and calculate the Fourrier transform.

$$T[x] = \frac{1}{N_0}\sum_{k\neq 0} e^{-i2\pi kx/N_0}\widetilde{T}[k] \tag{10}$$

(Why can I drop $k = 0$ although I have no m^2 term? Food for thought?) Do this with a regular "slow" FT and a super FFT. Compare multigrid vs FTT speeds for a range of a values of N_0.

In the lecture notes, we write 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:

- Using the codes from HW8 it is easy to add the FFT solution to the source files.`multigrid1D.cpp` and `multigrid2D.cpp`.
- 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 $N = 2^p$

$$x = n_0 + 2n_1 + 2^2 n_2 + \cdots + 2^{p-1} n_{p-1} \tag{11}$$

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:

$$\hat{x} = x/2 \quad , \quad \hat{x} = n_1 + 2n_2 + \cdots + 2^{p-1}n_{p-1} \tag{12}$$

Interpolate back to N values:

$$\text{even:} \quad x = 2\hat{x} \quad , \quad \text{odd:} \quad x = 2\hat{x} + 1 \tag{13}$$

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

$$\hat{r}[\hat{x}] = (1/2)(r[2\hat{x}] + r[2\hat{x} + 1]) \tag{14}$$

and *Interpolate* routine uses,

$$e[2\hat{x}] = e[\hat{x}] \quad , \quad e[2\hat{x} + 1] = e[\hat{x}] \tag{15}$$

Same algebra here for FFT needed for

$$y_k = \mathcal{FT}_{N/2}[a_{2n} + \omega_N^k a_{2n+1}] \tag{16}$$

$$y_{k+N/2} = \mathcal{FT}_{N/2}[a_{2n} - \omega_N^k a_{2n+1}] \tag{17}$$

or

$$y_{2k} = \mathcal{FT}_{N/2}[a_n + a_{n+N/2}] \tag{18}$$

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

They are both correct! Some code use one in the transform and other in the inverse to safe network traffic!

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.

3.1 Extra Credit

If you are bored take a look at Conjugate Gradient (CG) solvers and compare the CG performance with red/black and Multigrid for the 1D Laplacian above. Of course you may not time CG but it is easy to find programs and it open up a vast area for Solvers and Optimizers in common practice. You just need to apply the matrix operator for any `A x=b` system and be able to take scalar product of the vectors.

4 Comment of Variety of Linear Solvers

There are many linear solvers – indeed a vast landscape. One example venerable and elegant one is the `Conjugate Gradient Method` . If your project happens to have a linear solver or gradient descent you

might try it. It is easy to find code and explanation in the Numerical Recipes text. This is a class of methods that use so called Krylov space. Look at Conjugate Gradient reference below. If you want an extensive (but readable) set of notes on Conjugate Gradient see `painless-conjugate-gradient.pdf` in the file `LinearSolverReference` on GitHUB.

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)