CFM First Project Documentation

Release 0.0.1

Sayop Kim

February 13, 2016

Contents

Contents:

CHAPTER 1

Code Instruction

The present project is aimed to develop a computer program for solving a steady solution with different numerical method. The code being used for answering all the question here is written with Python language. This program is to run with simple command:

```
$ python main.py
```

Before running this Python script, please make sure that you have also *functions* folder that contains pre-defined function for specific purpose use. The problem 1 and problem 2 have different sets of python scripts. However running the script is working in the same methodology.

1.1 Quick instruction for problem 1

To run the simulation, you only need to simply open the file named *main.py* using editor for example, VI on unix system:

```
$ vi main.py
```

You should be able to find the following line:

```
# maximum iterations
# For problem1-b, set nIters to 1
# For poroble1-c, set nIters big enough to run until it converge
nIters = 20
```

This is to set maximum iteration number. To run the steady-state linear equation, set it to 1 in order not to repeat redundant iteration.

```
# grid: i,j,k resolution
iDim = 20
jDim = 1
kDim = 1
# x: spatial dimension of cube
xmin = 0.0
xmax = 1.0
# Boundary condition
# Dirichlet BC:
phi_xLeft = 1
phi_xRight = 0
```

In this script, you will also find *iDim* for setting up grid size, and *xmax* for computational domain dimension. Here we are supposed to use Dirichlet boundary condition, so you should speicify the required ϕ at left and right boundaries.

If you run the simulation with this script, you will see the following files in the same directory:

```
sayop@reynolds:~/data/GaTech-CourseWorks/ME-CFM/CFM01/src/pr1$ ls
dataOut.csv  functions  main.py  prob1Solution.png
```

dataOut.csv is the text file for having numerical solution and analytical solution. And *prob1Solution.png* is a automatically produced line plot for the solution comparison.

1.2 Quick instructions for problem 2

You will have following file and folder for the problem 2 and multi-grid:

```
$ sayop@reynolds:~/data/GaTech-CourseWorks/ME-CFM/CFM01/src/multigrid$ ls
functions  main.py  plotCenterData.py
$ cd /functions
$ ll
drwxrwxr-x 2 sayop sayop 4096 Feb 12 23:01 ./
drwxrwxr-x 3 sayop sayop 4096 Feb 12 23:35 ../
-rw-rw-r-- 1 sayop sayop  859 Feb 10 21:48 exactSol.py
-rw-rw-r-- 1 sayop sayop 1176 Feb 10 21:56 exactSol.pyc
-rw-rw-r-- 1 sayop sayop  258 Feb 10 21:48 grid.py
-rw-rw-r-- 1 sayop sayop  471 Feb 10 21:56 grid.pyc
-rw-rw-r-- 1 sayop sayop 3005 Feb 12 22:39 multigrid.py
-rw-rw-r-- 1 sayop sayop 2582 Feb 12 22:39 multigrid.pyc
-rw-rw-r-- 1 sayop sayop 4016 Feb 11 02:03 numericalMethods.py
-rw-rw-r-- 1 sayop sayop 3886 Feb 11 01:50 numericalMethods.py.bak
-rw-rw-r-- 1 sayop sayop 2701 Feb 11 02:04 numericalMethods.pyc
-rw-rw-r-- 1 sayop sayop 2105 Feb 10 21:48 post.py
-rw-rw-r-- 1 sayop sayop 2791 Feb 10 21:56 post.pyc
```

To run the second simulation with point-iterative method, Jacobi, Gauss-Seidel and SOR for example, you also need to open the file named as *main.py* in /src/pr2/ directory:

```
$ vi main.py
```

First user specified-inputs you will see in this file are:

```
# iSwitch for numerical method choice: 0-Jacobi, 1-Gauss-Seidel
iSwitch = 1
iMultiGrid = 1
```

Here, *iSwitch* is supposed to define which method you want to use. *iSwitch=0* will allow you to use Jacobi. Or Set it to 1 to use Gauss-Seidel method and SOR method.

Followings are used for iteration controls:

```
# maximum iterations
nIters = 99999
# if residual rate comes below this criterion, the iteration will stop!
convergeCrit = 0.001
```

Set *nIters* big enough such that your simulation can go over the convergence point. *convergeCrit* is set to 0.001 as requested in the problem.

Followings are grid setup intpus:

```
# grid: i,j resolution
iDim = 40
jDim = 40
```

```
# x, y: spatial dimension of N^2 nodes
xmin = 0.0
xmax = 1.0
ymin = 0.0
ymax = 1.0
```

For the setup of SOR, you can specify α from the following line:

```
# Relaxation coefficient for SOR: applies to Jacobi and Gauss-Seidel
relaxCoeff = 1.0
```

CHAPTER 2

Results

2.1 Problem 1 - a

Develop a finite difference algorithm using central differences for the solution of the transport equation. Describe the essential steps.

Given equation:

$$U\frac{\partial \phi}{\partial x} = \frac{\partial}{\partial x}\left(\Gamma \frac{\partial \phi}{\partial x}\right) + Q$$

To make the given form of equation convenient to be converted into descritized algebraic equations, terms having dependent variables are sorted out in the left hand side leaving Q in the right hand side as:

$$U\frac{\partial \phi}{\partial x} - \frac{\partial}{\partial x}\left(\Gamma \frac{\partial \phi}{\partial x}\right) = +Q$$

Now, the left hand side is composed of convection term and diffusion term, respectively. These two term are going to be referred to as *divergence* term and *laplacian* term, respectively.

- Divergence term descritization:

$$U\frac{\partial \phi}{\partial x} : \;\; U\left(\frac{\phi_{i+1} - \phi_{i-1}}{\Delta x}\right)$$

  Again, further descritzation should be done for the remaining first derivative in the middle points at $i + 1/2$ and $i - 1/2$ and it leads to:

$$-\frac{1}{\Delta x}\left[\frac{\Gamma_{i+1/2}\left(\phi_{i+1} - \phi_{i-1}\right) - \Gamma_{i-1/2}\left(\phi_i - \phi_{i-1}\right)}{\Delta x}\right]$$

- Laplacian term descritization:

$$-\frac{\partial}{\partial x}\left(\Gamma \frac{\partial \phi}{\partial x}\right) : \;\; -\frac{1}{\Delta x}\left[\left(\Gamma \frac{\partial \phi}{\partial x}\right)_{i+1/2} - \left(\Gamma \frac{\partial \phi}{\partial x}\right)_{i-1/2}\right]$$

Constructing every terms with source term in right hand side at i node point becomes such that the final form has three terms with respective corresponding i node and neightbor points, $i - 1$ and $i + 1$ will become:

$$a_{i-1}\phi_{i-1} + a_i\phi_i + a_{i+1}\phi_{i+1} = Q_i$$

where,

$$a_i = \frac{\left(\Gamma_{i+1/2} + \Gamma_{i-1/2}\right)}{\Delta x^2}$$

$$a_{i-1} = -\left(\frac{\Gamma_{i-1/2}}{\Delta x^2} + \frac{U}{\Delta x}\right)$$

$$a_{i+1} = -\left(\frac{\Gamma_{i+1/2}}{\Delta x^2} - \frac{U}{\Delta x}\right)$$

Here again we need to quantify the diffusion coefficient Γ at middle points where are not actually in presence. Therefore, linear interpolation is done for those middle point for having diffusivity in the second derivative terms:

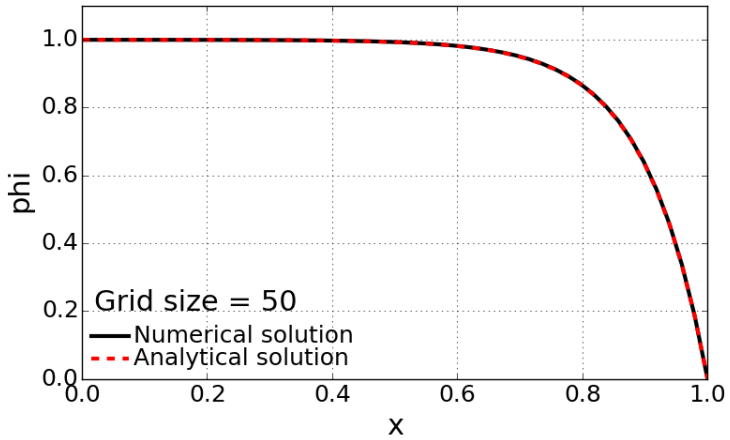
$$\Gamma_{i+1/2} = \frac{1}{2}\left(\Gamma_{i+1} + \Gamma_{i-1}\right)$$

Now we have single algebraic equation for each single node point. The node point is now linked to the neighbor right next to it. Thus, we can construct tri-diagonal matrix for those coefficients when we construct system of linear equations in 1-dimensional problem: $A\Phi = Q$. A can be described as a matrix or tensor form with two ranks. Φ and Q are vectors.

2.2 Problem 1 - b

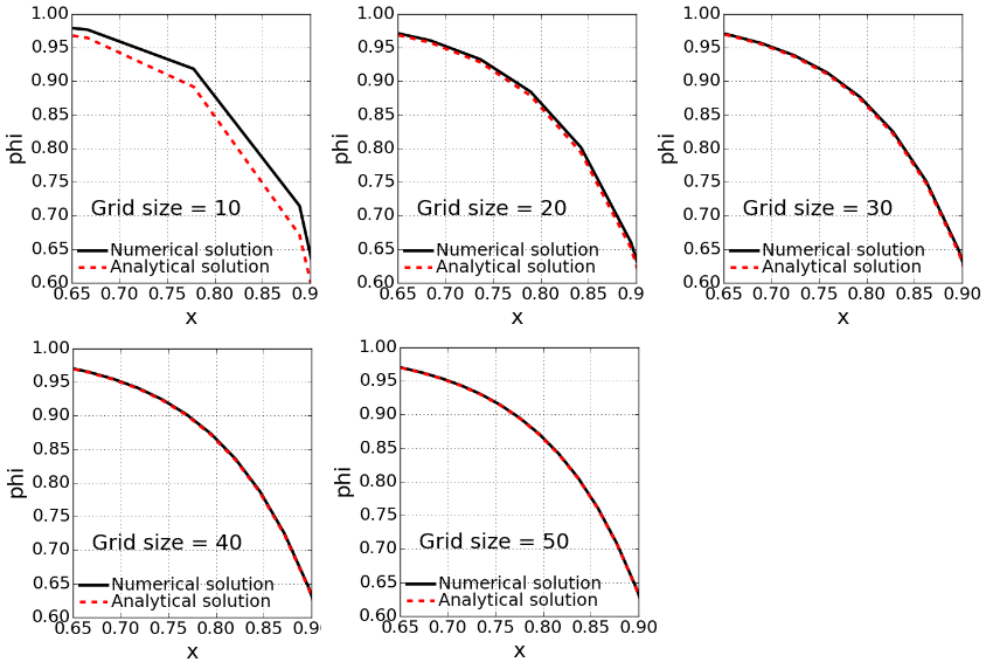
Given conditions: U = 1, Γ = 0.1, and Q = 0. Use TDMA to find ϕ.

1. **Plot** ϕ vs. x on the same graph using grid sizes of N = 10, 20, 30, 40, 50 and **compare** your result to the analytical solution.
    1. *Grid size, N = 50* with full scale



- Discussions on numerical solution
    - As observed in the first figure, the final solution looks having strong convection effect with such a flat upstream solution and getting (diffused) down to the boundary fixed value at right wall.
    - The numerical solution with 50 grid points is having consistent match with exact solution.

2. Effect of grid size on numerical solution in magnified scale



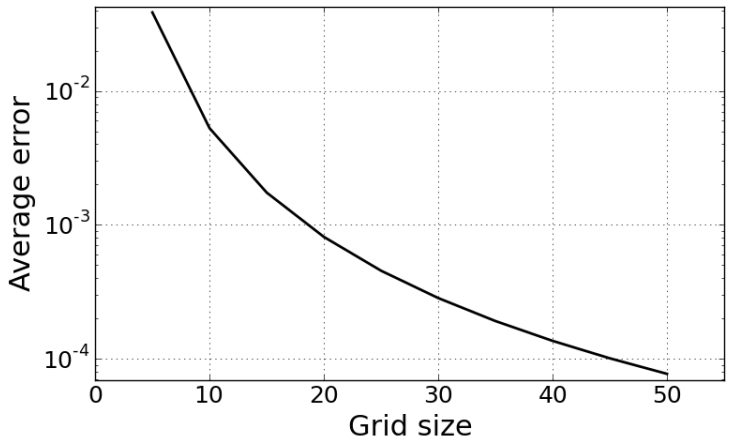
- Discussions on grid size effect
  - Deviation of numerical simulation from the analytic solution is getting bigger as the grid become coarser.
  - Biggest inaccuracy in numerical solution (descretization errors) with the case of $N = 10$.

2. **Plot** the average error as a function of N in a log scale and normalization with respect to grid size.

   Here the average error is defined as Root Mean Square of deviation of numerical solution from the analytical solution. The calculation was done with the following definition.

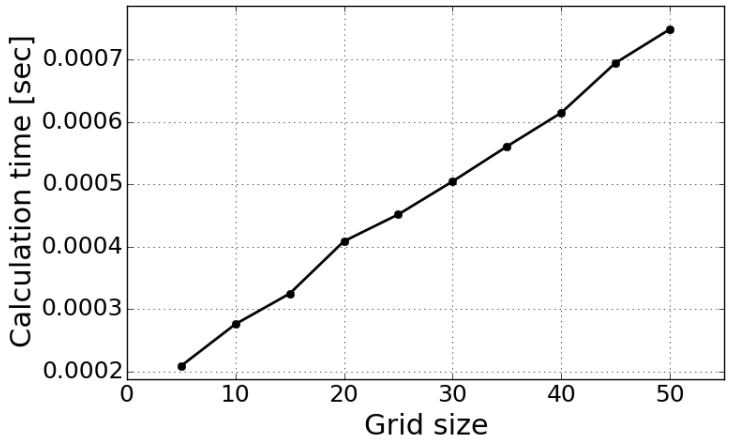
$$err = \frac{\sqrt{\sum_{i=1}^{N}(\phi_i - \phi_i^{exac})^2}}{N}$$

- Discussion
  - Here, the average error is dropping dramatically as the smaller grid size is used.
  - As the central descritization is applied for finite difference, the accuracy is improved with second order accuracy.



3. **Examine** how the calculation time changes with N and **evaluate** the time complexity of the algorithm.

   The computational time is defined as the consumed real calculation time while the numerical solution is being solved. The time is measured only during the numerical scheme process, meaning that pre-processing and post-processing time has been eliminated.



- Discussion
  - The compute time is increased linearly as the grid size become bigger.
  - The linear increment of time is because of the 1-dimensional grid use.
  - The currently used solver, TDMA sweeps back and forth one time along the grid points to have solutions of system of linear equations. Thus, the actually calculation time must be only dependent on the number of node points.

2.3 Problem 1 - c

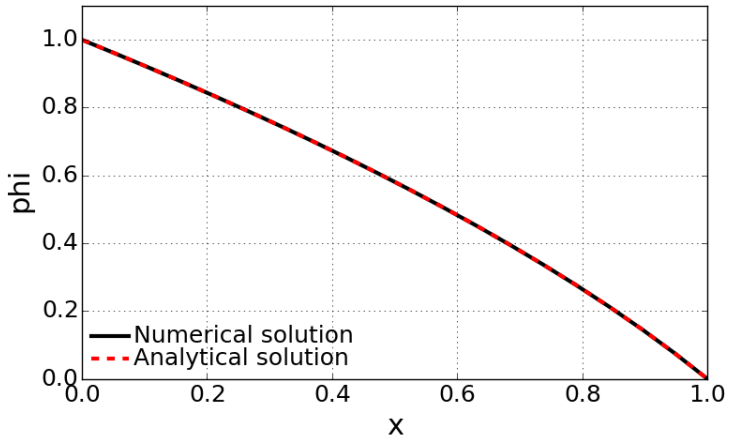
Given conditions: $U = 0$, $\Gamma = 0.1 + 0.1\ \phi$, and $N = 20$.

1. Find ϕ using TDMA for following Q and corresponding exact solution cases:

- About solution method

  - Compared to the problem 1-b, this is non-linear problem because the diffusivity is dependent on the solution ϕ. For this reason, it can't be solved with simple block iterative method like TDMA. The reason is because we cannot build tri-diagonal A matrix for this case due to the non-linear behavior.
  - The solution method for this problem has to be iterative method to track the converging solution.
  - The iterative method updates the initial guess of solution space and seeks the converged value by repeating the numerical algorithm.
  - Initially ϕ_i space is initialized with certain value. Here, every points is initialized with zero.
  - Then, the every step of TDMS method will propagate the updated info from the guessed ϕ.
  - The code has a function to check the residual value from the updated ϕ and quantify how much it propagates from the solution of previous step.
  - It repeats until the deviation of solution from the past reaches to zero level. (NOTE: the residual will never reach to zero level because of truncation errors of our finite difference equations)

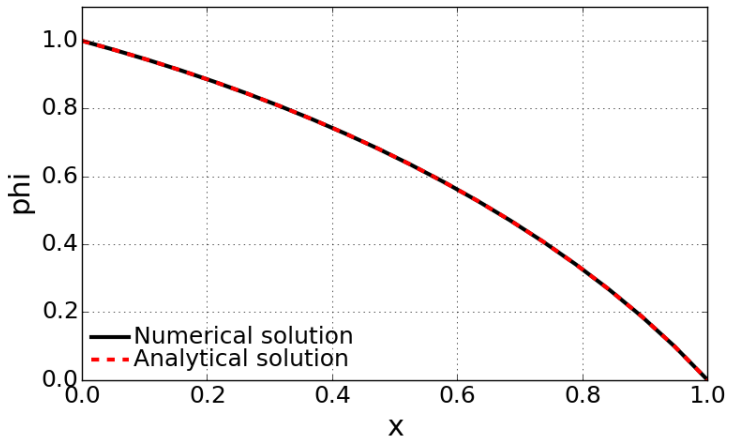
- CASE1: $Q = 0, \quad \phi(x) = -1 + \sqrt{4 - 3x}$

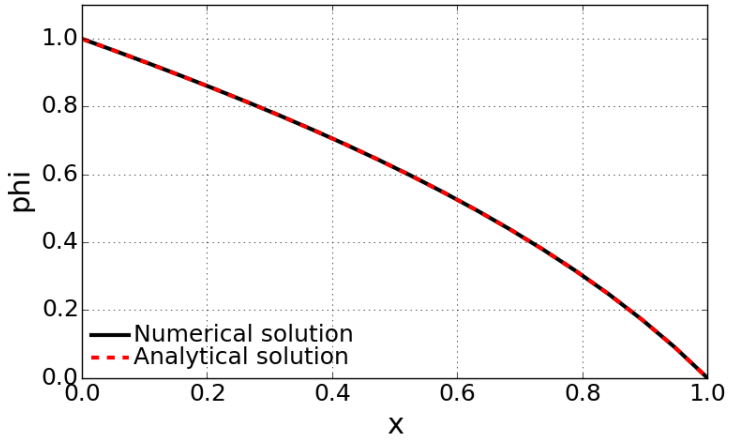


  - Converged at 5th iterations with residual of 0.000876728469266
  - Average error = 9.0129271003e-06
  - Calculation time [sec] = 0.001337

- CASE2: $Q = 0.1, \quad \phi(x) = -1 + \sqrt{4 - 2x - x^2}$
  - Converged at 5th iterations with residual of 0.000716942145114
  - Average error = 1.01159413052e-05
  - Calculation time [sec] = 0.001383

- CASE3: $Q = 0.1x, \quad \phi(x) = -1 + \sqrt{4 - (8/3)x - x^3/3}$


phi
x
Numerical solution
Analytical solution
0.0
0.2
0.4
0.6
0.8
1.0


phi
x
Numerical solution
Analytical solution
0.0
0.2
0.4
0.6
0.8
1.0

  - Converged at 5th iterations with residual of 0.000707116960016
  - Average error = 8.99517612803e-06
  - Calculation time [sec] = 0.001404

- Discussions on comparison between different source terms
  - Only difference between these three cases has come from the source term deviation.
  - Since we set U to zero, we do not expect any convective effect. Only matter of the problem is diffusion and source terms.
  - Pure diffusion problem with constant diffusivity will result in linear ϕ profile along the x-direction.
  - The slight bump from the pure diffusion problem is a combination of non-linear diffusivity and the source terms like heat source in heat conduction problem.
  - The CASE 1 represents the pure diffusion problem with non-linear diffusivity. Since the diffusivity is proportional to ϕ, higher value of ϕ is more likely to propage to the lower ϕ region.
  - CASE 2 and CASE 3 are showing source term effect. When the heat conduction with heat source is imagined, temperature rises with the heat source.
  - CASE 3 has less heat source as it goes far away from the origin. It then leads to less bump compared to the CASE 2.

2. *Comparison in calculation time*

	calculation time	total # of iterations
2.	0.000417	1
3. CASE 1	0.001337	5
3. CASE 2	0.001383	5
3. CASE 3	0.001404	5

   - Discussion
     - As discussed earlier, the previous problem 1-b was a linear set of equations. So it can be solved with linear system of equations using block iterative method called here TDMA.
     - So, the TDMA in problem 1-b was not iterated multiple times.
     - The non-linear problem here is asking us to seek the converged solution with iterative method.
     - Three differenct source terms cases requires only 5 iterations until it satifies the convergence criteria. Here, convergence criteria was set to 0.001.
     - Since the solution method is identical, the calculation time is almost identical because it proceeds same repetition of TDMA 5 times.

- Slight change of calculation time might have come from the compute resource occupied by the other software.

2.4 Problem 2 - a

Given two-dimensional steady heat equation (Poisson's equation:

$$\lambda \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + Q = 0$$

Discretize the PDE using a central difference scheme to generate a matrix equation $[A]T = Q$.

Applying second order accuracy central scheme to each second derivative with respect to x and y gives

$$\lambda \left[\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i,j-1} - 2T_{i,j} + 2T_{i,j+1}}{\Delta y^2} \right] + Q_{i,j} = 0$$

To construct the coefficient matrix of system of linear equations, the coefficients of each neighbor point should be organized as shown below:

$$a_{i-1,j}T_{i-1,j} + a_{i+1,j}T_{i+1,j} + a_{i,j-1}T_{i,j-1} + a_{i,j+1}T_{i,j+1} = Q_{i,j}$$

where each coefficietns are recast in terms of diffusivity and grid size:

$$a_L = a_R = \frac{\lambda}{\Delta x^2}$$

$$a_B = a_U = \frac{\lambda}{\Delta y^2}$$

and

$$a_{i,j} = -\left(\frac{2\lambda}{\Delta x^2} + \frac{2\lambda}{\Delta y^2} \right)$$

Here, L, R, B, and U indicate 'Left', 'Right', 'Bottom' and 'Up' neighbors, respectively.

The coefficients based on a single point (i, j) will compose $i \times j$-th row.

$$AT = Q$$

where T and Q are vector having n^4 elements:

$$\begin{bmatrix} T_{1,1} & T_{1,2} & \cdots & T_{1,n} & T_{2,1} & \cdots & T_{2,n} & \cdots & T_{i,j} & \cdots \\ T_{n,n} \end{bmatrix}^T$$

$$\begin{bmatrix} Q_{1,1} & Q_{1,2} & \cdots & Q_{1,n} & Q_{2,1} & \cdots & Q_{2,n} & \cdots & Q_{i,j} & \cdots \\ Q_{n,n} \end{bmatrix}^T$$

Here, A matrix is not formed tridiagonal as it is seen in the first problem because the current set of solution is defined in 2-dimensional domain. For that reason, coefficient matrix should contain all four neighbor's constant. The resulting A matrix will have $N^2 \times N^2$ elements.

2.5 Problem 2 - b

For each of the Jacobi, Gauss-Seidel, and SOR methods, **express** $T_{i,j}^k$, the temperature at node (i, j) and iteration $k + 1$, in terms of the temperatures of its neighbors at either the current or past iterations.

- Jacobi method

  Jacobi method is to update every node points with solution from past iterations. Therefore the tempertaure on the neighbor nodes represents previous teme level, so it does not matter with the sweep direction.

$$T_{i,j}^{k+1} = \frac{1}{a_{i,j}} \left[Q_{i,j} - a_L T_{i-1,j}^k - a_R T_{i+1,j}^k - a_B T_{i,j-1}^k - a_U T_{i,j+1}^k \right]$$

- Gauss-Seidel method

  Gauss-Seidel method is different from the Jacobi method in terms of neighbor points' time level. Contrary to the Jacobi's, the Gauss-Seidel method allows us to use lately updated solution. When it sweeps in i-direction, and j-direction and is located on (i, j) node, left and bottom point data have already been updated. Instead using past time level data, Gauss-Seidel method is to use the latest ones such that it promotes the faster convergence.

  Now this method propagates the infomation from left and bottom to right and up at one time level.

$$T_{i,j}^{k+1} = \frac{1}{a_{i,j}} \left[Q_{i,j} - a_L T_{i-1,j}^{k+1} - a_R T_{i+1,j}^k - a_B T_{i,j-1}^{k+1} - a_U T_{i,j+1}^k \right]$$

- Successive Over-Relaxation (SOR)

  Successive over-relaxation (SOR) is a variant of the Gauss-Seidel method, resulting in faster convergence. Thus, the basic form of this method is almost equivalent to the Gauss-Seidel method. The only difference is to put relaxation coefficient in the increment of solution. This can be expressed by:

$$T_{i,j}^{k+1} = T_{i,j}^k + \frac{\alpha}{a_{i,j}} \left[-a_{i,j} T_{i,j}^k + Q_{i,j} - a_L T_{i-1,j}^{k+1} - a_R T_{i+1,j}^k - a_B T_{i,j-1}^{k+1} - a_U T_{i,j+1}^k \right]$$

2.6 Problem 2 - c

1. **Plot** both temperature contours and centerline profiles (T vs. x or y through center) for each different method.

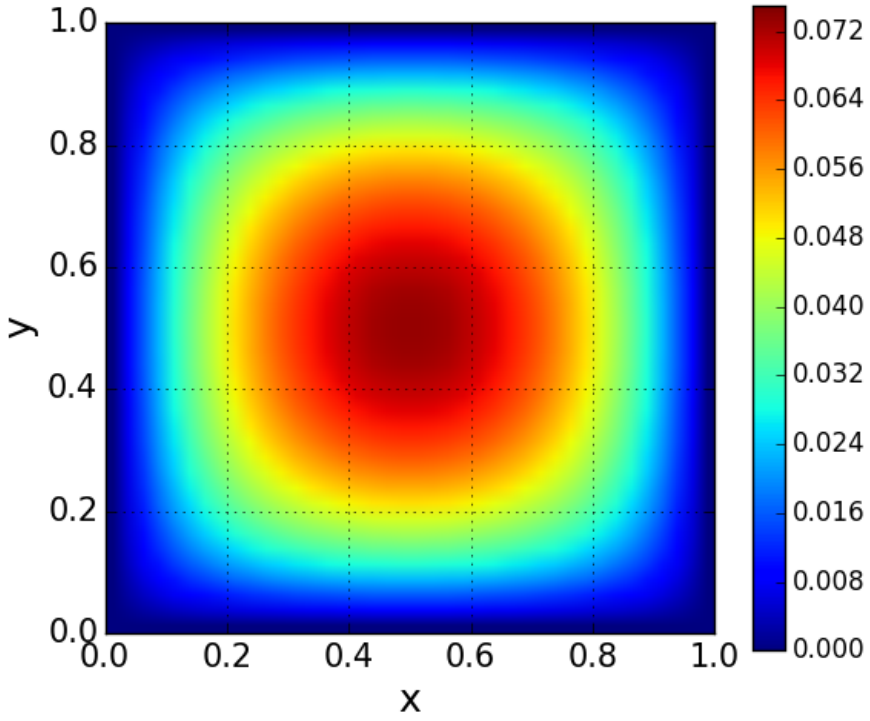
- CASE1: Jacobi method

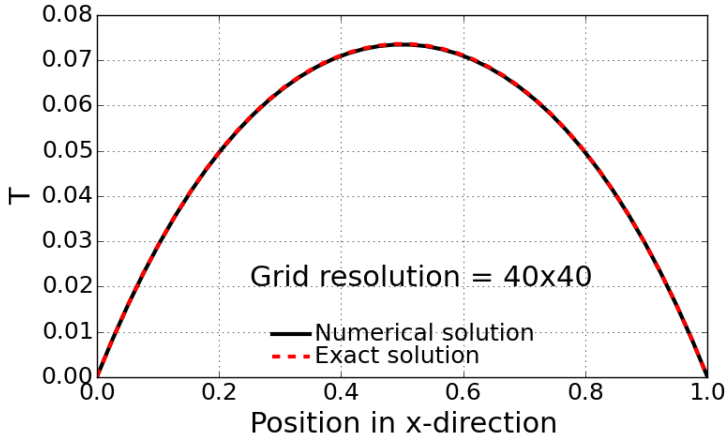
  1. Temperature contour

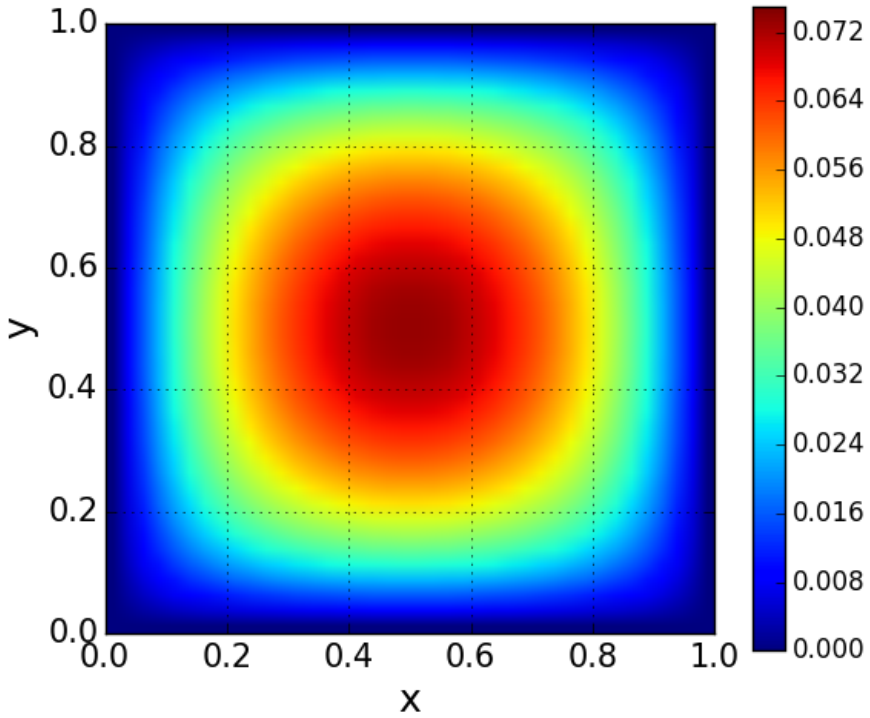
  2. Temperature along the center line

  - Converged at 2068th iteration
  - Averaged error: 1.75249104594e-06
  - Calculation time: 27.746111 [sec]

- CASE2: Gauss-Seidel method

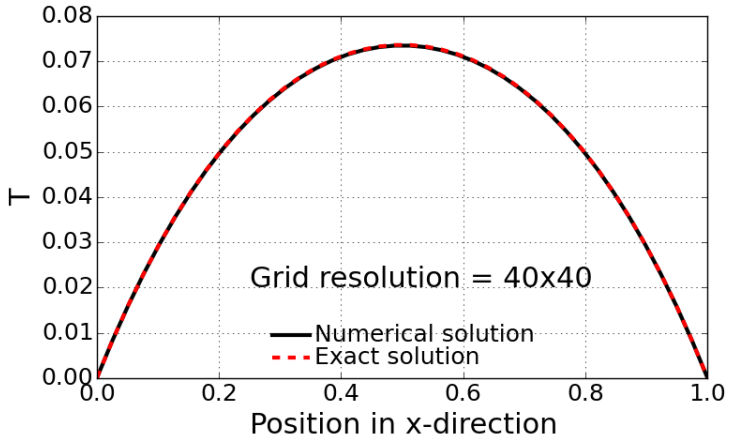

1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.072
0.064
0.056
0.048
0.040
0.032
0.024
0.016
0.008
0.000


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
T
Grid resolution = 40x40
Numerical solution
Exact solution
0.0
0.2
0.4
0.6
0.8
1.0
Position in x-direction



1. Temperature contour

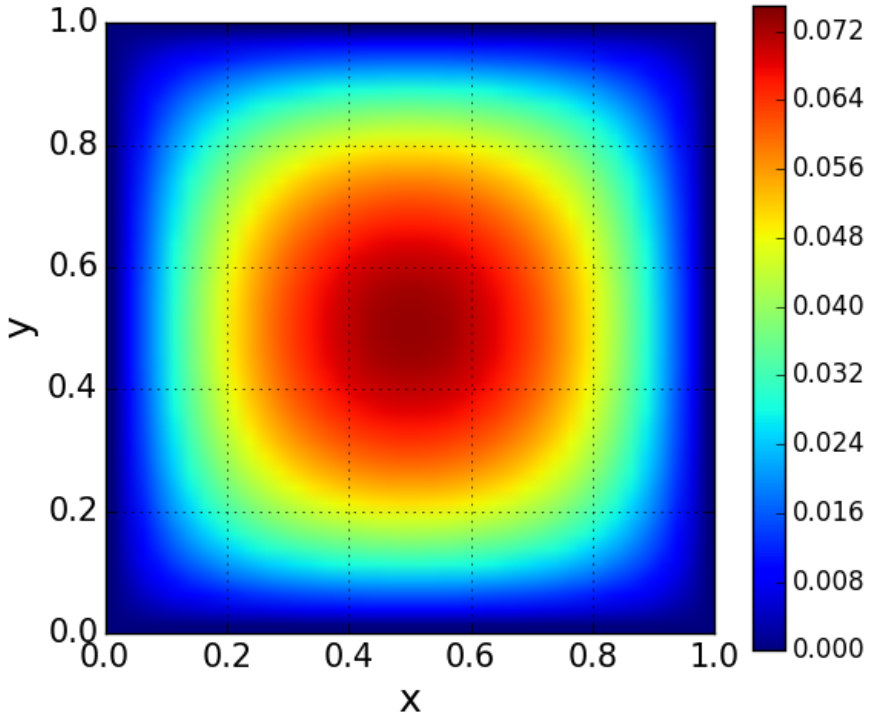
2. Temperature along the center line



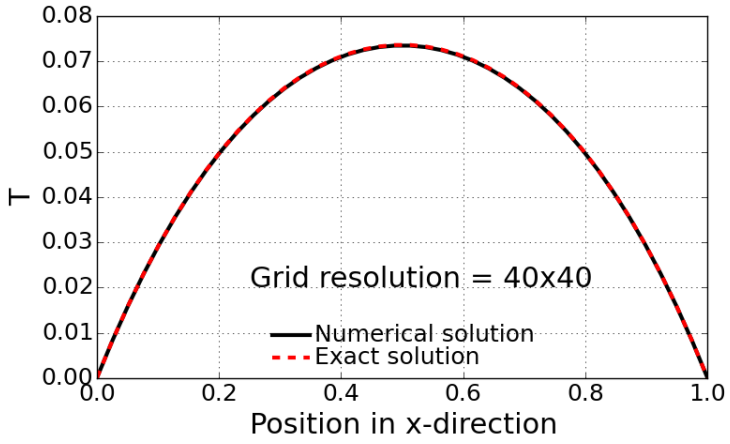
- Converged at 1034th iteration (faster than Jacobi method by a factor of 2)
- Averaged error: 1.74474732047e-06
- Calculation time: 13.999504 [sec]

- CASE2: Successive Over-Relaxation (SOR) method (Variant form of G-S method): $\alpha = 1.2$

  1. Temperature contour



  2. Temperature along the center line



  - Converged at 689th iteration (faster than Jacobi method by a factor of 3)
  - Averaged error: 1.72946829409e-06

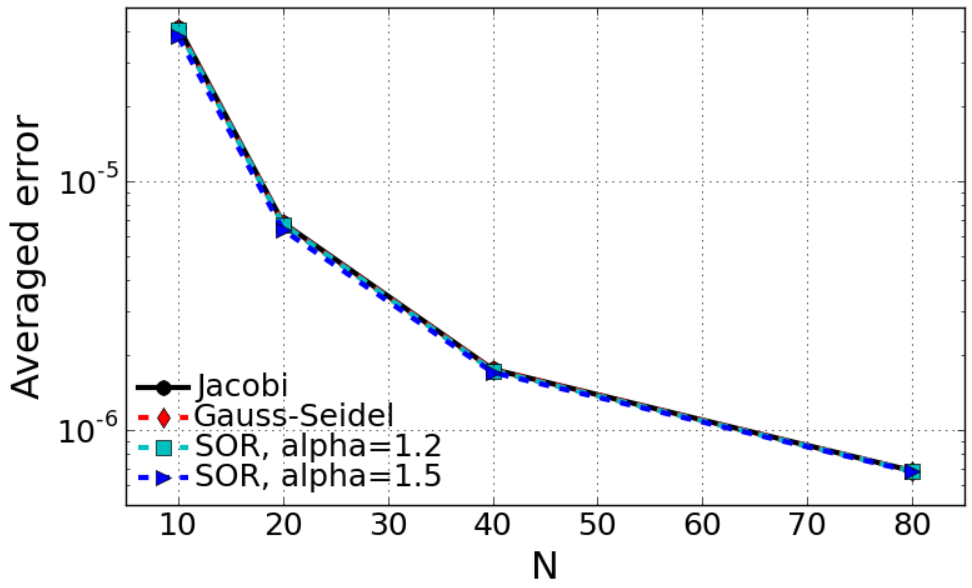
- Calculation time: 9.319121 [sec]

2.7 Problem 2 - d

Compare average error for grids [10,10], [20,20], [40,40], [80,80].

Here, the average error is defined as solution difference between numerical solution at final iteration and analytical solution:

$$err = \frac{\sqrt{\sum_{i=1}^{N\times N}(\phi_i - \phi_i^{exac})^2}}{N^2}$$



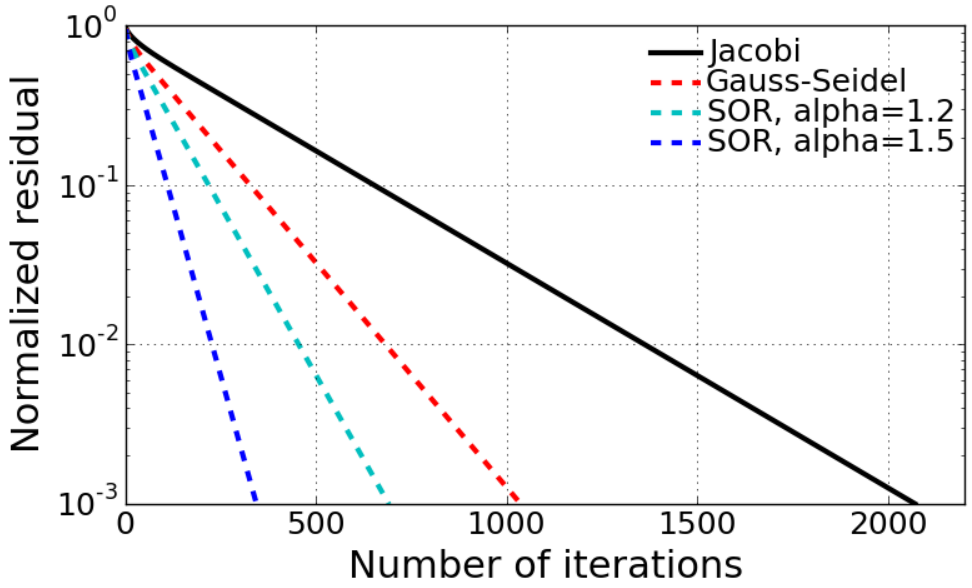
- Discussions
  - Here we see again the second order accuracy with changed of grid size. This is an outcome of central scheme finite difference method.
  - Accuracy is improved by only decreasing grid size.
  - Impact of numerical scheme on accuracy is NOT noticeable.
  - The details of error quantity comparison shown below are showing slight improvement of accuracy with advanced schemes, SOR with higher relaxation coefficient for instance.

Average error according to different schemes

N x N	Jacobian	Gauss-Seidel	SOR (alpha = 1.2)	SOR (alpha = 1.5)
10x10	4.17597212966e-05	4.12637711927e-05	4.07371621739e-05	3.80998823969e-05
20x20	6.79949662608e-06	6.73688065785e-06	6.7040448814e-06	6.36361405949e-06
40x40	1.75249104594e-06	1.74474732047e-06	1.72946829409e-06	1.70191775112e-06
80x80	6.84568684695e-07	6.83126277673e-07	6.82203360121e-07	6.78950654795e-07

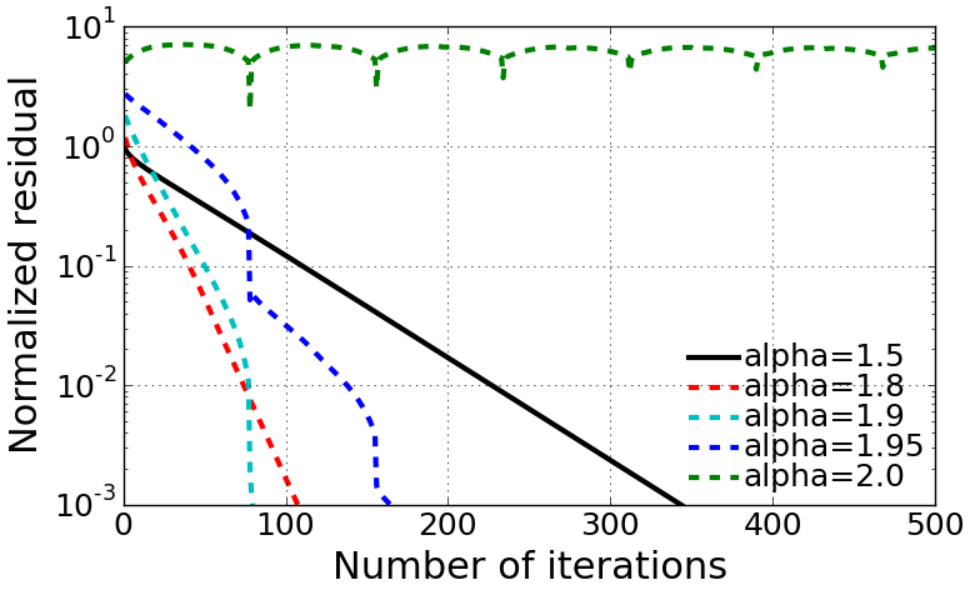
2.8 Problem 2 - e

Plot and discuss convergence rate for [40,40].



- Discussions
  - Using faster convergence scheme proves dramatical improvement of convergence performance.
  - This is because the faster convergence scheme is basically supposed to propagate the infomation faster than Jacobi method, which is based on past level info use while updating the next time level.
  - Relaxation coefficient in SOR method have an additional promotion in fast convergence.

- Effect of relaxation factor on convergence performace
  - Discussions
    - Using higher value of relaxation factor improves substantial fast convergence.
    - The relaxation factor cannot be used beyond 2. So recommended use of α for SOR is $1 < \alpha < 2$
    - $\alpha > 2$ is giving an oscillated residual behavior and its mean value keeps consistent over repeated interation and it never ended.



* Even if it is converged, the use of α greater than 1.8 is also showing an unstable residual evolution in time.
* Best convergence performance was achieved with α = 1.9, because it ends at less than 100 iteration with current case setup.

2.9 Problem 2 - f

Compare computational time obtained with different methods and evaluate the time complexity of the methods.

The scaling test has been done for the different numerical methods on 40x40 grid case. The computational time was measured solely for the essential parts of computing process. Pre- and Post-processing computation was not included in that time measurement. The following table show the direct comparison of time consumed with different methods as well as average error.

methods	Compute time	average error
Jacobi	29.486377	1.75249104594e-06
G-S	14.826672	1.74474732047e-06
SOR 1.2	9.79673	1.72946829409e-06
SOR 1.5	4.852003	1.70191775112e-06
SOR 1.8	1.53902	1.39199894688e-06
SOR 1.9	1.150155	5.73021575907e-07
SOR 1.95	2.353955	5.96044300326e-07

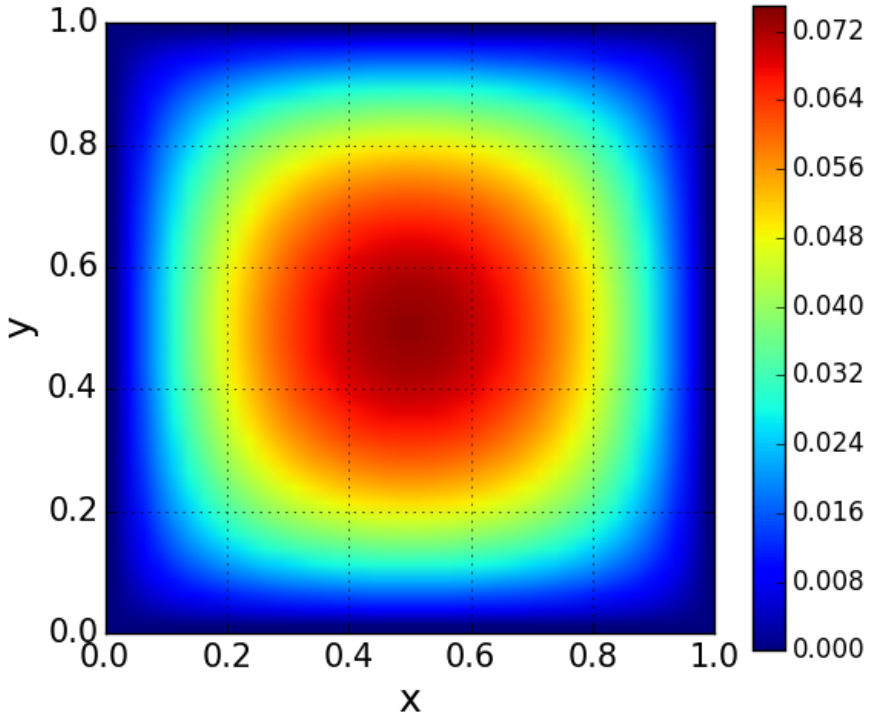
- Discussions
  - As observed in the previous section, the best performance was achieved with choice of α = 1.95 usign SOR method.
  - The computational time is reduced almost by a factor of 25 from the worst case.
  - This dramatical improvement is obtained even with slightly improved average error.
  - All of the numerical method used here is esentially based on iterative method which is supposed to propagate the updated info to the final solution.

- For that reason, the over-relaxed increment of solution is actually aimed to promote the speed of infomation propagation such that it fastly reaches to the converged solution.
- However, exceesive promotion with greater value than $\alpha = 2$ is likely to make over-shoot effect of increment that incudes divergence of the solution.

2.10 Problem 2 - g

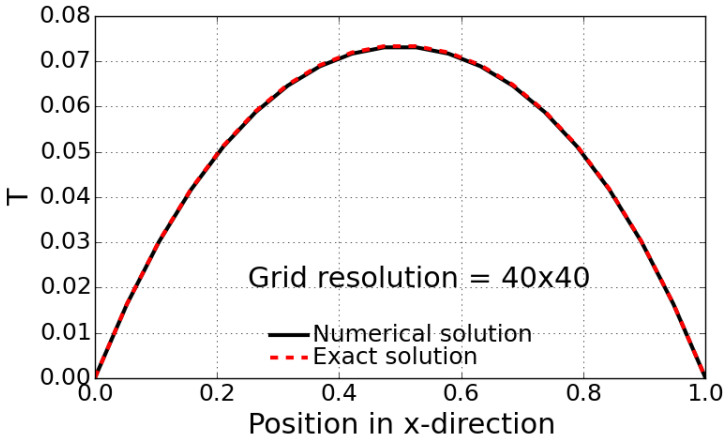
Solve the problem using a 2-level geometric multigrid method. Compare the computational time with other methods.

Following figures show the contour and line plots of multigrid solution with Gauss-Seidel method. Those two figures are illustrating converged temperature solution consistent with the previous solution obtained by the other methods.



The following tests were performed on 40x40 grid case. Four different setup has been set to find the performace of multi-grid method.

test case	Main-iteration # at final	Computation time
Jacobi w/o MG	2069	42.83 sec
Jacobi w/ MG	20	14.07 sec
G-S w/o MG	1035	27.49 sec
G-S w/ MG (1)	8	13.28 sec
G-S w/ MG (2)	14	13.78 sec



- Discussions
  - The multi-grid has been tested on different set of numerical methods and different sub-iteration numbers. The best solution was achieved by using 10 sub-iteration on coarse grid with properly specified e matrix.
  - The computational time with multi-grid method was reduced by a factor of 2 for G-S and 3 for Jacobi.
  - Interestingly, using multi-grid method and its fast convergence cover up the slowness of Jacobi method: Note the computational time between G-S and Jacobi is not significant when multi-grid is used.

- Effect of initial condition for e matrix on coarse grid.
  - The final two test cases with G-S with multi-grid were conducted with the different initial condition for e matrix on coarse grid.
  - The first one 'G-S w/ MG (1)' runs with non-zero value of initial condition for e matrix. The initial condition was set from the half of errors between current time level solution and exact solution.
  - The reason that 'half' of the error was used is to avoid the divergence. Under relaxing effect in initial condition was helpful to improve the convergency in coarse grid.
  - On the other hand, the zero set of initial condition for e matrix required more iterations. But total computational time is not significantly far away from the first test case, 'G-S w/ MG (1)'.