CAAM520 Computational Science

Homework 1

AO CAI
S01255664
Earth Science Department
Rice University
Jan 30, 2019

Contents

1 Introduction

The Goal is to solve the matrix system $Au = b$ resulting from an $(N+2) \times (N+2)$ 2D finite difference method for Laplace's equation $-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x, y)$ on $[-1, 1]^2$. At each point (x_i, y_i), derivatives are approximated by:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

where $h = 2/(N+1)$. Assuming zero boundary conditions $u(x, y) = 0$ reduce this to an $N \times N$ system for the interior nodes.

2 Forward modeling

The first problem to be solved is apply the Finite difference (FD) matrix A to the wavefield u. In order to save the computational storage, I would like to directly compute the response of Au instead of computing and storing the A matrix. For the grids that are not near the boundaries. The FD operator is:

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \approx \frac{4u_{i,j} - u_{i,j+1} - u_{i+1,j} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

For the grids on the different boundaries, the operator changes correspondingly:

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \approx$$

left boundary

$$\frac{4u_{i,j} - u_{i,j+1} - u_{i+1,j} - u_{i,j-1}}{h^2}$$

right boundary

$$\frac{4u_{i,j} - u_{i,j+1} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

top boundary

$$\frac{4u_{i,j} - u_{i,j+1} - u_{i+1,j} - u_{i-1,j}}{h^2}$$

bottom boundary

$$\frac{4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j-1}}{h^2}$$

Therefore, the Implementation of the forward modeling can be written as:

$FD = 4 \times u_{i,j}$
If $i > 1, FD = FD - u_{i-1,j}$
If $i < N, FD = FD - u_{i+1,j}$

If $j > 1, FD = FD - u_{i,j-1}$
If $j < N, FD = FD - u_{i,j+1}$

The implementation of the operator is written in the function computeAu.

3 Weighted Jacobi method

The weighted Jacobi method use the following algorithm:

$$x^{(k+1)} = \omega D^{-1}(b - Rx^{(k)}) + (1 - \omega)x^{(k)}$$

To verify the code, for a small $N = 2$ (4 interior nodes) grid, we have four variables u_1,u_2,u_3,u_4, where the linear equations is $h = 2/3$:

$$4 \times u_1 - u_2 - u_3 = \frac{-\sqrt{3}}{2} \times \frac{-\sqrt{3}}{2} \times \frac{4}{9} = \frac{1}{3}$$

$$4 \times u_2 - u_1 - u_4 = -\frac{1}{3}$$

$$4 \times u_3 - u_1 - u_4 = -\frac{1}{3}$$

$$4 \times u_4 - u_2 - u_3 = \frac{1}{3}$$

The solution is: $u_1 = u_4 = 0.05556, u_2 = u_3 = -0.05556$

```
{556}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_Jacob
The objective at iter 0 is: 0.66666668653
The model objective at iter 0 is: 0.07599088550
The final objective at iter 10 is: 0.00000063578
The final model objective at iter 10 is: 0.03512011841
The numerical solution u1=0.055556 u2=-0.055556 u3=-0.055556 u4=0.055556
The b matrix b1=0.333333 b2=-0.333333 b3=-0.333333 b4=0.333333
The Total computing time is: 0.000000(s)
```

Figure 1: Verify Jacobi method on N=2 situation

Figure 1 shows the numerical results computing using the executable file HW1Jacob with $N = 2$, which verifies the code. For the other numerical methods, I also verified the code with $N = 2$ but will not listed in the following sections.

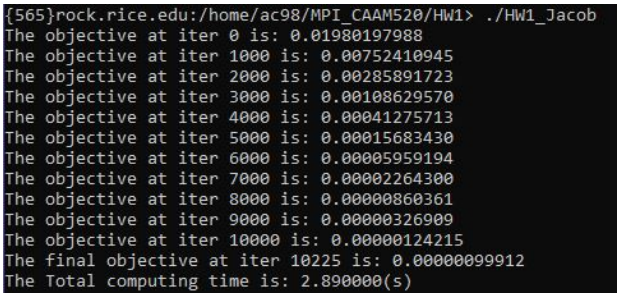

Figure 2: Weighted Jacobi method on N=100 situation

Figure 2 shows the numerical results computing using the executable file HW1Jacob with $N = 100$. It takes 10225 iterations for the algorithm to reach the tolerant misfit, the computing time is $2.89(s)$

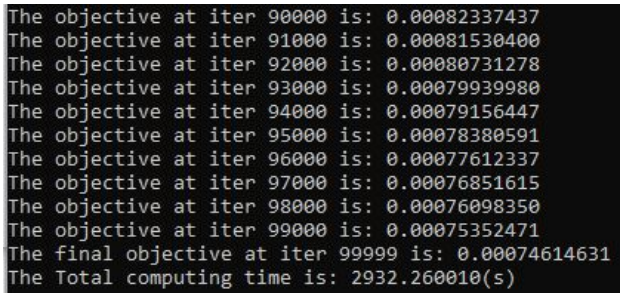

Figure 3: Weighted Jacobi method on N=1000 situation

For the situation with $N = 1000$, the convergence speed is too slow that I spent $2932(s)$ but the misfit was still away from the tolerance. The limitation for the Weighted Jacobi method is the computational run time.

Finally, I will make a operation counts for the weighted Jacob method. Assuming the multiplication between zero elements is trivial, the first step we need to compute the product of off-diagonal matrix U and wavefield u. There are $4 \times (N-1) \times 3$ subtractions for points on the edge, 4×2 subtractions for points in the corner, and $(N-2) \times (N-2) \times 4$ subtractions for the inner points.
For computing Uu $4(N-2)^2 + 12(N-1) + 8 = 4N^2 - 4N + 12$
For computing $b - Uu$ $N \times N$ subtractions
For computing $D^{-1}(b - Uu)$ $N \times N$ divisions
Updating $u_{(k+1)}$ two multiplications and one summation for all elements $3 \times N \times N$
Total operations per iteration: $9N^2 - 4N + 12$

4 Gauss-Seidel method

The Gauss-Seidel method uses the following equations:

$$x_i{}^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j{}^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j{}^{(k)} \right), i = 1, 2, ..., N$$

We first compute the product of upper triangular matrix U and wavefield u, and then compute the b-Ux. Finally, we update the solution u by computing the inverse of lower triangular matrix using forward substitution. The numerical computing results are provided below:

```
{514}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_GaussSeidel
The objective at iter 0 is: 0.01980197988
The model objective at iter 0 is: 2.55835986137
The objective at iter 1000 is: 0.00041528756
The model objective at iter 1000 is: 0.05316554010
The objective at iter 2000 is: 0.00000960365
The model objective at iter 2000 is: 0.00099410932
The final objective at iter 2712 is: 0.00000099775
The final model objective at iter 2712 is: 0.00078203512
The Total computing time is: 0.870000(s)
```

Figure 4: Gauss-Seidel method on N=100 situation

Figure 4 shows the numerical results computing using the executable file HW1Gauss-Seidel with $N = 100$. It takes 2712 iterations for the algorithm to reach the tolerant data misfit. The model misfit is the difference to the analytical solution. The computing time is $0.87(s)$, which is faster than the weighted Jacobi method.

```
The model objective at iter 98000 is: 0.53355306387
The objective at iter 99000 is: 0.00004042311
The model objective at iter 99000 is: 0.51293891668
The final objective at iter 99999 is: 0.00003886299
The final model objective at iter 99999 is: 0.49314063787
The Total computing time is: 3242.689941(s)
```

Figure 5: Gauss-Seidel method on N=1000 situation

For the situation with $N = 1000$, the convergence speed is too slow that I spent $3242(s)$ but the misfit was still away from the tolerance. The limitation for the Gauss-Seidel method is the computational run time.

Finally, I will make a operation counts for the Gauss-Seidel method. Assuming the multiplication between zero elements is trivial, the first step we need to compute the product of upper-triangular matrix U and wavefield u. There are $2 \times (N-1)$ subtractions for points on the right edge and bottom, and $(N-1) \times (N-1) \times 2$ subtractions for the inner points.

For computing Uu $2(N-1)^2 + 2(N-1) = 2N(N-1)$
For computing $b - Uu$ $N \times N$ subtractions
For computing $L^{-1}(b - Uu)$ 1 division for the first point; 1 subtraction, 1 division for the 2 to N point; 2 subtraction, 1 division for the N+1 to N*N points, $3N^2 - N - 1$ in total.
Total operations per iteration: $6N^2 - 3N - 1$

5 Successive over-relaxation (SOR) method

The SOR method uses the following equations:

$$x_i{}^{(k+1)} = (1-\omega)x_i{}^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j{}^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j{}^{(k)}\right), i = 1, 2, ..., N$$

Where the *omega* is called the relaxation factor. The method goes back to Gauss-Seidel method when $\omega = 0$. We first compute the product of upper triangular matrix U and wavefield u, and then compute the b-Ux. Finally, we update the solution u by computing the linear combination of updates from Gauss-Seidel method and solution u at last time step. The numerical computing results are provided below, the relaxation factor is $\omega = 0.9$:

```
{506}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_SOR1
The objective at iter 0 is: 0.01980197988
The model objective at iter 0 is: 2.55835986137
The objective at iter 1000 is: 0.00083741097
The model objective at iter 1000 is: 0.10753877461
The objective at iter 2000 is: 0.00003604277
The model objective at iter 2000 is: 0.00410636840
The objective at iter 3000 is: 0.00000193338
The model objective at iter 3000 is: 0.00069518760
The final objective at iter 3260 is: 0.00000099833
The final model objective at iter 3260 is: 0.00076322025
The Total computing time is: 1.080000(s)
```

Figure 6: Successive over-relaxation method on N=100 situation

Figure 6 shows the numerical results computing using the executable file HW1SOR1 with $N = 100$. It takes 3260 iterations for the algorithm to reach the tolerant misfit, the computing time is $1.08(s)$, which is faster than the weighted Jacobi method but slower than the Gauss-Seidel method. This is because the A matrix is well-posed, and the system is linear. The relaxation of the equation will mainly slow down the convergence speed. The limitation of SOR method is also the computational run time.

Finally, I will make a operation counts for the SOR method. For computing Gauss-Seidel updates $6N^2-3N-1$ Updating $u_{(k+1)}$ two multiplications and one summation for all elements $3 \times N \times N$
Total operations per iteration: $9N^2 - 3N - 1$

But we don't see a increase of computing time from Gauss-Seidel method to SOR method, there might be some optimization for computing the product of scalar with vector.

6 Conjugate gradient (CG) method

The algorithm is described below for solving our problem $Au = b$:

$$r_0 = b - Au_0$$

$$p_0 = r_0$$

$$k = 0$$

while $||r_k|| >= tol$

$$\alpha_k = \frac{{r_k}^T r_k}{{p_k}^T A p_k}$$

$$u_{k+1} = u_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$\beta_k = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

$$k = k + 1$$

From the algorithm of CG method, we can estimate that because the algorithm uses the gradient information and has approximate the Hessian matrix, if the equation is linear, it may take 1 iteration for the CG method to find out the solution. The numerical examples demonstrate our estimation. The main problem for the CG method is the storage limitations as we have the extra variables r_k and p_k to store while optimize the least squares.

```
{511}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_CG
The objective at iter 0 is: 0.01980197988
The model objective at iter 0 is: 2.55835986137
The objective at iter 1 is: 0.00000000000
The model objective at iter 1 is: 0.00082524220
The final objective at iter 1 is: 0.00000000000
The final model objective at iter 1 is: 0.00082524220
The Total computing time is: 0.000000(s)
```

Figure 7: Conjugate gradient method on N=100 situation

```
{532}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_CG
The objective at iter 0 is: 0.00199800194
The model objective at iter 0 is: 25.35562705994
The objective at iter 1 is: 0.00000000000
The model objective at iter 1 is: 0.00008325025
The final objective at iter 1 is: 0.00000000000
The final model objective at iter 1 is: 0.00008325025
The Total computing time is: 0.18999999761581420 90(s)
```

Figure 8: Conjugate gradient method on N=1000 situation

```
{516}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_CG
The objective at iter 0 is: 0.00039992001
The model objective at iter 0 is: 126.67681121826
The objective at iter 1 is: 0.00000000000
The model objective at iter 1 is: 0.00001666335
The final objective at iter 1 is: 0.00000000000
The final model objective at iter 1 is: 0.00001666335
The Total computing time is: 4.870000(s)
```

Figure 9: Conjugate gradient method on N=5000 situation

```
{519}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_CG
The objective at iter 0 is: 0.00019998000
The model objective at iter 0 is: 253.32829284668
The objective at iter 1 is: 0.00000000000
The model objective at iter 1 is: 0.00000833250
The final objective at iter 1 is: 0.00000000000
The final model objective at iter 1 is: 0.00000833250
The Total computing time is: 19.500000(s)
```

Figure 10: Conjugate gradient method on N=10000 situation

```
{522}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_CG
The objective at iter 0 is: 0.00009999500
The model objective at iter 0 is: 506.63125610352
The objective at iter 1 is: 0.00000000000
The model objective at iter 1 is: 0.00000416778
The final objective at iter 1 is: 0.00000000000
The final model objective at iter 1 is: 0.00000416778
The Total computing time is: 78.059998(s)
```

Figure 11: Conjugate gradient method on N=20000 situation

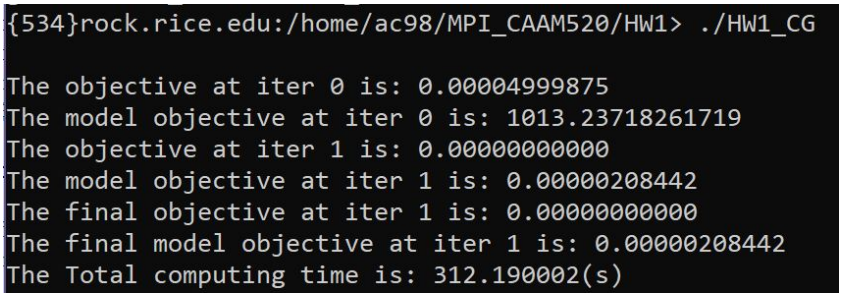

Figure 12: Conjugate gradient method on N=40000 situation

Figure 13: Conjugate gradient method on N=50000 situation

From the above figures, we can find that the memory will corrupt when $N = 50000$ and the computing time of conjugate gradient method is increasing as the order of N^2 for different N values. But no matter what N values we are using, the conjugate gradient method will reach the ideal misfit tolerance with one iteration. This is because the CG method is based on gradient and approximate Hessian that can converge in one iteration for the well-posed linear equations.

N value	Computing time (s)
1000	0.190
5000	4.870
10000	19.500
20000	78.060
40000	312.190

Table 1: Table for computing time using Conjugate gradient method

Table 1 logs computing time requirements for Conjugate Gradient method at different N value. The computing time of conjugate gradient method is increasing approximately as the order of N^2 for different N values.

Finally, I will make the estimations for memory requirements and computing operations for Conjugate gradient method.

For memory requirements, I am writing in double precision (64 bit/8 bytes), we have the equations b, variables u_k, r_k, p_k and working matrix $temp$ to store, the storage requirements for the other variables are trivial. The storage requirements are approximately:

$$5 \times N \times N \times sizeof(double) = 320N^2(bits)$$

For N = 50000, we need

$$8 \times 10^{11}(bits) \approx 93.1(gigabytes/GB)$$

Finally, for operation counts, first consider the product of FD operator matrix A and wavefield u. There are $N \times N multiplications for diagonal items$ $4 \times (N-1) \times 3$ subtractions for points on the edge, 4×2 subtractions for points in the corner, and $(N-2) \times (N-2) \times 4$ subtractions for the inner points. Therefore, computing Au needs: $N^2 + 4(N-2)^2 + 12(N-1) + 8 = 5N^2 - 4N + 12$ operations.

For computing $\alpha_k = \frac{{r_k}^T r_k}{{p_k}^T A p_k}$: $2N^2$ multiplications, $2N^2 - 2$ summation, compute Ap_k for $5N^2 - 4N + 12$, 1 division;

For computing $u_{k+1} = u_k + \alpha_k p_k$: N^2 summation and N^2 multiplications;

For computing $r_{k+1} = r_k - \alpha_k A p_k$: N^2 subtractions and N^2 multiplications, Ap_k has been stored in working matrix already;

For computing $\beta_k = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$: $2N^2$ multiplications, $2N^2 - 2$ summation, 1 division;

For computing $p_{k+1} = r_{k+1} + \beta_k p_k$: N^2 summation and N^2 multiplications.

Total operation number per iteration is: $19N^2 - 4N + 10$