BOISE STATE UNIVERSITY

BSU

FINAL PROJECT

MULTIGRID FOR SOLVING ELLIPTIC DIFFERENTIAL EQUATIONS

Author
BRIAN KYANJO

Supervisor
Prof. GradyWRIGHT

May 6, 2021

BOISE STATE UNIVERSITY

Contents

1 Introduction

1.1 Background

Many problems that arise from physical applications give us a natural feel to the multigrid methods. These methods have been applied directly to non-linear problems, and many researchers have used them to perform different studies on a variety of problems. In fact these methods were the first to overcome the complexity barrier. However here we are going to concentrate on using them to solve elliptic partial differential equations (solutions defined by boundary conditions), e.g poision equation (1).

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \tag{1}$$

1.2 General Objectives

- To examine why the method works [10].
- To use the method to solve elliptic partial differential equations (PDEs)
- To apply Fourier Analysis to the two-grid operator.
- To experiment with different smothers e.g. red-black and Gauss-Seidel to see how errors are smoothed.
- To compare the method with Discrete Sine Transform (DST) and the Sparse Gaussian Elimination (SGE) solver in solving Elliptic Differential Equations , i.e, look at the computational time take to solve the problem in question.
- To validate the method results with a DST solver.

2 Literature

In this section, we explored the various research works on muiltgrid by scholars in the field. Many have concentrated on convergence studies, for instance, Fedorenko [5], not only provided the first convergence analysis in history but also suggested a combination of (1). smoothing rough error components using direct methods, (2). approximating smooth error on coarser grids, and applying recursively (1) and (2) on an array of coarser grids. Doing this resulted in a complete multigrid cycle.

The rigorous convergence, based on the sum of splitting techniques was first provided and proved by Bakhvalov[1]. He showed that we can obtain approximate solutions with full multigrid (nested iteration technique process) with optimal complexity, that alter from the exact solution of the boundary value problem in the order of discretization error.

The introduction of nonlinear multigrid methods and other essential contributions like systematic application of full multigrid (FMG) were made by Brandt [2],furthermore, he showed multigrid methods actual efficiency.

3 Formulation

In this section, we explore how and why the multigrid method works in solving elliptic partial differential equations, other methods: SPE, DST to compare with MG have also been introduced. Different smothers used in the MG solver and Fourier analysis of the two grid operators are also defined in this section.

3.1 Multigrid (MG)

MG is one of the fastest method used in solving elliptic PDEs and it has been applied in combination with most of the discretization techniques. Its generally treatment of the arbitrally regions and boundary conditions, and independent of separability of the equations combined with other special properties of the equation, makes it different from other methods.

Consider using damped Jacobi as a smother (smothers will be described later in this chapter), so after smoothing with e.g. three iterations, we find out that the high-frequency components of the error have already significantly decayed, however, convergence slows down due to low-frequency components. But since we have the error smother already, the remaining part of the problem is shifted to the coarser grid.

The main idea behind multigrid is to switch between a coarser and finer grid to estimate the remaining smoothed error. This is a good approach because it's cheap iterating on a coarser grid than further continuing on the original grid. Even though this might not be very useful, but the convergence rate of most components of the error is greatly improved on shifting them on the coarser grid.

The basic recursive multigrid algorithm on level l is illustrated below, with S, an iteration operator, K is a tridiagonal matrix;

```
**Multigrid method** MGM($l, u, f$)
  if $l = 0$ then
    $u := K_0^{-1} f$ // exact solve on coarsest level
  else
    $u := S_l^{\nu_1}(u, f)$; //presmoothing step
    $d := I_l^{l-1}(K_l u - f)$; //restriction of the defect
    $v := 0$; //starting value
    for $j := 1, \dots, \gamma$, do
      MGM($l-1, v, d$) // solve $K_{l-1} v_{l-1} = d_{l-1}$ by applying $\gamma$ step of MGM($l-1, v, d$)
    end for
    $u := u - I_{l-1}^{l} v$ //update on level $l$
    $u := S_l^{\nu_2}(u, f)$ //postsmoothing step
  end if
```

Figure 1: MG algorithm [10]

Figure 2 shows a v-cycle, this is implemented in the multigrid algorithm, begining on the finnest grid, recursing down to the coarsest grid and then back up as the figure depicts.

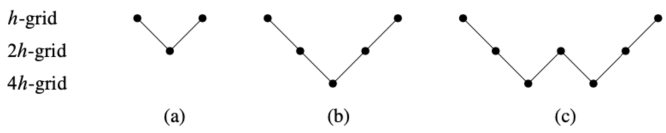


Figure 2: (a) and (b) respectively are One V-cycle with two and three levels. (c) One W-cycle with three levels [9].

Taking a single V-cycle results in a significant reduction in the magnitude of the error, however, taking more than one v-cycle may be required to converge to suficiently accurate solution[9]. The W-cycle is obtained by taking two stages at each cycle of the finner grids.

Using many v-cycles to obtain a better solution may be overcome by the full multigrid (FMG) algorithm which instead begins from the coarsest grid to the finest grids, and previous research has shown that it gives a more accurate solution even with a single v-cycle. For instance, Scott R. et ([6]) reviewed techniques of multigrid (MG) methods, where he concentrated on the role of MG as fast solvers for elliptic boundary-value problems using FMG.

FMG algorithm works as follows, these steps are replicated in figure 3;

- Discretization of the problem.
- Solved on the coarsest level using a few iterations of any solver.
- A good initial guess is obtained by interpolating the approximate solution to the next finer grid.
- Then use the 2-level MG algorithm to solve the problem at this point.
- Results obtained are further interpolated to the next level finer grid to obtain another good initial data their, and so on.

The process above is done till wen when we reach the finest grid with a very good initial guess to start the MG process described above.

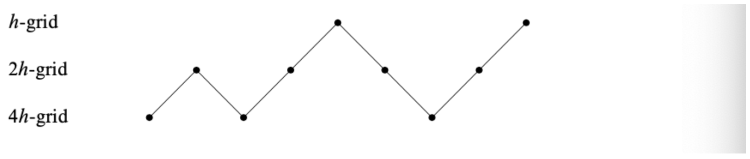


Figure 3: FMG with one V-cycle on three levels

MG solves elliptic PDEs to a given accuracy in a number of operations proportional to the number of unknowns i.e., it scales linearly with the number of discrete nodes used.
The computational cost associated with using multigrid method as a solver is O(N), however there is an exception to O(N), since the W-cycle multigrid uses O(Nlog(N) time to solve a 1D problem.

3.1.1 Smothers

In MG algorithm smothers act as the central components that determine the performance of the algorithm by reducing high frequency errors. The two smothers: Damped Jacobi and red-black are the only one we implemented in the MG algorithm to perform this study.

3.1.2 Damped Jacobi

Damped Jacobi described in equation (2) is Jacobi method is with a relaxation parameter ω. It converges faster for high frequency.

$$u^{(m+1)} = (I - \omega D^{-1} A)u^{(m)} + \omega D^{-1} b, \qquad \omega \in (0, 1] \tag{2}$$

where $D = diag(A)$ with A the coefficient matrix of the problem. The error decay is given by the iteration matrix H_ω:

$$e^{(m)} = [H_\omega]^{(m)} e^{(0)}, \qquad H_\omega = I + \frac{1}{4}\omega A \tag{3}$$

where I is an identity matrix.

3.1.3 Red-Black Gauss Seidel Method

Other than implementing the Damped Jacobi smoother, another effective alternative smother called Red-Black Gauss Seidel was also implemented in this study. This is basically the red-black ordering of grid points in a more clear way as shown in figures 4, 5a, and 5b [10]. First, even components are updated, followed by updating all odd components. Doing this we obtain the red-black Gaus Seidel stencil illustrated in figure 5b.

The red points (o) correspond to even indexed points in $1D$ [4], and points whose index sum is even in $2D$, and vice versa to black points$(\cdot)$.

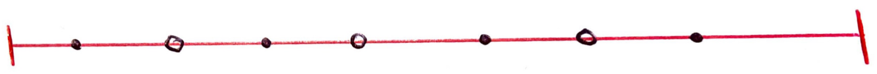
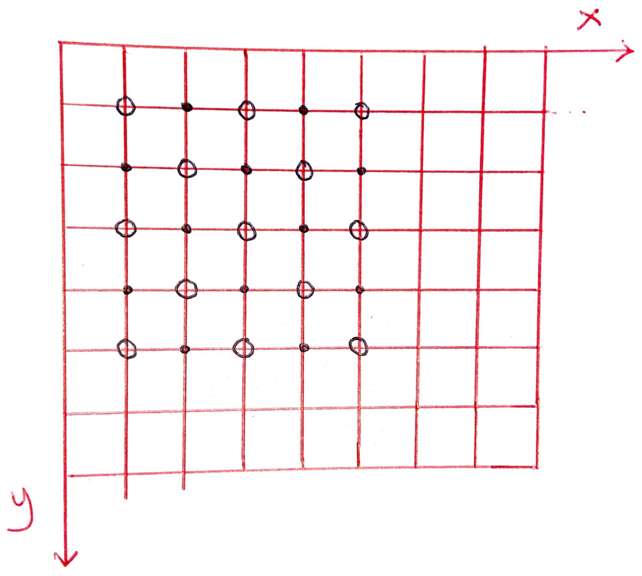
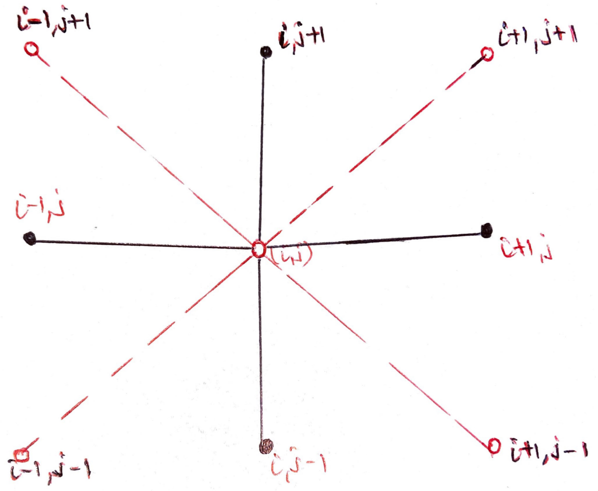


Figure 4: $1D$ grid (red (o) and black $(\cdot)$ points



(a)



(b)

Figure 5: (a) and (b) respectively show $2D$ grid and stencil for red-black with red (o) and black $(\cdot)$ points

3.2 Fourier Analysis (FA) to the two-grid operator.

FA has used to chose individual multigrid components for different situations, and it has also been applied in the calculation of convergence rates of MG methods while solving elliptic PDEs by using the exact eigenfunction of the discrete operator that is compatible with the boundary conditions [8].

The corresponding Fourier representation based on small subspaces of Fourier components for the relaxation method is possible to be calculated. Which means that the discrete fine-grid solution u_h can be given by the linear combination of of Fourier components that generate the whole space of bounded infinite grid functions.
The error before and after the i^{th} 2-grid-cycle is given by equations (4) and (5) respectively.

$$e_h^{(i)} = u_h^{(i-1)} - u_h \tag{4}$$

$$e_h^{(i)} = u_h^{(i)} - u_h \tag{5}$$

So their corresponding two-grid cycle error transformation is given by equation (6);

$$e_h^{(i)} = M_h^{2h} e_h^{(i-1)} \tag{6}$$

with M_h^{2h} is the error-transformation operator given by

$$M_h^{2h} = S_h^{\nu_2} K_h^{2h} S_h^{\nu_1} \tag{7}$$

where $K_h^{2h} = I_h - P_{2h}^h L_{2h}^{-1} R_h^{2h} L_h$ is the coarse-grid correction operator, ν_1 and ν_2 represent the number of pre- and postsmoothing iterations, I_h the G_h identity, L_{2h} the approximation of L_h on a coarse grid G_{2h}, P_{2h}^h and R_{2h}^h are transfer operators from coarse to fine grids and vice versa.

In the 2D case, the Fourier space is divided into 4D spaces which introduces us to the spaces of $2h$-harmonics($\mathcal{F}_{2h}(\theta)$) that reads

$$\mathcal{F}_{2h}(\theta) := span\{\varphi_h(\theta^{00}, .), \varphi_h(\theta^{10}, .), \varphi_h(\theta^{01}, .)\}$$

with $\theta = \theta^{00} \in \Theta_{2h} := (-\frac{\pi}{2}, \frac{\pi}{2}]^2$, $\theta^\alpha := \theta^{00} - (\alpha_1 \text{sign}(\theta_1), \alpha_2 \text{sign}(\theta_2))\pi$.
The standard coarsening ($H = 2h$) is related $2h$-harmonics in the following mannar: $\Theta_{2h} = \Theta_{low}$, where low is taken as standard coarsening. Each low fequency $\theta^{00} \in \Theta_{2h}$ is coupled with high frequencies θ^α with $\alpha \neq (00)$ during the transistion from G_h to G_{2h}.

On the coarse grid, the three high frequency components are not visible when coinciding with the corresponding low-frequency componet as given in equation (8).

$$\varphi_h(\theta^{00}, x) = \varphi_h(\theta^{11}, x) = \varphi_h(\theta^{10}, x) = \varphi_h(\theta^{01}, x) \quad \text{for} \quad x \in G_{2h} \tag{8}$$

Applying the periodicity of the exponential function, equation (8), can be verified by taking the Fourier components as coarse-grid functions as described in equation (9)

$$\varphi_h(\theta^{00}, x) = \varphi_{2h}(\mathbf{2}\theta^{00}, x) = \varphi_{2h}(\mathbf{2}\theta^\alpha, x) \text{with} \quad \theta^{00} \in \Theta_{2h},\ x \in G_{2h} \tag{9}$$

Figure 6, represents a sample set of coupled frequencies.

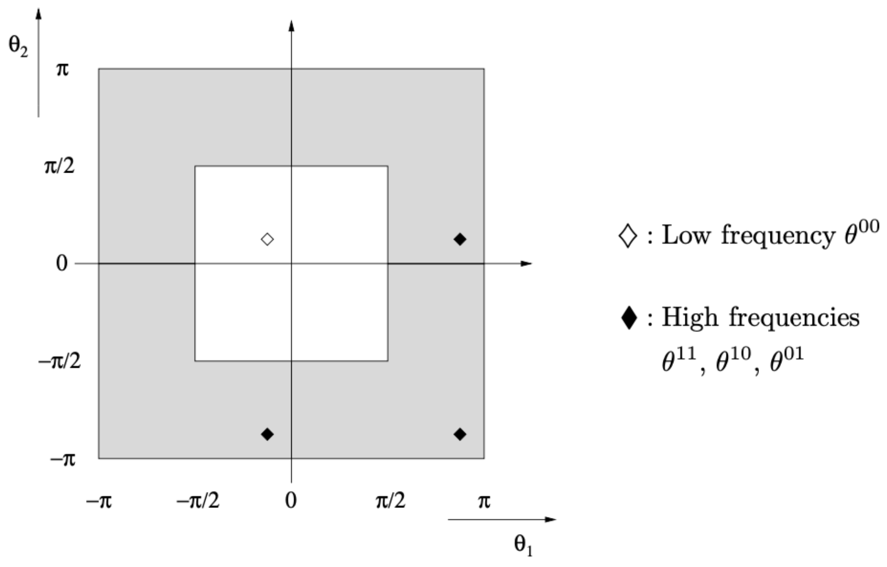

θ_2
π
$\pi/2$
0
$-\pi/2$
$-\pi$
$-\pi$
$-\pi/2$
0
$\pi/2$
π
θ_1
◇ : Low frequency θ^{00}
◆ : High frequencies $\theta^{11}, \theta^{10}, \theta^{01}$

Figure 6: High and low Fourier frequencies generating a space of 2h- harmonics. Standard coarsening, d = 2. [11]

The coarse-grid correction ($K_h^{2h} = I_h - P_{2h}^h L_{2h}^{-1} R_h^{2h} L_h$) leaves the $2h$-hamonics invariant space generating a simple (4×4) block diagonal representation of K_h^{2h} that reads

$$K_h^{2h}|_{\mathcal{F}_{2h}(\theta)} := K^{2g}(\theta) \quad (\theta \in \Theta_{2h}).$$

The two grid method coarse grid correction with respect to the $2h$-hamonics invloves Fourier representation of the different operators depending on the prescribed order of the multiindex α. The Fourier smoothing analysis of the relaxation and coarse grid correction uses the order :

$$(0,0),\ (1,1),\ (1,0),\ (0,1)$$

The Fourier components of the fine grid discretization are eigen functions for a constant coefficient operators $L_h = [l_k]_h$. So the Foureir representation of L_h with respect to the $2h$-harmonics given by eqaution (10).

$$L_{\mathcal{F}_{2h}(\theta)} = L^{2g}\theta = \begin{pmatrix} \tilde{L}_h(\theta^{00}) & 0 & 0 & 0 \\ 0 & \tilde{L}_h(\theta^{11}) & 0 & 0 \\ 0 & 0 & \tilde{L}_h(\theta^{10}) & 0 \\ 0 & 0 & 0 & \tilde{L}_h(\theta^{01}) \end{pmatrix} \in \mathbb{C}^{4\times 4} \tag{10}$$

where $\tilde{L}_h(\theta^\alpha)$ are Fourier symbols given by

$$\tilde{L}_h(\theta^\alpha) = \sum_{\mathcal{K}\in J} l_{\mathcal{K}} \exp(i\mathcal{K}\theta^\theta)$$

The coarse grid discretisation has two types of approximations: Discretization coarse grid approximation (DCA) and the Galerkin coarse grid approximation (GCA). This grid discretization L_{2h} is given by:

$$L_{2h} = \tilde{R}_h^{2h} L_h \tilde{P}_{2h}^h$$

where $\tilde{R}_h^{2h}$ and $\tilde{P}_{2h}^h$ are the Galerkin transfer operators.

3.3 Discrete Sine Transform (DST)

This solver has been used in comparing and validating our results. it uses a signal at a discrete domain, and at the two ends of the arraydifferent variants corresponding to almost different odd/even boundary conditions. Futhermore the solver expresses a function interms of a sinusoids with diffrernt frequencies and amplitudes implying different boundary conditions and an odd extension of the original signal [7].

3.4 Sparse Gaussian Elimination (SGE)

This solver uses sparse matrix (only a small fraction of the matrix elements are non zeros) capabilities to form D2x and D2y sparse matrices (see the appendix for more details). In this solver, pointers of some kind are explored to track the non zero entries, however costs are incured in storing and maniplating these pointers. Initially zero elements that don't fill-in are neither stored nor used as operands but maximum advantage of these enntries is used by SGE in saving time and storage by not maniplating and storing these elements [3].

3.5 Case Study

The above solvers are used to solve the following $2D$ Poison problem (11), and the solutions are displayed in the next section(Results). The $2D$ elliptic problem is solved over the domain [a,b]x[a,b] with $a = 0$ and $b = 1$, grid spacing $dx = dy = h = \frac{b-a}{m+1}$, with $m = 2^{k-1}$ where k $=$7 for this study. As a matter of fact, we used k = 7, since the method we are comparing with cant exceed that due to insufficient resources13. But Multigrid shoots more that, and the large the k the greater the accuracy.

$$\nabla^2 u = f(x, y) \tag{11}$$

where

$$f(x, y) = 10\pi^2(1 + \cos(4\pi(x + 2y))) - 2 * \sin(2\pi(x + 2y))e^{sin(2\pi(x+2y))}$$

with Boundary conditions.

$$u(x, y) = g(x, y)$$

where

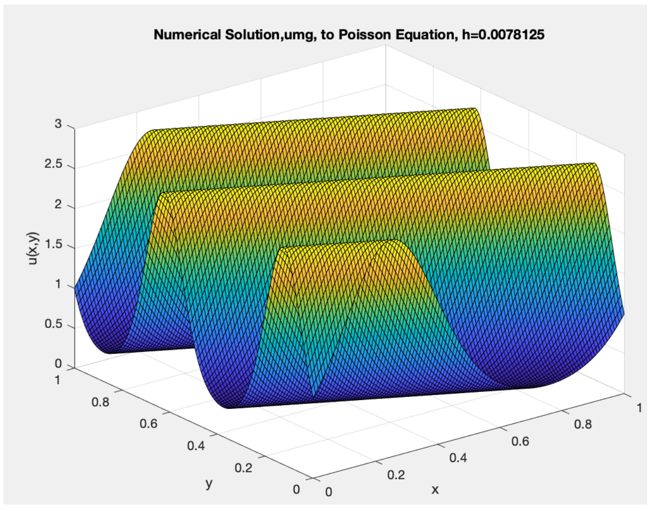
$$g(x, y) = e^{sin(2\pi(x+2y))}$$

The exact solution used to validate results and compute the error is given by equation (12).

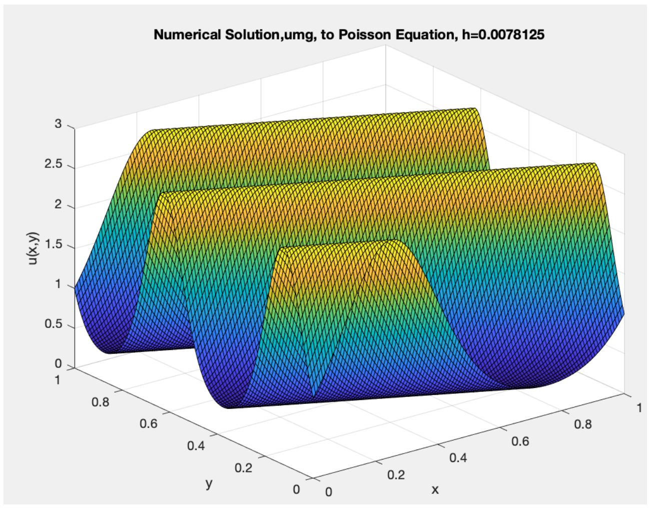
$$ue(x, y) = g(x, y) \tag{12}$$

4 Results

Figure 7a and 7b show the respective solutions obtained by using the two smothers: damped Gaus Jacobi and Red-black Gaus seidel.



(a)

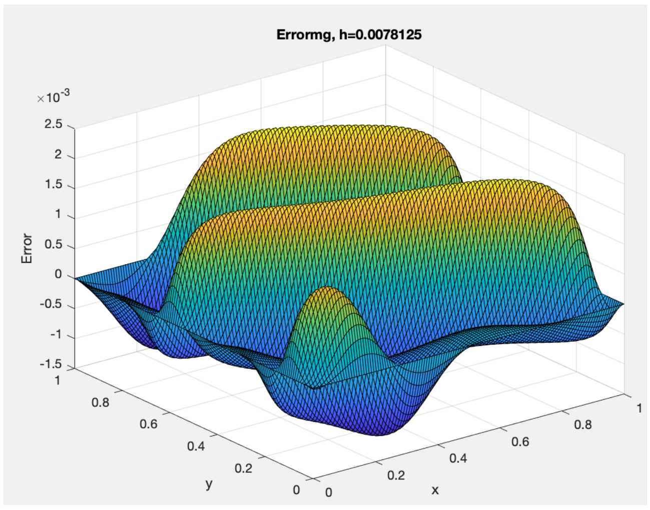


(b)

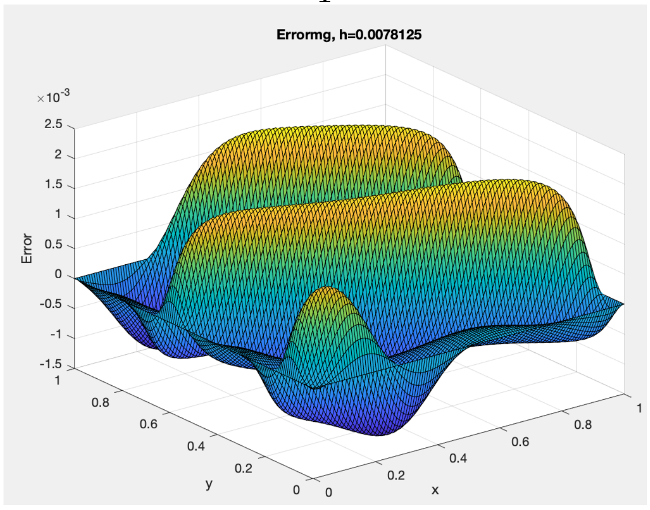
Figure 7: (a) and (b) respectively are solutions obtained using damped jacobi and Red-black smothers.

Figure 8a and 8b show the respective errors obtained by using the two smothers: damped Gaus Jacobi and Red-black Gaus seidel.

T



(a)

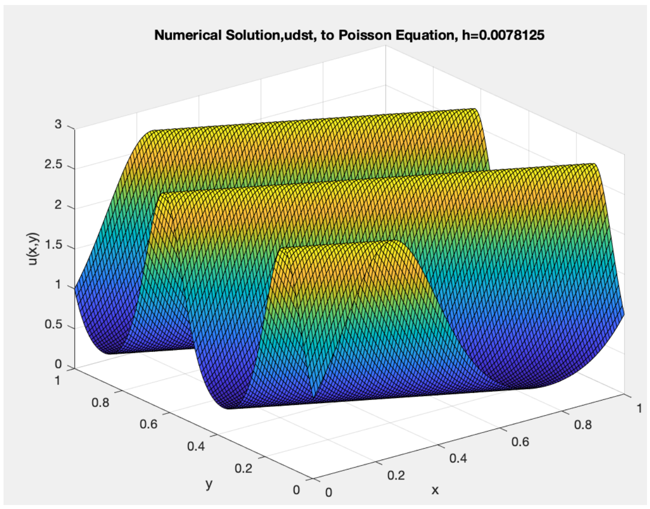


(b)

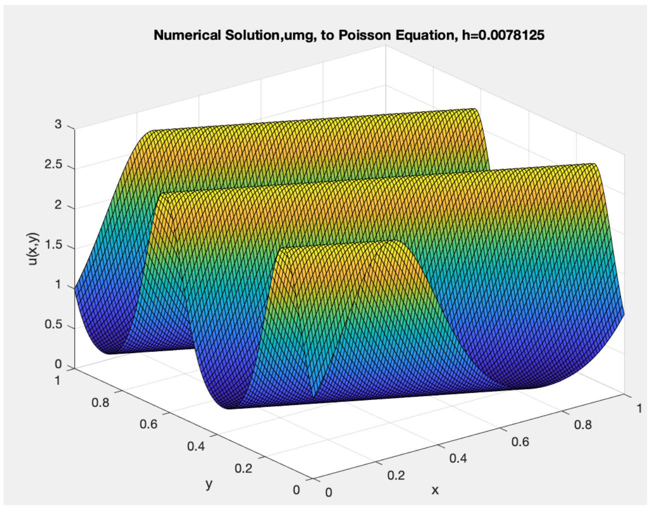
Figure 8: (a) and (b) respectively are errors obtained using damped jacobi and Red-black smothers.

4.1 Comparison and Validation of Results

The obtained solution for MG is validated against the DST solver result as shown in figures 9a and 9b. According to figures 10a and 10b, these error plots depict that the results obatined are realistic, since both have an upper bound error of magnitude 10^{-3}.

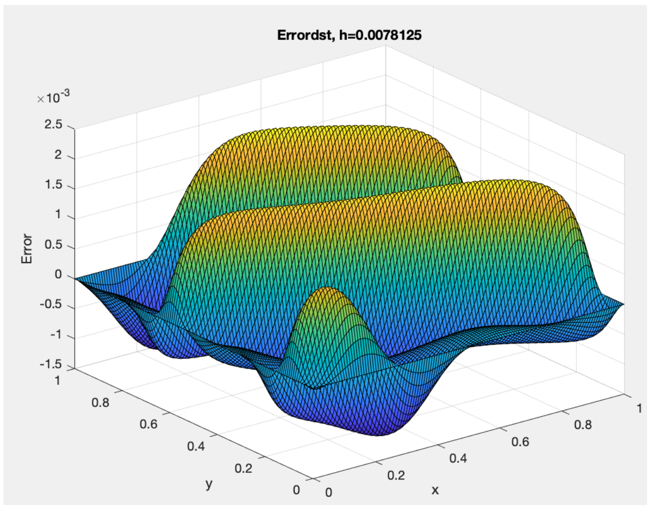


(a)

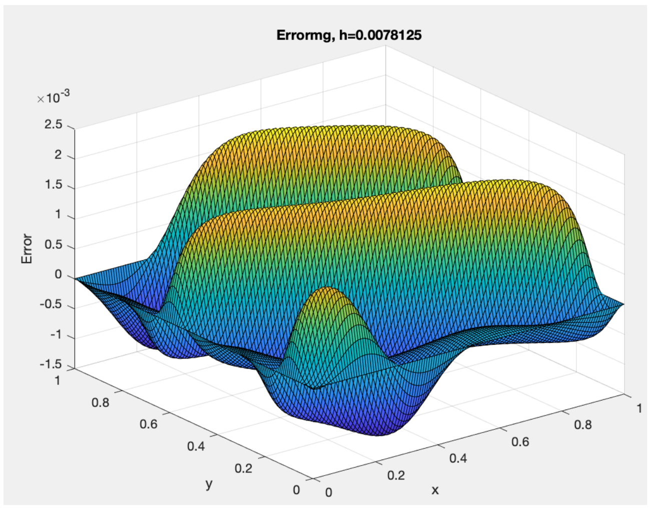


(b)

Figure 9: (a) and (b) respectively are solutions obtained using DST and Multigrid solver with damped Jacobi smother.



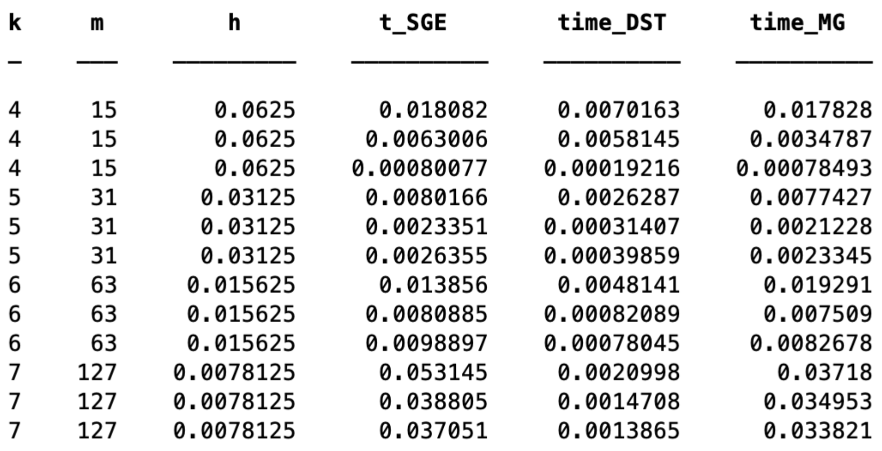
(a)



(b)

Figure 10: (a) and (b) respectively are errors obtained using DST and Multigrid solver with damped Jacobi smother.

A computational time cost check was performed, as a comparision with SGE, DST, and MG solvers on three different runs for $k = 4, 5, 6, 7$, and results are depicted in table 11. The results displayed in table 11, look to be random, but the trend shows that run time decreases as h increases for DST, but for MG and SGE, appears to be the reverse. Thats why we hard to take the mean time to obtain the best minimum wall time for each run as displayed in table 12.

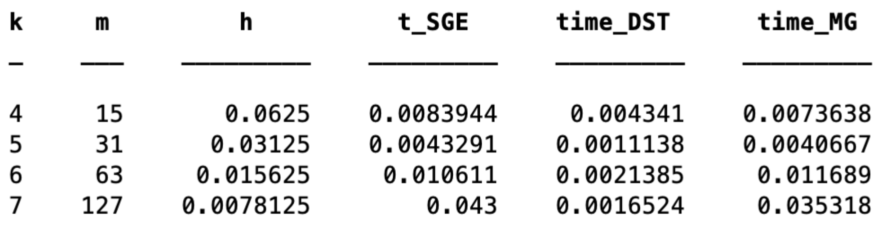


```
k     m        h            t_SGE         time_DST       time_MG
_    ___    _________    __________    __________    __________

4     15       0.0625      0.018082     0.0070163      0.017828
4     15       0.0625     0.0063006     0.0058145     0.0034787
4     15       0.0625    0.00080077    0.00019216    0.00078493
5     31      0.03125     0.0080166     0.0026287     0.0077427
5     31      0.03125     0.0023351    0.00031407     0.0021228
5     31      0.03125     0.0026355    0.00039859     0.0023345
6     63     0.015625      0.013856     0.0048141      0.019291
6     63     0.015625     0.0080885    0.00082089      0.007509
6     63     0.015625     0.0098897    0.00078045     0.0082678
7    127    0.0078125      0.053145     0.0020998       0.03718
7    127    0.0078125      0.038805     0.0014708      0.034953
7    127    0.0078125      0.037051     0.0013865      0.033821
```

Figure 11: Wall time at each run

According to the computed mean wall clock time shown in Table 12, DST appears to be the best since it has the lowest computation time amongest all other method as m increases followed by MG (using damped Jacobi smoother). However SGE also compited with MG due to the use of the sparse matrix that quickens the computations.

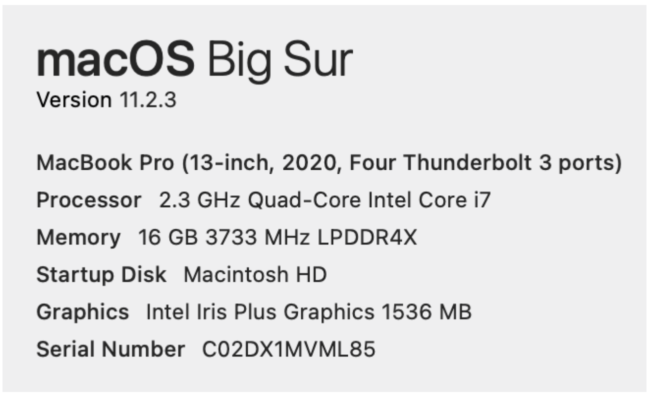


```
k     m        h           t_SGE        time_DST       time_MG
_    ___    _________    _________    _________    _________

4     15       0.0625    0.0083944     0.004341    0.0073638
5     31      0.03125    0.0043291    0.0011138    0.0040667
6     63     0.015625     0.010611    0.0021385     0.011689
7    127    0.0078125        0.043    0.0016524     0.035318
```

Figure 12: Mean wall time

Note: I used only k values from 4 to 5, because when i tried to run for $k = 8$ and above the MATLAB on my computer terminated for SGE, so i wouldnot perform any further simulations beyond $k = 7$. But for MG and DST i was able to do simulations for $k \geq 7$.
The simulations were perfomed on a computer with these the following specifications as shown in figure 13. Its concluded that simulations depend much on different systems specifications.



macOS Big Sur
Version 11.2.3

MacBook Pro (13-inch, 2020, Four Thunderbolt 3 ports)
Processor 2.3 GHz Quad-Core Intel Core i7
Memory 16 GB 3733 MHz LPDDR4X
Startup Disk Macintosh HD
Graphics Intel Iris Plus Graphics 1536 MB
Serial Number C02DX1MVML85

Figure 13: Computer specifications

5 Conclusion

This section discusses about the conclusions deduced from the above methods used. All the objectives have been well implemeted and achived.

We used the Discrete Sine Transform (DST) to compare and validate our solution. And according to figures 9a, 9b, 10a, and 10b, we were able to validate our solution since the results dipict the same solution and magnitude of the error (10^{-3}). However using bigger k values we obtain better results and better error tolerance much smaller than (10^{-3}). But we were restricted to use $k \geq 8$ since SGE couldn't perform for $k \geq 8$, so for a fair comparison i had to use $k < 8$.

According to table 12, overall, DST wins in terms of cost, however the mean wall time for MG, is similar to the two methods we are validating with, hence the results obtained from using MG to solve the same problem are realistic.

In a case where the matrix of the original equation is symmetric positive definite (SPD), then a multgrid method with a forced reduced tolerance may be used as a preconditioner, however, the SPD constraint makes construction of the preconditioner complicated.

References

[1] Nikolai Sergeevich Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Computational Mathematics and Mathematical Physics*, 6(5):101–135, 1966.

[2] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[3] Robert K Brayton, Fred G Gustavson, and Ralph A Willoughby. Some results on sparse matrices. *Mathematics of Computation*, 24(112):937–954, 1970.

[4] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.

[5] Radii Petrovich Fedorenko. The speed of convergence of one iterative process. *USSR Computational Mathematics and Mathematical Physics*, 4(3):227–235, 1964.

[6] Scott R Fulton, Paul E Ciesielski, and Wayne H Schubert. Multigrid methods for elliptic problems: A review. *Monthly Weather Review*, 114(5):943–959, 1986.

[7] A Gupta and K Raghava Rao. A fast recursive algorithm for the discrete sine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(3):553–557, 1990.

[8] C-C Jay Kuo and Bernard C Levy. Two-color fourier analysis of the multigrid method with red-black gauss-seidel smoothing. *Applied Mathematics and Computation*, 29(1):69–87, 1989.

[9] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[10] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[11] Roman Wienands and Wolfgang Joppich. *Practical Fourier analysis for multigrid methods*. CRC press, 2004.

6 Appendix

6.1 MG solver

```
% ===============================================================================
% Author      : Prof. Grady wright
% modified by : Brian Kyanjo
% Class       : MATH566
% Date        : March 18th, 2021
%
% Numerical approximation to Poisson’s equation over the square [a,b]x[a,b] with
% Dirichlet boundary conditions.  Uses a uniform mesh with (n+2)x(n+2) total
% points (i.e, n interior grid points).
%
% Solves with classical multigrid (v-cycle and damped-Jacobi, and red-black Gaus
% Seidel smoother).
%
% Input:
%     ffun : the RHS of poisson equation (i.e. the Laplacian of u).
%     gfun : the boundary function representing the Dirichlet B.C.
%      a,b : the interval defining the square
%        m : m+2 is the number of points in either direction of the mesh.
% smoother : Selection of the smoother.
% Ouput:
%        u : the numerical solution of Poisson equation at the mesh points.
%      x,y : the uniform mesh.
%
% ===============================================================================
function [u,x,y] = multigrid_solver(ffun,gfun,a,b,m)

k = round(log2(m+1));

if abs(log2(m+1)-k) > 100*eps
error('multigrid_solver:power2','m+1 must be a power of 2');
end

h = (b-a)/2^k;   % Mesh spacing

[x,y] = meshgrid(a:h:b);   % Uniform mesh, including boundary points.

idx = 2:m+1;
idy = 2:m+1;

% Compute boundary terms, south, north, east, west
ubs = feval(gfun,x(1,1:m+2),y(1,1:m+2));     % Include corners
ubn = feval(gfun,x(m+2,1:m+2),y(m+2,1:m+2)); % Include corners
ube = feval(gfun,x(idy,m+2),y(idy,m+2));     % No corners
ubw = feval(gfun,x(idy,1),y(idy,1));         % No corners

% Evaluate the RHS of Poisson’s equation at the interior points.
f = feval(ffun,x,y);
```

```
% Inital guess
u = zeros(m+2,m+2);

% Add boundary terms to u
u(idx,1) = ubw; u(idx,m+2) = ube;
u(1,:) = ubs; u(m+2,:) = ubn;

tol = 1e-8;
r = inf;
nrmf = norm(f(:));
while norm(r(:)) > tol*nrmf
[u,r] = vcycle(u,f,h,k);
end

end

function [u,r] = vcycle(u,f,h,k)

numSmooths = 3;
n = 2^k; n2 = n/2;
i = 2:n; i2 = 2:n2;

if k == 1
% Only one interior point so solve the system:
u(2,2) = 0.25*(u(1,2)+u(3,2)+u(2,1)+u(2,3)-h^2*f(2,2));
return;
else
% Smooth the solution.
%Choice of the smoother
smoother = 2;  %1 for Damped Jacobi smoother
%2 for red-black-Gaus smoother

if smoother == 1
u = dampedJacobi(u,f,h,k,numSmooths);
elseif  smoother == 2
u = redblack(u,f,h,k,numSmooths);
end

% Calculate residual
r = zeros(n+1);
r(i,i) = f(i,i) - (-4*u(i,i)+u(i-1,i)+u(i+1,i)+u(i,i-1)+u(i,i+1))/h^2;

% Restrict the residual to the coarser grid using full-weighting
jj = 3:2:n-1;
f2 = zeros(n2+1);
f2(i2,i2) = (r(jj-1,jj)+r(jj+1,jj)+r(jj,jj-1)+r(jj,jj+1)+4*r(jj,jj))/8;

% Repeat call to vcylce with the residual on coarser grid and with a
% zero initial guess.
```

```
u2 = vcycle(zeros(n2+1),f2,2*h,k-1);

% Interpolate (prolongate) the correction to the finer grid
ut = zeros(n+1);
ut(1:2:n+1,1:2:n+1) = u2;
ut(2:2:n,1:2:n+1) = 0.5*(u2(1:n2,:)+u2(2:n2+1,:));
ut(:,2:2:n+1) = 0.5*(ut(:,1:2:n-1)+ut(:,3:2:n+1));

% Update the solution and smooth again:
if smoother == 1
u = dampedJacobi(u+ut,f,h,k,numSmooths);
elseif  smoother == 2
u = redblack(u+ut,f,h,k,numSmooths);
end

% Calculate residual
r(i,i) = f(i,i) - (-4*u(i,i)+u(i-1,i)+u(i+1,i)+u(i,i-1)+u(i,i+1))/h^2;
end
end

%Damped Jacobi smoother
function u = dampedJacobi(u,f,h,k,numSmooths)
omega = 4/5;
n = 2^k;
i = 2:n;
f = h^2*f;
% Smooth u using underrelaxed Jacobi
for j=1:numSmooths
u(i,i) = (0.25*omega)*(u(i-1,i)+u(i+1,i)+u(i,i-1)+u(i,i+1)-f(i,i)) + ...
(1-omega)*u(i,i);
end

end

%Red black Gauss-Seidel step
function u = redblack(u,f,h,k,numSmooths)
n = 2^k;
f = h^2*f;
for k = 1:numSmooths
%Red points
for i = 2:n
for j = 2:n
if (mod(i+j,2) == 0) %sum of indices is even
u(i,j) = 0.25*(u(i-1,j)+u(i+1,j)...
+u(i,j-1)+u(i,j+1) - f(i,j));
end
end
end

%Black points
```

```
for i = 2:n
for j = 2:n
if (mod(i+j,2) == 1) %sum of indices is odd
u(i,j) = 0.25*(u(i-1,j)+u(i+1,j)...
+u(i,j-1)+u(i,j+1) - f(i,j));
end
end
end
end
end
```

6.2 DST Solver

```
% ================================================================================
% Author      : Prof. Grady wright
% modified by : Brian Kyanjo
% Class       : MATH566
% Date        : March 18th, 2021
%
% Numerical approximation to Poisson's equation over the square [a,b]x[a,b] with
% Dirichlet boundary conditions.  Uses a uniform mesh with (n+2)x(n+2) total
% points (i.e, n interior grid points).
%
% Solves with the DST.
%
% Input:
%     pfun : the RHS of poisson equation (i.e. the Laplacian of u).
%     bfun : the boundary function representing the Dirichlet B.C.
%      a,b : the interval defining the square
%        m : m+2 is the number of points in either direction of the mesh.
% Ouput:
%        u : the numerical solution of Poisson equation at the mesh points.
%      x,y : the uniform mesh.
% ================================================================================

function [u,x,y] = dst_solver(pfun,bfun,a,b,m)

h = (b-a)/(m+1);   % Mesh spacing

[x,y] = meshgrid(a:h:b);   % Uniform mesh, including boundary points.

idx = 2:m+1;
idy = 2:m+1;

% Compute boundary terms, south, north, east, west
ubs = feval(bfun,x(1,1:m+2),y(1,1:m+2));     % Include corners
ubn = feval(bfun,x(m+2,1:m+2),y(m+2,1:m+2)); % Include corners
ube = feval(bfun,x(idy,m+2),y(idy,m+2));     % No corners
ubw = feval(bfun,x(idy,1),y(idy,1));         % No corners
```

```
% Evaluate the RHS of Poisson's equation at the interior points.
f = feval(pfun,x(idy,idx),y(idy,idx));

% Adjust f for boundary terms
f(:,1) = f(:,1) - ubw/h^2;              % West
f(:,m) = f(:,m) - ube/h^2;              % East
f(1,1:m) = f(1,1:m) - ubs(idx)/h^2;     % South
f(m,1:m) = f(m,1:m) - ubn(idx)/h^2;     % North

% Computation of fhat=(S*f)*S^(-1), where S is the discrete sine transform.
fhat = idst(dst(f,1),2);

% Denominator for the computation of uhat:
denom = bsxfun(@plus,cos(pi*(idx-1)./(m+1)).',cos(pi*(idx-1)./(m+1)))-2;

uhat = h^2/2*(fhat./denom);

% Computation of u = (S^(-1)*uhat)*S
u = dst(idst(uhat,1),2);

% Append on to u the boundary values from the Dirichlet condition.
u = [ubs;[ubw,u,ube];ubn];

end
```

6.3 SGE Solver

```
% ================================================================================
% Author      : Prof. Grady wright
% modified by : Brian Kyanjo
% Class       : MATH566
% Date        : March 18th, 2021
%
% Numerical approximation to Poisson's equation over the square [a,b]x[a,b] with
% Dirichlet boundary conditions.  Uses a uniform mesh with (n+2)x(n+2) total
% points (i.e, n interior grid points).
% Input:
%     ffun : the RHS of poisson equation (i.e. the Laplacian of u).
%     gfun : the boundary function representing the Dirichlet B.C.
%      a,b : the interval defining the square
%        m : m+2 is the number of points in either direction of the mesh.
% Ouput:
%        u : the numerical solution of Poisson equation at the mesh points.
%      x,y : the uniform mesh.
% ================================================================================

function [u,x,y] = sge_solver(ffun,gfun,a,b,m)

h = (b-a)/(m+1);   % Mesh spacing
```

```
[x,y] = meshgrid(a:h:b);   % Uniform mesh, including boundary points.

idx = 2:m+1;
idy = 2:m+1;

% Compute boundary terms, south, north, east, west
ubs = feval(gfun,x(1,1:m+2),y(1,1:m+2));     % Include corners
ubn = feval(gfun,x(m+2,1:m+2),y(m+2,1:m+2)); % Include corners
ube = feval(gfun,x(idy,m+2),y(idy,m+2));     % No corners
ubw = feval(gfun,x(idy,1),y(idy,1));         % No corners

% Evaluate the RHS of Poisson's equation at the interior points.
f = feval(ffun,x(idy,idx),y(idy,idx));

% Adjust f for boundary terms
f(:,1) = f(:,1) - ubw/h^2;             % West
f(:,m) = f(:,m) - ube/h^2;             % East
f(1,1:m) = f(1,1:m) - ubs(idx)/h^2;    % South
f(m,1:m) = f(m,1:m) - ubn(idx)/h^2;    % North

f = reshape(f,m*m,1);

%Using sparse matrix capabilities to form D2x and D2y matrices
I = eye(m);
e = ones(m,1);
e1 = zeros(m,1);

%D2x
T = spdiags([e1 -2*e1 e1],[-1 0 1],m,m);
S = spdiags([e e],[-1 1],m,m);
D2x = (1/h^2)*(kron(I, T) + kron(S,I));

%D2y
Ty = spdiags([e -2*e e],[-1 0 1],m,m);
Sy = spdiags([e1 e1],[-1 1],m,m);
D2y = (1/h^2)*(kron(I, Ty) + kron(Sy,I));

% Solve the system
u = (D2x + D2y)\f;

% Convert u from a column vector to a matrix to make it easier to work with
% for plotting.
u = reshape(u,m,m);

% Append on to u the boundary values from the Dirichlet condition.
u = [ubs;[ubw,u,ube];ubn];

end
```

6.4 Main

```
%=========================================================================
% Author        : Brian Kyanjo
% Supervised by : Prof. Grady wright
% Class         : MATH566
% Date          : March 18th, 2021
%
% Main code for calling all necessary functions. And the sample fuctions to
% be used are defined here.
% Multigrid and DST solvers are used to solve the problem in question.
% The wall time and mean wall time for each method in 3 runs has been
% performed.
%=========================================================================

clear all;
close all;

% Script for testing fd2poisson over the square [a,b]x[a,b]
a = 0; b = 1;

% Laplacian(u) = f
f = @(x,y) 10*pi^2*(1+cos(4*pi*(x+2*y))-2*sin(2*pi*(x+2*y))).*exp(sin(2*pi*(x+2*y)));
% u = g on Boundary
g = @(x,y) exp(sin(2*pi*(x+2*y)));

%Exact solution is g.
uexact = @(x,y) g(x,y);

% Compute and time the solution
k1 = zeros(1,4); h1 = zeros(1,4); m1 = zeros(1,4);
t = zeros(1,4);
t_dst = zeros(1,4);
t_mg = zeros(1,4);

t1 = [];
tdst = [];
tmg = [];
for ii=1:3
for k=4:7
%k=10;

k1(k-3) = k;
m1(k-3) = 2^k-1;
m = 2^k-1;
h1(k-3) = (b-a)/(m+1);
h = (b-a)/(m+1);
```

```
tic
[u,x,y] = sge_solver(f,g,a,b,m);
gedirect = toc;
t(k-3) = gedirect;

tic
[udst,x,y] = dst_solver(f,g,a,b,m);
gedirect = toc;
t_dst(k-3) = gedirect;

tic
[umg,x,y] = multigrid_solver(f,g,a,b,m);
gedirect = toc;
t_mg(k-3) = gedirect;
end
t1 = [t1,t];
tdst = [tdst, t_dst];
tmg = [tmg,t_mg];
end

%k=4
c4=[t1(1);t1(5);t1(9)]';
fd4=[tdst(1);tdst(5);tdst(9)]';
h4=[tmg(1);tmg(5);tmg(9)]';

%k=5
c5=[t1(2);t1(6);t1(10)]';
fd5=[tdst(2);tdst(6);tdst(10)]';
h5=[tmg(2);tmg(6);tmg(10)]';

%k=6
c6=[t1(3);t1(7);t1(11)]';
fd6=[tdst(3);tdst(7);tdst(11)]';
h6=[tmg(3);tmg(7);tmg(11)]';

%k=7
c7=[t1(4);t1(8);t1(12)]';
fd7=[tdst(4);tdst(8);tdst(12)]';
h7=[tmg(4);tmg(8);tmg(12)]';

k4 = [k1(1);k1(1);k1(1)];
m4 = [m1(1);m1(1);m1(1)];
h_4 = [h1(1);h1(1);h1(1)];

%Table showing timing results of each method and for each value of m.
Table4 = table(k4,m4,h_4,c4(:),fd4(:),h4(:), 'VariableNames',{'k','m','h','t_SGE','time_

k5 = [k1(2);k1(2);k1(2)];
m5 = [m1(2);m1(2);m1(2)];
h_5 = [h1(2);h1(2);h1(2)];
```

```
%Table showing timing results of each method and for each value of m.
Table5 = table(k5,m5,h_5,c5(:),fd5(:),h5(:), 'VariableNames',{'k','m','h','t_SGE','time_

k6 = [k1(3);k1(3);k1(3)];
m6 = [m1(3);m1(3);m1(3)];
h_6 = [h1(3);h1(3);h1(3)];
%Table showing timing results of each method and for each value of m.
Table6 = table(k6,m6,h_6,c6(:),fd6(:),h6(:), 'VariableNames',{'k','m','h','t_SGE','time_

k7 = [k1(4);k1(4);k1(4)];
m7 = [m1(4);m1(4);m1(4)];
h_7 = [h1(4);h1(4);h1(4)];
%Table showing timing results of each method and for each value of m.
Table7 = table(k7,m7,h_7,c7(:),fd7(:),h7(:), 'VariableNames',{'k','m','h','t_SGE','time_


%General Table
Table = [Table4; Table5; Table6; Table7]


%mean
Tablem4 = table(k1(1),m1(1),h1(1),mean(c4),mean(fd4),mean(h4), 'VariableNames',{'k','m',
Tablem5 = table(k1(2),m1(2),h1(2),mean(c5),mean(fd5),mean(h5), 'VariableNames',{'k','m',
Tablem6 = table(k1(3),m1(3),h1(3),mean(c6),mean(fd6),mean(h6), 'VariableNames',{'k','m',
Tablem7 = table(k1(4),m1(4),h1(4),mean(c7),mean(fd7),mean(h7), 'VariableNames',{'k','m',

Table_mean = [Tablem4; Tablem5; Tablem6; Tablem7]

%
% % Plot solution
% figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
% surf(x,y,udst), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
% title(strcat('Numerical Solution,udst, to Poisson Equation, h=',num2str(h)));
%
% % Plot error
% figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
% surf(x,y,udst-uexact(x,y)),xlabel('x'),ylabel('y'), zlabel('Error'),
% title(strcat('Errordst, h=',num2str(h)));


% %Plot solution
% figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
% surf(x,y,umg), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
% title(strcat('Numerical Solution,umg, to Poisson Equation, h=',num2str(h)));
%
% %Plot error
% figure, set(gcf,'DefaultAxesFontSize',10,'PaperPosition', [0 0 3.5 3.5]),
% surf(x,y,umg-uexact(x,y)),xlabel('x'),ylabel('y'), zlabel('Error'),
% title(strcat('Errormg, h=',num2str(h)));
```