ESAM445 HW3 Computing Report

Mingfu Liang, Student ID:3146919, NetID:MLQ4767

June 2019

1. Introduction and compute the $f(x)$ by twice differentiation

In this computing report, we are going to consider the model problem

$$-u_{xx} = f(x), \quad u(-1) = u(1) = 0. \tag{1}$$

Determine the function $f(x)$ so that the exact solution is

$$u_{exact}(x) = e^{-px^2}\left(1 - x^2\right), \tag{2}$$

By differentiating the (2) twice and based on the Equation (1), we have

$$f(x) = -\frac{\partial^2\left(u_{exact}(x)\right)}{\partial x^2} = -\left(10px^2 - 2 + 4p^2x^2 - 4p^2x^4 - 2p\right)e^{-px^2}. \tag{3}$$

2. Set $p = 1$. Solve (1) using both CG and PCG

2.1 Determine a Convergence Criteria

Now we assume N interior grid points, $x_j = jh, j = 0, 1, ..., N, N+1$ where $h = 1/(N+1)$ is the grid size. Letting u_j denote the approximation to the exact solution $u(x_j)$, manipulation of the Taylor series for u around x_j gives,

$$u_{xx}(x_j) = \frac{u_{j+1} + u_{j-1} - 2u_j}{h^2} - h^2\frac{u_{xxxx}}{12} + O\left(h^4\right) \tag{4}$$

We can see from Equation (4) that the second order, central difference approximation to u_{xx} is

$$u_{xx}(x_j) \simeq \frac{u_{j+1} + u_{j-1} - 2u_j}{h^2}, \tag{5}$$

and the truncation error is

$$-h^2\frac{u_{xxxx}}{12}. \tag{6}$$

Based on the Equation 2 we have:

$$u_{xxxx} = \frac{\partial^4\left(u_{exact}(x)\right)}{\partial x^4} = -4\mathrm{e}^{-px^2}p\left(-6 + 4p^3x^4\left(-1 + x^2\right) - 4p^2x^2\left(-3 + 7x^2\right) + p\left(-3 + 39x^2\right)\right)$$

So that for Equation (1) each grid point we have the linear equation, which is the discretization of the Equation (1)

$$-\frac{u_{j+1} + u_{j-1} - 2u_j}{h^2} = f_j \tag{7}$$

where $f_j = f(x_j)$ and $j = 1...N$. Note that at $j = 0$ and $j = N + 1$ our approximation calls for value of $u_0 = u(-1) = 0$ and $u_{N+1} = u(1) = 0$. When we are using iterative method like PCG with preconditioner or unpreconditioned CG for solving the problem, in each iteration what we do know is the amount by which the current approximation $\vec{u}_{approx}$ fails to solve the equation, i.e., (omit the iterate index n here for simplicity)

$$\overrightarrow{r} = \overrightarrow{f} - A\vec{u}_{approx}, \tag{8}$$

where $\overrightarrow{r}$ is called the residual and $\overrightarrow{f}$ is a column vector contains $f_j = f(x_j)$ where $j = 1...N$, and the approximation solution $\vec{u}_{approx}$ is also a column vector contains u_j where $j = 1...N$. A is the coefficient matrix of the discretization linear system. When we are using the iterative method like PCG and CG, we are reducing the residual in each iteration. However, since we are using the second order, central difference approximation to discretize the problem, therefore the truncation error can never be reduced. What we should do is to make sure that we sufficiently reduce the residual error from the discretization linear system such that all the error come from the truncation error, which means after the convergence iteration no matter how many additional iteration we use to reduce the residual error, the actual error between the actual solution $\vec{u}_{exact}$ and the approximation $\vec{u}_{approx}$ should remain the same as the truncation error. Therefore the actual error can be written as:

$$\vec{e}_{actual} = \vec{u}_{exact} - \vec{u}_{approx} \simeq \vec{e}_{truncation} + \vec{r} \tag{9}$$

where $\vec{e}_{truncation}$ denotes the truncation error and the $\vec{r}$ denotes the residual. Since truncation error term is proportion to the h^2, therefore we should reduce the $\vec{r}$ to less than $O(h^2)$ such that all error come from the truncation error since it can not be reduced and will dominate the whole error if the residual is small enough. Now let's analysis the convergence checking $\frac{\|\overrightarrow{r_k}\|}{\|\overrightarrow{r_0}\|}$ in CG and PCG, where $\overrightarrow{r_0} = \vec{f}$. When $p = 1$ and based on Equation (3) we have:

$$u_{xxxx} = -4\mathrm{e}^{-x^2}\left(-9 + 51x^2 - 32x^4 + 4x^6\right), \tag{10}$$

$$-u_{xx} = 2\mathrm{e}^{-x^2}\left(2 - 7x^2 + 2x^4\right) = f(x). \tag{11}$$

We can see that when $p = 1$ the leading term in u_{xxxx} and $f(x)$ should be constant term, which means they are $O(1)$, and then the convergence checking should also be $O(h^2)$. Therefore to efficiently reduce the residual error we should set the convergence checking at least smaller than h^2 such that all the error come from the truncation error. So the convergence criteria I choose is $\epsilon = h^2$. To justify my choice, here I give an example of analysis for the PCG using Jacobi, SSOR as preconditioner and unpreconditioned CG when $h = 0.1$ (the result are the same when $h = 0.01, 0.001$) and I visualize how the infinite norm of the residual error $\vec{r}_k$ and actual error $\vec{e}_{actual}$ change as the iteration increases. The relationship is shown in Figure 1-3 where the blue line denote the infinite norm of residual and the orange line denote the infinite norm of the actual error vector. For instance, as shown in Figure 1 where I use Jacobi as preconditioner for PCG method, when $p = 1$ and $h = 0.1 = 10^{-1}$, the convergence criteria I set is $h^2 = 10^{-2}$ and I get the convergence when $iteration = 10 = 10^1$ as shown in the first row in Table 1. Then base on Figure 1 we can see that the infinite norm of the residual do decrease sufficient at $iteration = 10$ and when the iteration keep increasing after $iteration = 10$, the infinite norm of the actual error almost does not

change. This scenario only happens when the PCG and CG method converge sufficiently by reducing the residual error sufficiently such that all the error come from the truncation error. Since the truncation error cannot be reduce therefore the actual error remains the same as what we see empirically in Figure 1. Furthermore, as shown in Table 1-3, when the grid size h is the same, these three methods get almost the same infinite norm of actual error at convergence as shown in the third column, which also indicates that the convergence criteria can drive the residual down sufficiently and all the error come from the truncation error. I also give a visualization as shown in Figure 1-3 and we can see that the infinite norm of the actual error are almost the same for these three method at convergence. Therefore based on the empirical results shown above, convergence criteria $\epsilon = h^2$ should be a good enough choice when $p = 1$ that we can save our time to get a good enough approximation. The results and conclusions are the same when we set $p = 1$ and use different grid size $h = 0.01, 0.001$ for Jacobi, SSOR as the preconditoner for PCG method and unprecondtioned CG method.

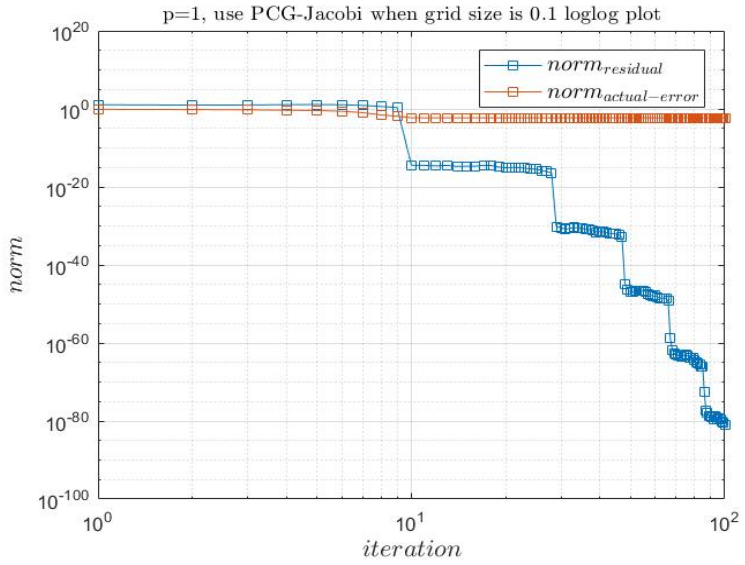

Figure 1: When $p = 1$ and $h = 0.1$, loglog plot of the iteration verse the infinite norm of the residual vector (denotes as the blue) and the actual error vector (denotes as the orange) using Jacobi as preconditioner for PCG method.

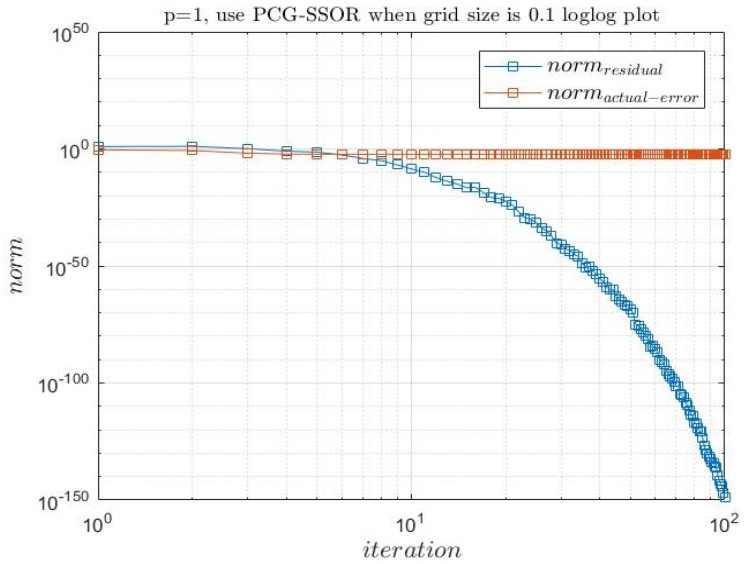

Figure 2: When $p = 1$ and $h = 0.1$, loglog plot of the iteration verse the infinite norm of the residual vector (denotes as the blue) and the actual error vector (denotes as the orange) using SSOR as preconditioner for PCG method.

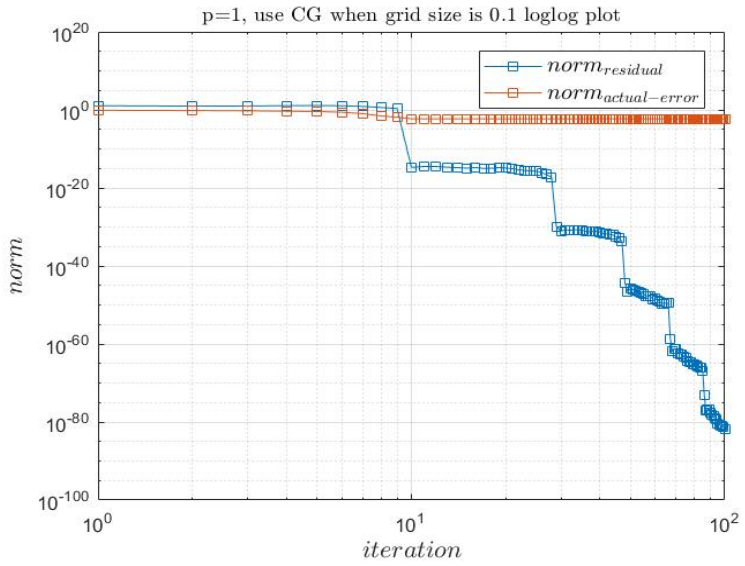

Figure 3: When $p = 1$ and $h = 0.1$, loglog plot of the iteration verse the infinite norm of the residual vector (denotes as the blue) and the actual error vector (denotes as the orange) using unpreconditioned CG.

2.2 Compare your results from PCG with unpreconditioned CG.

As shown in Table 1, 2 and 3, we can see that when using the SSOR as the precondtioner for PCG, it can converge much faster than using Jacobi as preconditioner for PCG and also faster than using the unpreconditioned CG, where grid size h is the same. For example we can see when $h = 0.01$, using the SSOR as preconditioner for PCG only need 9 iteration to converge while using the unpreconditioned CG or using the PCG with Jacobi as preconditioner need 100 iterations to converge. It means that in each

iteration, the PCG using SSOR as preconditioner reduce much more residual error than the other two methods. When the grid size h decrease, the infinite norm of actual error at convergence also decrease. This behavior is the same for all these three methods. Additionally, these three methods get almost the same infinite norm of actual error at convergence, which indicates that the convergence criteria $\epsilon = h^2$ can drive the residual down sufficiently and all the error come from the truncation error. Then we can see from the Table 1 and 3 that using the Jacobi (diagonal scaling) as the preconditioner for PCG has the same convergence iteration as using unprecondtioned CG at the same grid size h. This makes sense since diagonal scaling does not afford a dramatic improvement. From Figure 4-6 we can see that when we set $p = 1$ and use Jacobi as the preconditioner for PCG, as the grid size increases, the approximation solution becomes almost the same as the exact solution, and we can also come up with the same conclusions when we use PCG with SSOR as the preconditioner and unpreconditioned CG as shown in Figure 7-9 and Figure 10-12 respectively.

2.3 Start with a grid size $h = .01$ and deteremine the behavior of the CG and PCG algorithm when h is both increased by 10 and decreased by 10.

As shown in Table 1-3, for the PCG with Jacobi, SSOR as preconditioner and unpreconditioned CG, we can see in the first column that when grid size h is both increased, the convergence iterations are also increased. From Figure 4-6, 7-9 and 10-12 respectively we can see that increase the grid size h can lead to better approximation results, and it seems that the grid size $h = 0.01$ is a good grid size choice since it take a better trade-off between the iteration and the actual error. What's more, the third column in Table 1-3 shows the infinite norm of the actual error when each method converges based on the convergence criteria $\epsilon = h^2$, and by comparing the infinite norm of actual error at convergence when using grid size $h = 0.1, 0.01, 0.001$, we can see that when the grid size h increase or decrease by a factor of 10^1, the corresponding infinite norm of actual error at convergence also increases or decreases in a factor of 10^2. This result is the same for all these three methods, which means the norm of the error scales as h^2 and it also indicates that I choose a good enough convergence criteria such that the PCG and CG can converge sufficiently then all of the error come from truncation error since truncation error is proportional to h^2 as I discussed before.

2.4 Determine how closely the residuals track the actual error obtained from using Equation 2.

Since the actual error can never be driven down to zero based on the truncation error, hence what we can do is to sufficiently reduce the residual error so that all the error should come from truncation error. From Table 1-3, the third column shows the infinite norm of the actual error when each method is converged using the convergence criteria $\epsilon = h^2$, and by comparing the infinite norm of actual error at convergence when using grid size $h = 0.1, 0.01, 0.001$, we can see that when the grid size h decreases in a factor of 10^{-1}, the corresponding infinite norm of actual error at convergence also decreases in a factor of 10^{-2}, which means the norm of the error scales as h^2. Therefore the results again justify my convergence criteria when $p = 1$. What's more, as shown in Figure 1-3, when $p = 1$ and $h = 0.1$, the

relationship between infinite norm of residual and actual error norm shows that the residual track the actual error closely and the infinite norm of the residual vector and actual error vector almost move in the same trend as the iteration increases. However, this pattern stops when the convergence is satisfied, at $iteration = 10, 6, 10$ for the PCG with Jacobi, SSOR as preconditioner and unpreconditioned CG respectively under the convergence criteria $\epsilon = h^2$ when $p = 1$ and $h = 0.1$ as shown in first row of the Table 1-3. After the convergence iteration, when we keep increasing the iteration, the infinite norm of the actual error remains almost the same while the residual error still goes down to zero as shown in Figure 1-3. Therefore under the convergence criteria $\epsilon = h^2$, the PCG method with Jacobi, SSOR as preconditioner and unpreconditioned CG converges sufficiently with the convergence criteria $\epsilon = h^2$ and all the error comes from the truncation error when the convergence is satisfied. Here I only give the examples about these three methods when $p = 1$ and $h = 0.1$ for illustration and the conclusion are the same in different grid size $h = 0.01, 0.001$.

Table 1: Numerical Results using PCG with Jacobi as preconditioners when $p = 1$

	Convergence Iteration	**Infinite-Norm of Actual Error at Convergence**
$h = 0.1$	10	5.1866e-03
$h = 0.01$	100	5.1730e-05
$h = 0.001$	1000	5.1727e-07

Table 2: Numerical Results using PCG with SSOR as preconditioners when $p = 1$

	Convergence Iteration	**Infinite-Norm of Actual Error at Convergence**
$h = 0.1$	6	5.1866e-03
$h = 0.01$	9	5.1732e-05
$h = 0.001$	14	5.1727e-07

Table 3: Numerical Results using unpreconditioned CG when $p = 1$

	Convergence Iteration	**Infinite-Norm of Actual Error at Convergence**
$h = 0.1$	10	5.1866e-03
$h = 0.01$	100	5.1731e-05
$h = 0.001$	1000	5.1727e-07

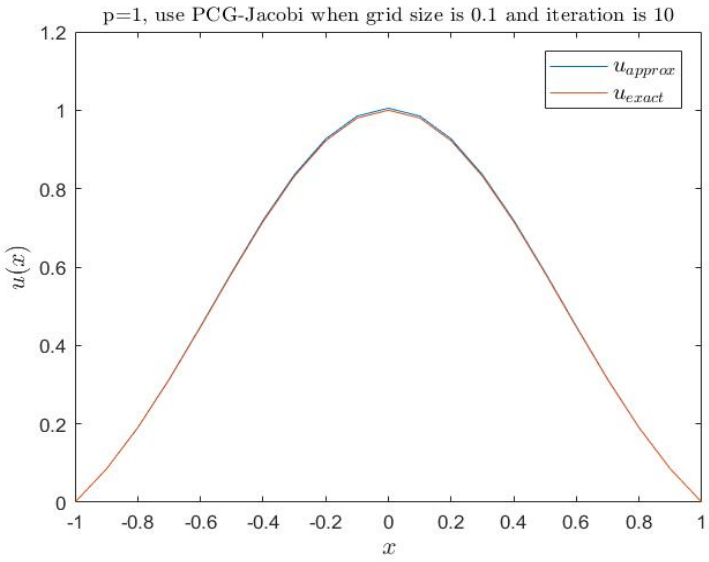

Figure 4: When $p = 1$ and $h = 0.1$, the approximation result using PCG with Jacobi method as preconditioner.

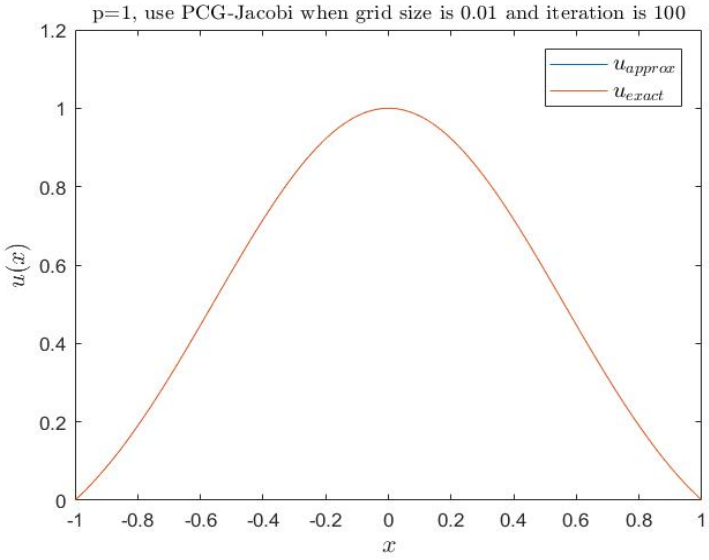

Figure 5: When $p = 1$ and $h = 0.01$, the approximation result using PCG with Jacobi method as preconditioner.

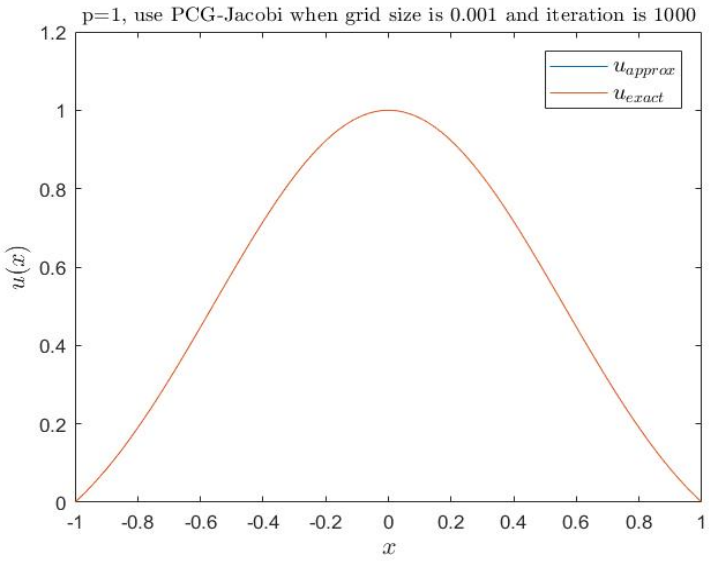

Figure 6: When $p = 1$ and $h = 0.001$, the approximation result using PCG with Jacobi method as preconditioner.

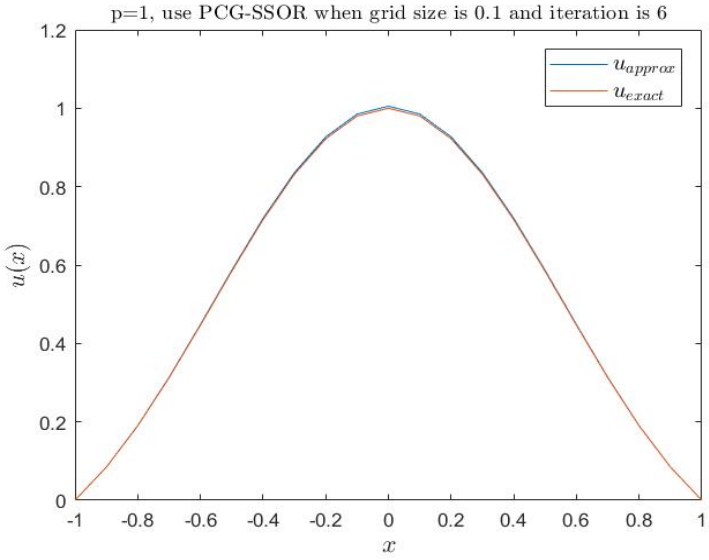

Figure 7: When $p = 1$ and $h = 0.1$, the approximation result using PCG with SSOR method as preconditioner.

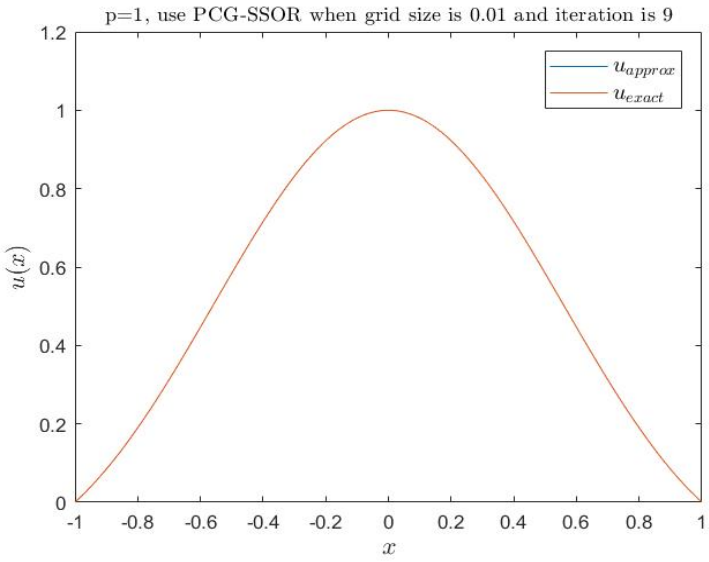

Figure 8: When $p = 1$ and $h = 0.01$, the approximation result using PCG with SSOR method as preconditioner.

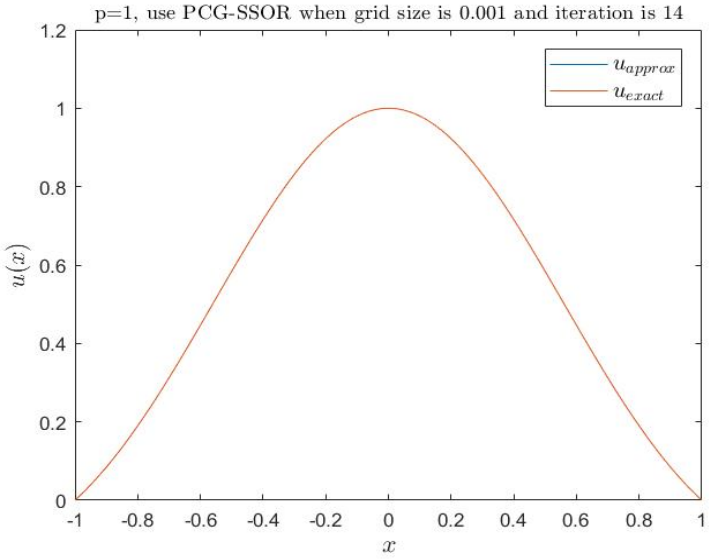

Figure 9: When $p = 1$ and $h = 0.001$, the approximation result using PCG with SSOR method as preconditioner.

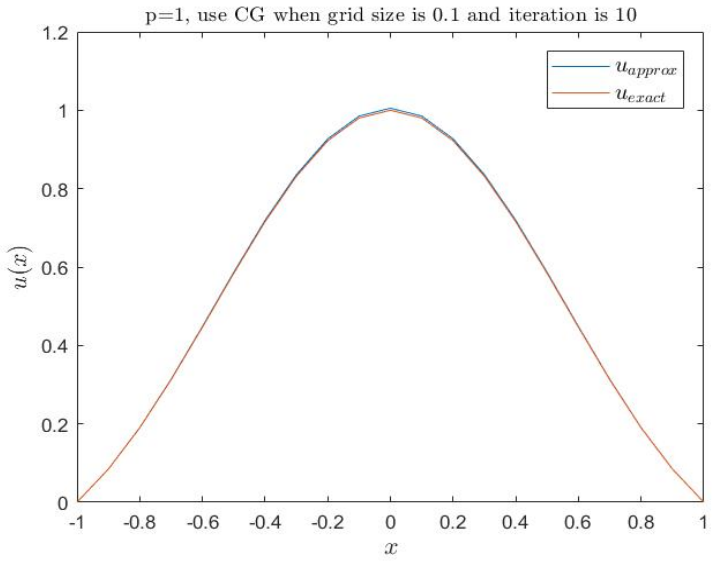

Figure 10: When $p = 1$ and $h = 0.1$, the approximation result using unpreconditioned CG.

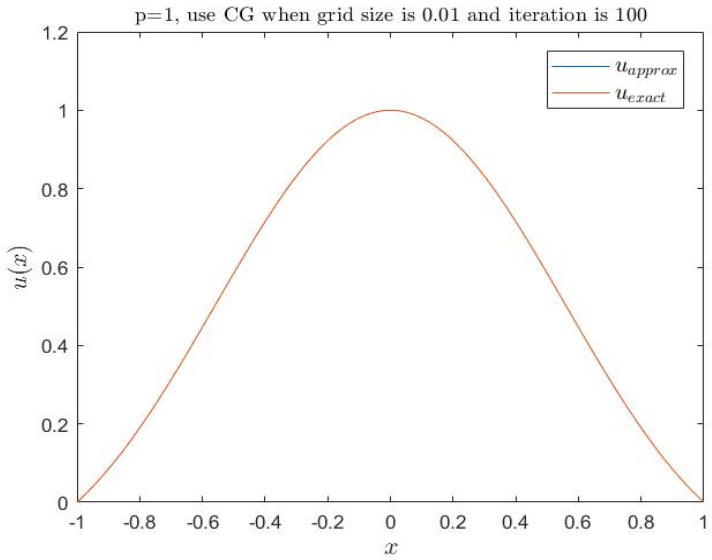

Figure 11: When $p = 1$ and $h = 0.01$, the approximation result using unpreconditioned CG.

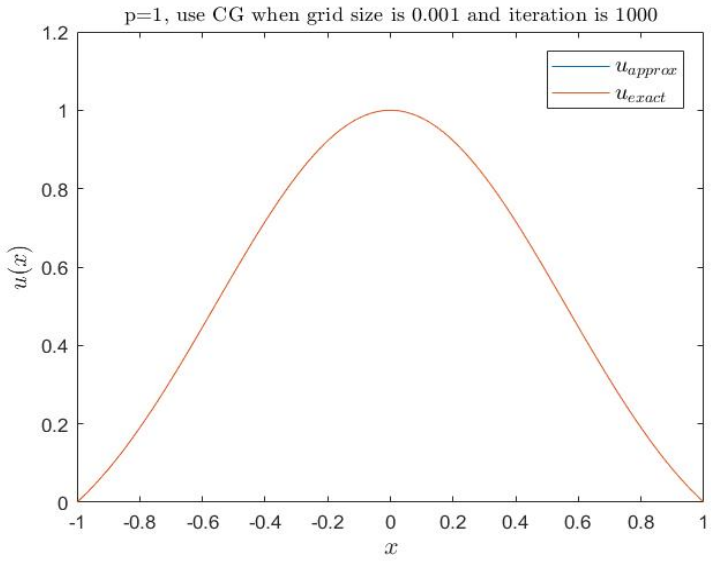

Figure 12: When $p = 1$ and $h = 0.001$, the approximation result using unpreconditioned CG.

3. Set $p = 1000$ and repeat the part (1)

When $p = 1000$, the exact solution of Equation (1) is show as Figure 13. We can see that it looks like a pulse between -0.1 and 0.1 approximately and the amplitude of it is 1.

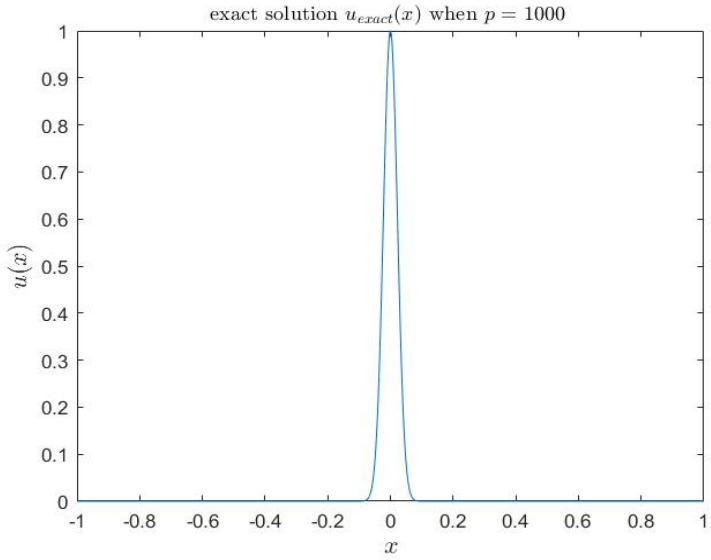

Figure 13: The exact solution of Equation (1) when $p = 1000$.

3.1 Convergence Criteria.

At this time since the $p = 1000$ is quite large, and we can see from Figure 13 that at $p = 1000$ the exact solution is not smooth as $p = 1$, which implies that if the grid size h is too large, there will very many grid points whose values are close to zero. For example when $h = 0.1$ there will be only one grid point inside the pulse and all the other's values are almost zero. I first numerically try the same

convergence criteria as $p = 1$ where $\epsilon = h^2$ and I find that for grid size $h = 0.01, 0.001$ it works well at $p = 1000$ and I can achieve the same conclusion as I mentioned at $p = 1$. The exception exists when $h = 0.1$, and I will elaborate that this is not due to the convergence criteria but due to the grid size itself in the later section. Since in practice we would not choose the grid size to be large and what I want is to choose a convergence criteria works for most of the grid size, therefore as discussed in the part (1), I will still use the same convergence criteria $\epsilon = h^2$ since I still want to sufficiently reduce the residual such that all the error come from the truncation error. The justification of the convergence criteria will be in each later section.

3.2 Case analysis when $h = 0.1$.

Based on the same convergence criteria, the approximation results obtained from the the PCG with Jacobi, SSOR as preconditioner and unpreconditioned CG, are shown as Table 4, 5 and 6 respectively and the numerical results are visualized as Figure 19-21, 22-24 and 25-27 respectively. Firstly I want to discuss the grid size $h = 0.1$ when $p = 1000$ since it is not a good choice for approximation. We can see that when $h = 0.1$, the numerical result obtained from these three methods as shown in Figure 19, 22 and 25 are bad and we can not come up with the same conclusions that as the grid size $h = 0.1$ increase a factor of 10^{-1} the infinite norm of the actual error does not increase by a factor of 10^{-2} as shown in the first and second rows in Table 4-6. The reason does not come from the convergence criteria since I will justify my convergence criteria later, but from the grid point size $h = 0.1$ itself. First I want to visualize the grid size $h = 0.1$ as shown in Figure 14. The grid point is denoted as blue $*$ and when grid size $h = 0.1$, we only have one grid point inside the pulse around $-0.1 < x < 0.1$. Since we need to approximate this pulse, with only one grid point inside we can only get a very sharp approximation since the corresponding u_{exact} will be sharp and behave like a triangle constructed by the values at $x = -0.1, 0, 0.1$. And it is definitely not sufficient for approximation propose. Furthermore, as shown in Figure 15 we can see that as the iteration increases, although the residual error can decrease, the actual error are increasing and then keep as the same, which implies that the actual error is actually at the minimum at the beginning. For illustration purpose I plot the numerical result at $iteration = 1$ as shown in Figure 16-18 using the PCG with SSOR, Jacobi as preconditioner and unpreconditioner CG respectively. The reason is that since the grid size $h = 0.1$ is too large for $p = 1000$, there are not enough grid point to approximate the pulse and then lead to larger truncation error. The result are the same when I change to the PCG with Jacobi as preconditioner and unpreconditioned CG. Since this relationship is unrelated to the convergence criteria I choose, so based on all the observation above I conclude that the grid size $h = 0.1$ is not a good choice when $p = 1000$ and the bad results are due to itself. I will further illustrate that when I change the grid size to $h = 0.025, 0.0025, 0.00025$, I can get the same results as part (1) using my convergence criteria $\epsilon = h^2$ in later session to justify that the convergence criteria still works for $p = 1000$. But first let's continue to look at the remaining results since I can get good approximation results when $h = 0.01, 0.001$ using these three methods. And in order to achieve the smallest actual error, I will only do one iteration when $h = 0.1$ such that the actual error will not be larger.

3.3 Compare your results from PCG with unpreconditioned CG.

First we can see from Table 4-6 that when $h = 0.01, 0.001$ and $p = 1000$, using SSOR as preconditioner for PCG can require less iterations to converge compared to the PCG with Jacobi as preconditioner and unpreconditioned CG, which means in each iteration, the PCG using SSOR as preconditioner reduce much more residual error than the other two methods. But at $h = 0.1$, using SSOR as preconditioner for PCG has larger infinite norm of actual error than the other two methods. The infinite norm of these three methods when $h = 0.01, 0.001$ and $p = 1000$ are almost the same, which indicates that the residual is sufficiently reduced and all the error come from the truncation error. Then we can see from the Table 5 and 6 that using the Jacobi (diagonal scaling) as the preconditioner for PCG has the same convergence iteration as using unpreconditioned CG method. This is the same result as shown in part (1) when $p = 1000$ and the reason is the same that diagonal scaling does not afford a dramatic improvement. Further more from Figure 20 and 21, Figure 23 and 24 and Figure 26 and 27, when $p = 1000$ and grid size $h = 0.01, 0.001$, the approximation obtained from PCG using Jacobi, SSOR as preconditioner and unprecondtioned CG are good enough based on my convergence criteria. The exception is the grid size $h = 0.1$ and I have already discussed the reasons before. To better elaborate on my convergence criteria, I do additional experiment at grid size $h = 0.025, 0.0025, 0.00025$ as shown in Figure 28-30, 31-33 and 34-36 for the PCG with Jacobi, SSOR as preconditioner and unpreconditioned CG respectively and we can come up with the same conclusions as I discussed in part (1) that as the grid size increases we can get a better approximation results and when the grid size $h = 0.0025, 0.00025$, the approximation results are almost the same as the exact solution.

3.4 Start with a grid size $h = 0.01$ and determine the behavior of the CG and PCG algorithms when h is both increased by 10 and decreased by 10.

As shown in the third column of Table 4-6, when the grid size $h = 0.01$ increase to $h = 0.001$ with a factor of 10^{-1}, the infinite norm of the actual error at convergence also increase with a factor of 10^{-2}. Again as shown in Figure 20-21, Figure 23-24 and Figure 26 and 27 for the PCG using Jacobi, SSOR as preconditioner and unpreconditioned CG respectively, when the grid size becomes smaller as $h = 0.001$, the approximation results are almost the same as the exact solution and they are all good enough. Therefore we get the same conclusions as I mentioned when $p = 1$ and to better justify my convergence criteria, I do additional experiments using grid size $h = 0.025, 0.0025, 0.00025$ and SSOR as preconditioner for PCG method with the same convergence criteria $\epsilon = h^2$. As shown in Table 7 we can see that when the grid size h increases or decreases by a factor of 10^1, the infinite norm of actual error at convergence also increases or decreases by a factor of 10^2. This is the same as I mentioned when $p = 1$ and it can also justify my convergence criteria that it also works well when $p = 1000$ and the grid point $h = 0.1$ is just a bad choice of grid size for approximation. The conclusions are the same as I turn to the PCG with Jacobi as preconditioner and unpreconditioned CG method when grid size $h = 0.025, 0.0025, 0.00025$ as shown in Table 8 and 9.

3.5 Determine how closely the residuals track the actual error obtained from using Equation (2)

As discussed in part (1) we know that the the actual error can never be driven down to zero based on the truncation error and what we can do is to sufficiently reduce the residual error such that all the error should come from truncation error. At $p = 1000$, the theoretical analysis still holds and the empirical conclusions are the same as $p = 1$. At $p = 1000$, we know that for grid size $h = 0.1$, the infinite norm of the actual error increase in the first two iteration and then remain the same as shown in Figure 15 since the grid size $h = 0.1$ is too large for $p = 1000$. Therefore I will use $h = 0.01, 0.001$ as instance for $p = 1000$ to illustrate again the empirical conclusions. From Table 4-6 we can still see that when the grid size $h = 0.01$ decreases in a factor of 10 tp $h = 0.001$, the corresponding infinite norm of actual error at convergence decreases in a factor of 10^2. Moreover in the same grid size h we can again find that the infinite norm of the actual error at convergence are almost the same. These results imply that the norm of the error scales as h^2 when the grid size is small enough at $p = 1000$ and the residual is reduced sufficiently such that all the error come from the truncation error. Also as shown in Figure 37 and 39, when using the PCG with Jacobi as preconditioner and unpreconditioned CG, we can see that before the convergence iteration, $iteration = 99$ as showin in Table 4 and 6, the infinite norm of the residual vector and the actual error vector are reducing in the same trend as iteration increases, and after the convergence the actual error remain almost the same and the residual keep going down, which is the same as I discussed in the part (1). To better justify my conclusion I do additional experiment using grid size $h = 0.025, 0.0025, 0.00025$ and using the PCG with SSOR, Jacobi as preconditioner and unpreconditioned CG, which is shown in Table 7, 8 and 9 respectively. We can come up with the same conclusions at $p = 1$ that when the grid size is increased or decreased in a factor of 10, the corresponding norm of actual error at convergence is also increased or decreased in a factor of 10^2, and all the other conclusions are the same as those at $p = 1$ when I use these grid size $h = 0.025, 0.0025, 0.00025$ for illustration.

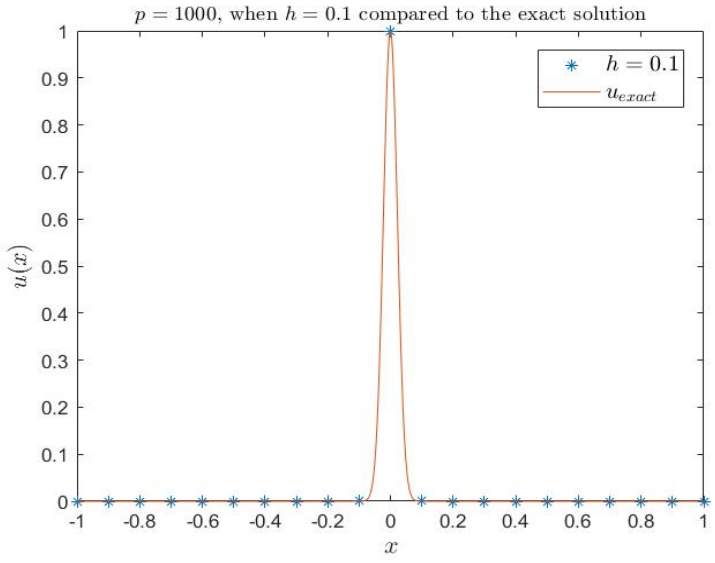

Figure 14: Illustration why grid size $h = 0.1$ is not a good choice for numerial approximation.

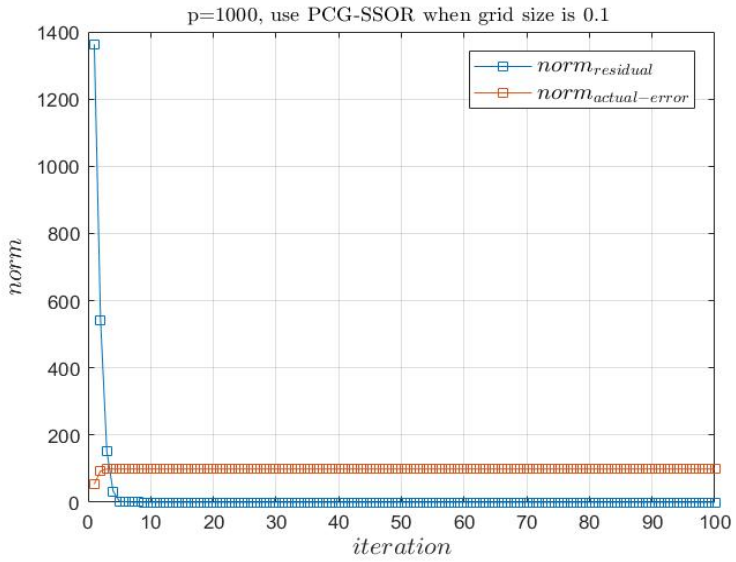

Figure 15: Relationship between the infinite norm of residual vector (denotes as the blue) and the actual error vector (denotes as the orange) when $p = 1000$ using SSOR as preconditioner for PCG method.

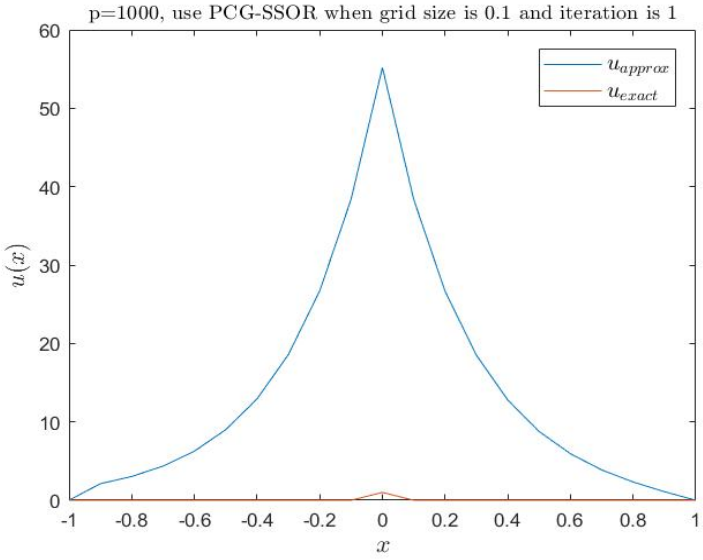

Figure 16: $p = 1000$, $iteration = 1$, the approximation result using PCG with SSOR as preconditioner.

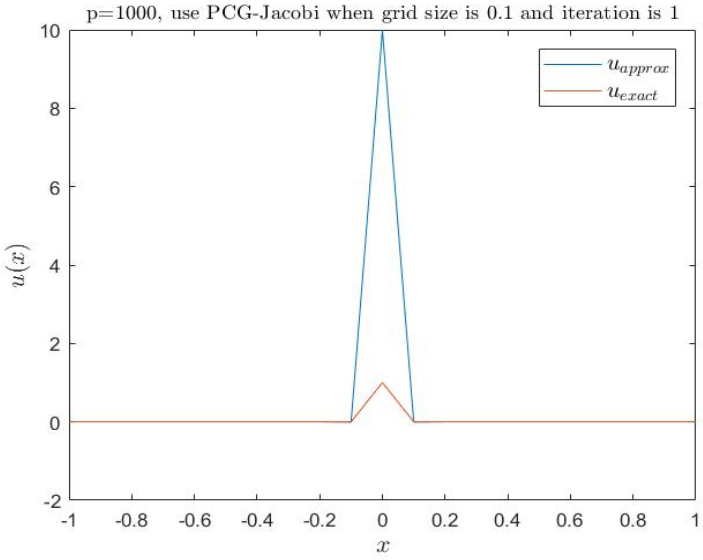

Figure 17: $p = 1000$, $iteration = 1$, the approximation result using PCG with Jacobi as preconditioner.

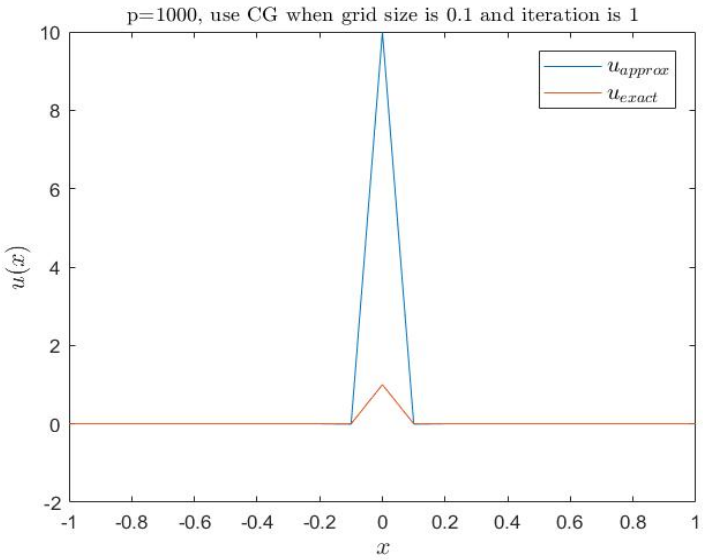

Figure 18: $p = 1000$, $iteration = 1$, the approximation result using PCG with CG as preconditioner.

Table 4: Numerical Results using PCG with Jacobi as preconditioners when $p = 1000$

	Convergence Iteration	**Infinite Norm of Actual Error at Convergence**
$h = 0.1$	1	8.9929
$h = 0.01$	99	1.720e-02
$h = 0.001$	943	1.6688e-04

Table 5: Numerical Results using PCG with SSOR as preconditioners when $p = 1000$

	Convergence Iteration	**Infinite-Norm of Actual Error at Convergence**
$h = 0.1$	1	54.1914
$h = 0.01$	7	1.720e-02
$h = 0.001$	11	1.6689e-04

Table 6: Numerical Results using unpreconditioned CG when $p = 1000$

	Convergence Iteration	Infinite-Norm of Actual Error at Convergence
$h = 0.1$	1	8.9929
$h = 0.01$	99	1.720e-02
$h = 0.001$	943	1.6688e-04

Table 7: Numerical Results using PCG with SSOR as preconditioner when $p = 1000$

	Convergence Iteration	Infinite-Norm of Actual Error at Convergence
$h = 0.025$	4	1.0514e-01
$h = 0.0025$	8	1.0447e-03
$h = 0.00025$	13	1.0427e-05

Table 8: Numerical Results using PCG with Jacobi as preconditioner when $p = 1000$

	Convergence Iteration	Infinite-Norm of Actual Error at Convergence
$h = 0.025$	39	0.0895
$h = 0.0025$	378	0.0010
$h = 0.00025$	3825	1.0427e-05

Table 9: Numerical Results using unpreconditioned CG as preconditioner when $p = 1000$

	Convergence Iteration	Infinite-Norm of Actual Error at Convergence
$h = 0.025$	39	0.0895
$h = 0.0025$	378	0.0010
$h = 0.00025$	3825	1.0427e-05

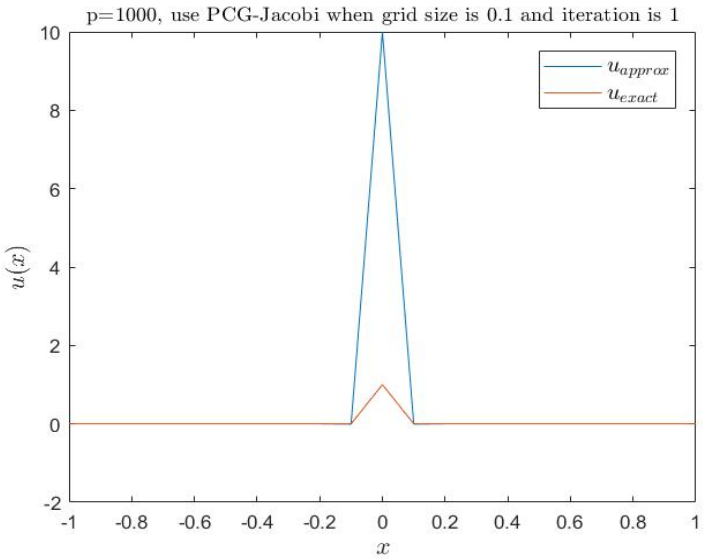

Figure 19: When $p = 1000$ and $h = 0.1$, the approximation result using PCG with Jacobi method as preconditioner.

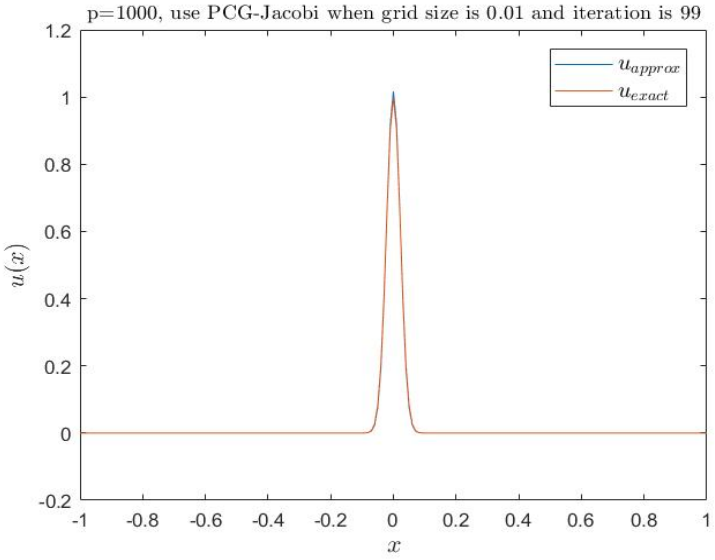

Figure 20: When $p = 1000$ and $h = 0.01$, the approximation result using PCG with Jacobi method as preconditioner.

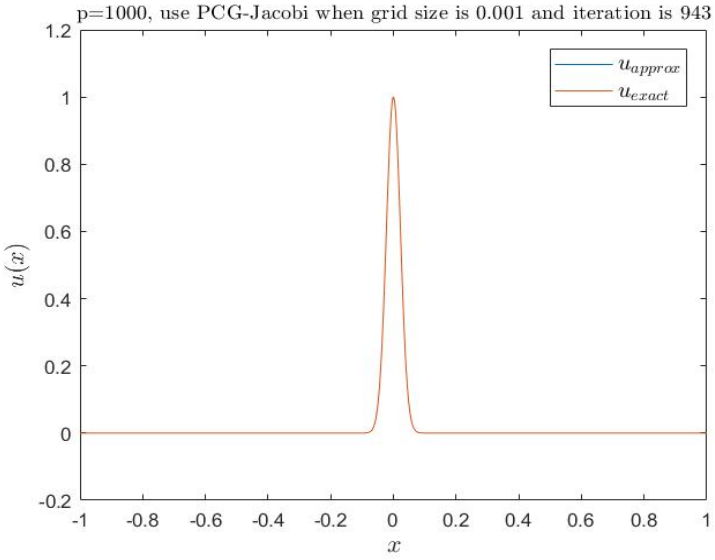

Figure 21: When $p = 1000$ and $h = 0.001$, the approximation result using PCG with Jacobi method as preconditioner.

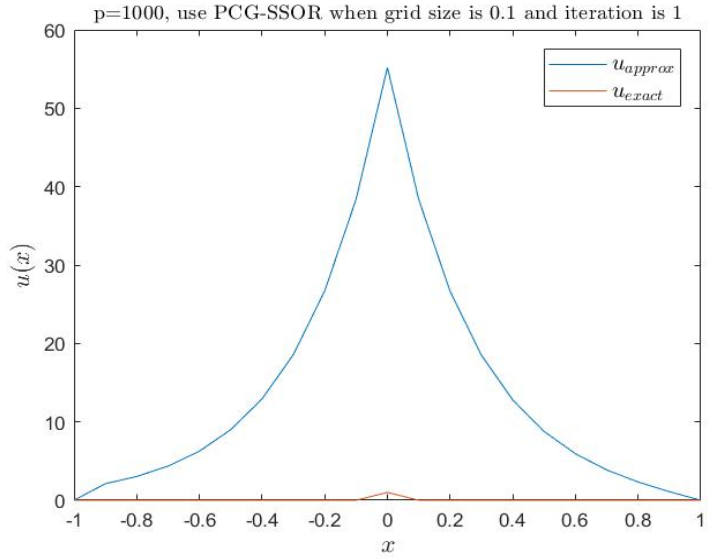

Figure 22: When $p = 1000$ and $h = 0.1$, the approximation result using PCG with SSOR method as preconditioner.

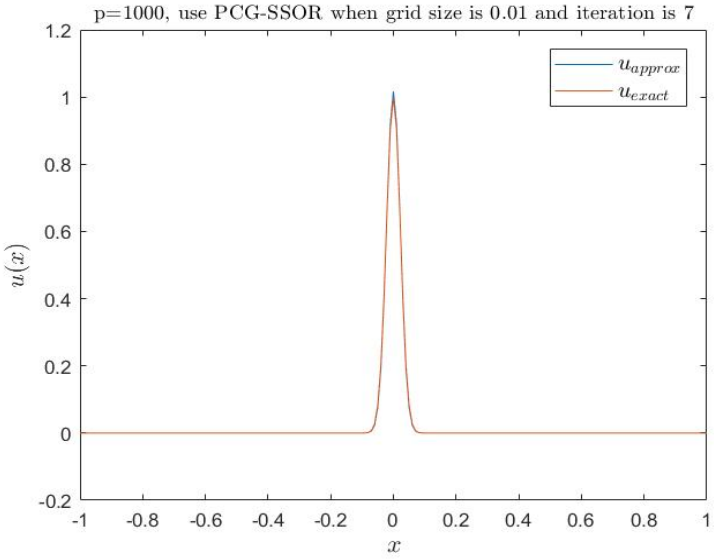

Figure 23: When $p = 1000$ and $h = 0.01$, the approximation result using PCG with SSOR method as preconditioner.

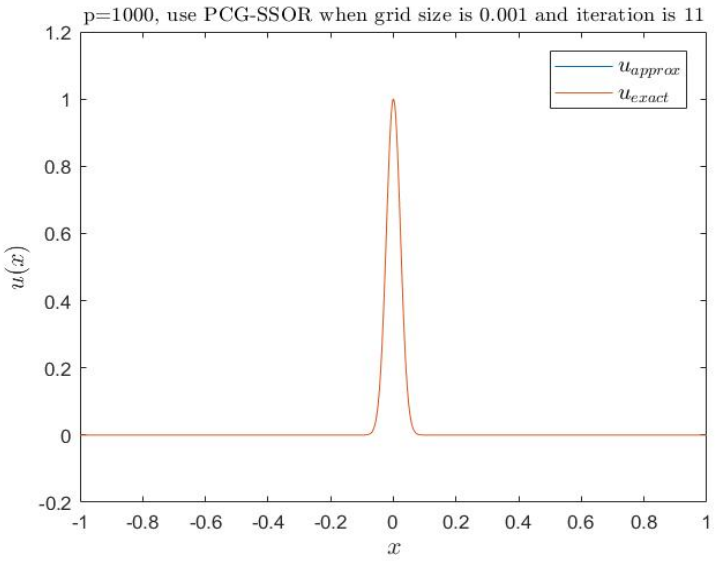

Figure 24: When $p = 1000$ and $h = 0.001$, the approximation result using PCG with SSOR method as preconditioner.

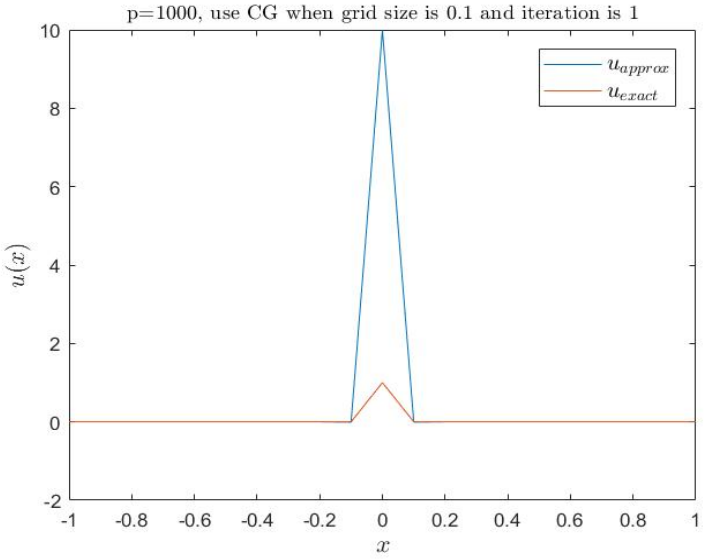

Figure 25: When $p = 1000$ and $h = 0.1$, the approximation result using unpreconditioned CG.

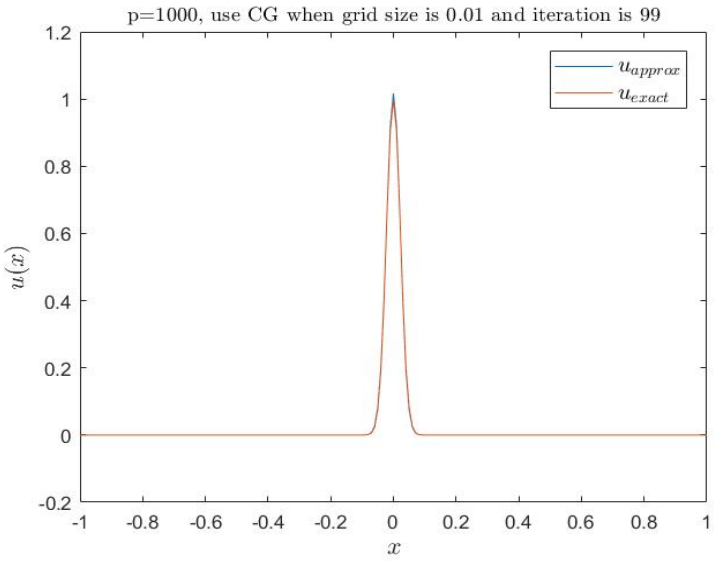

Figure 26: When $p = 1000$ and $h = 0.01$, the approximation result using unpreconditioned CG.

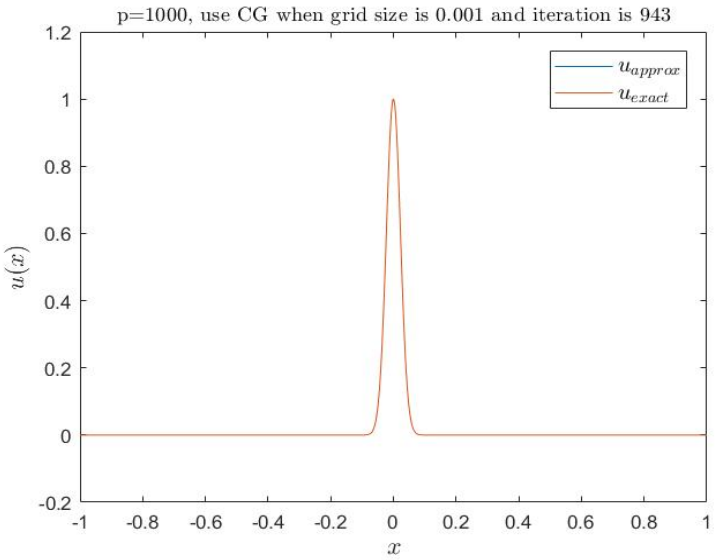

Figure 27: When $p = 1000$ and $h = 0.001$, the approximation result using unpreconditioned CG.

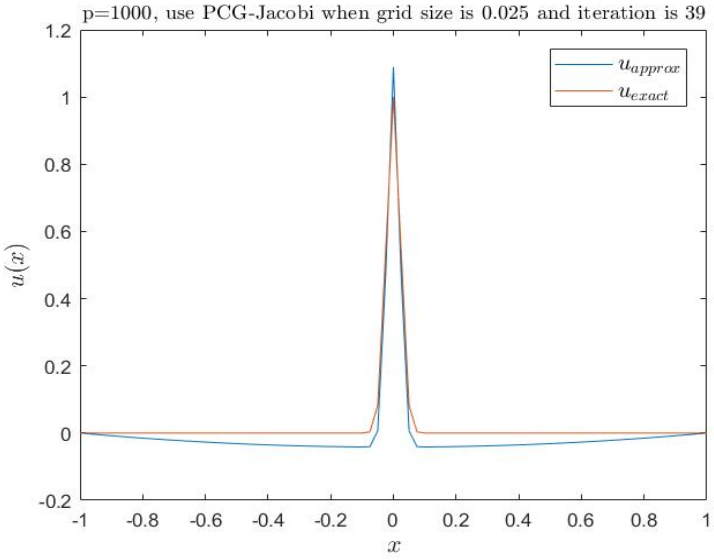

Figure 28: When $p = 1000$ and $h = 0.025$, the approximation result using PCG with Jacobi as preconditioner.

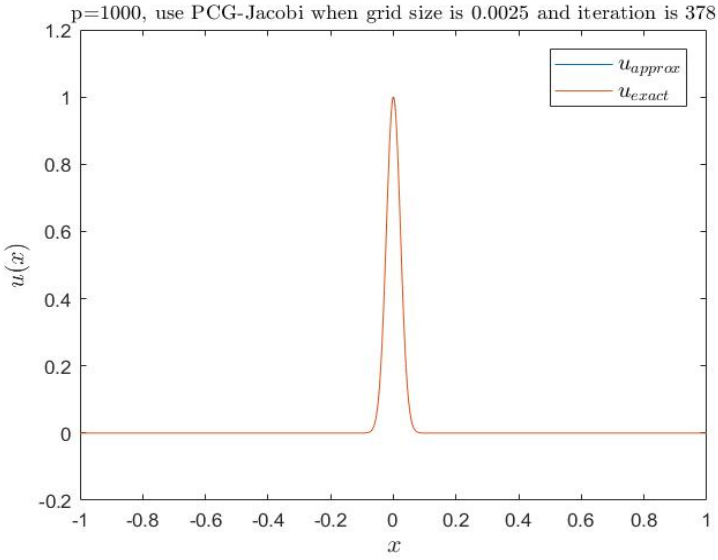

Figure 29: When $p = 1000$ and $h = 0.0025$, the approximation result using PCG with Jacobi as preconditioner.

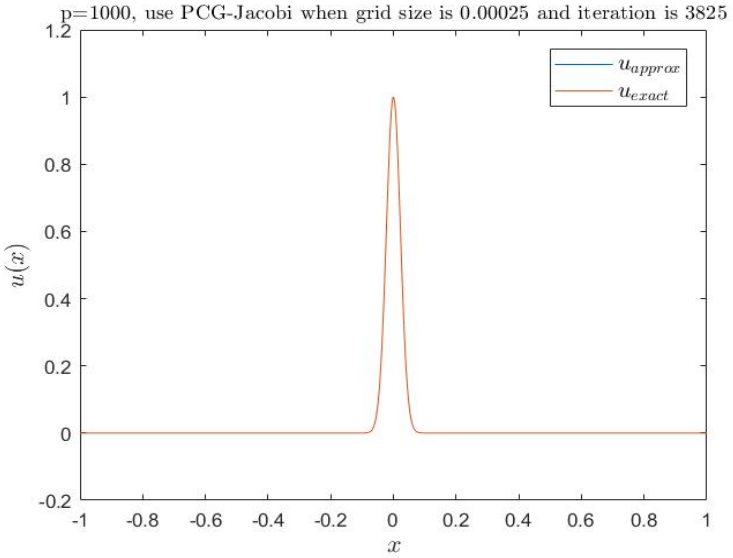

Figure 30: When $p = 1000$ and $h = 0.00025$, the approximation result using PCG with Jacobi as preconditioner.

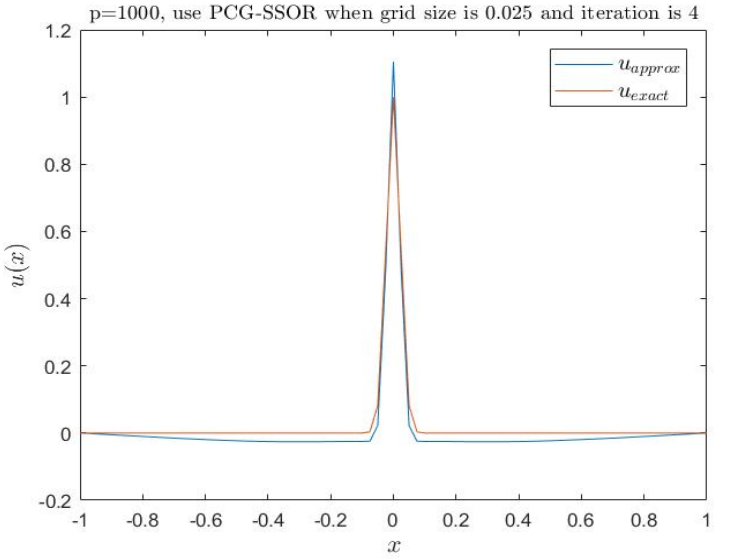

Figure 31: When $p = 1000$ and $h = 0.025$, the approximation result using PCG with SSOR as preconditioner.

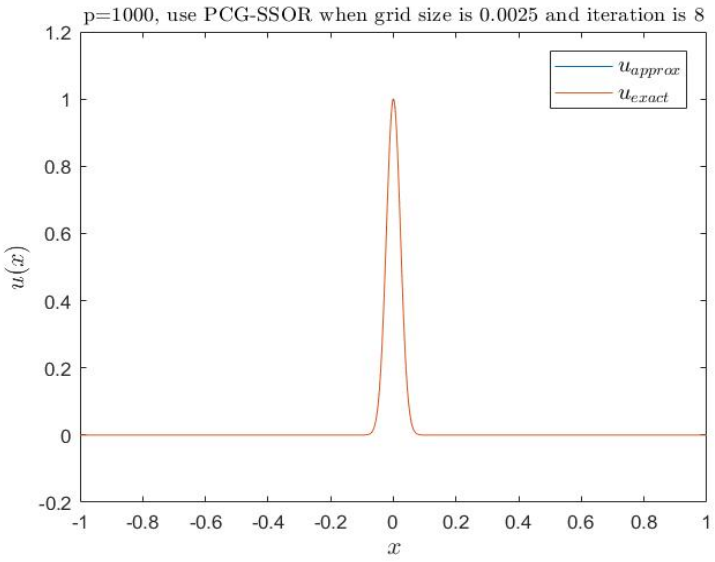

Figure 32: When $p = 1000$ and $h = 0.0025$, the approximation result using PCG with SSOR as preconditioner.

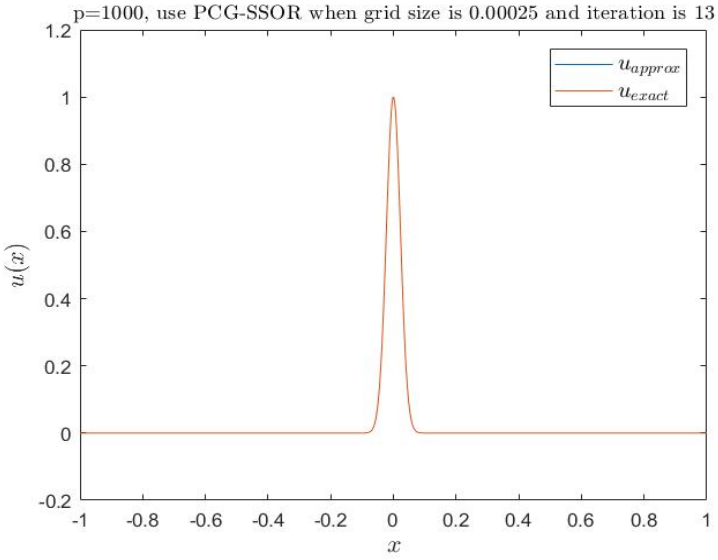

Figure 33: When $p = 1000$ and $h = 0.00025$, the approximation result using PCG with SSOR as preconditioner.

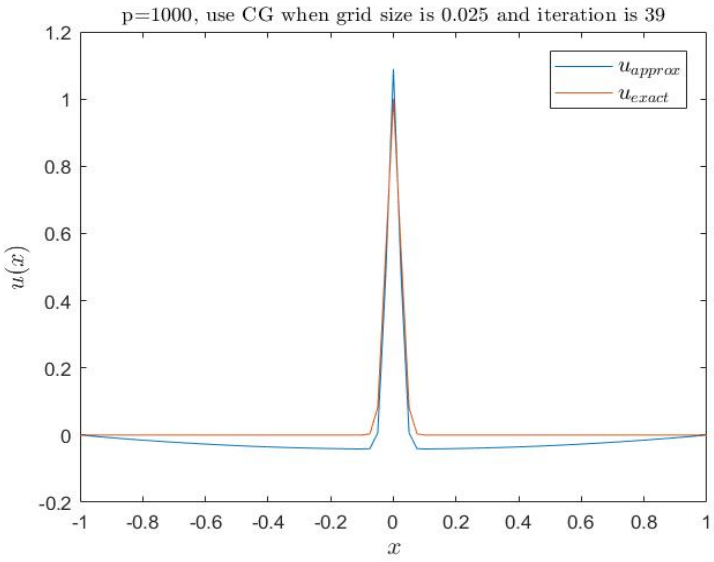

Figure 34: When $p = 1000$ and $h = 0.025$, the approximation result using unpreconditioned CG.

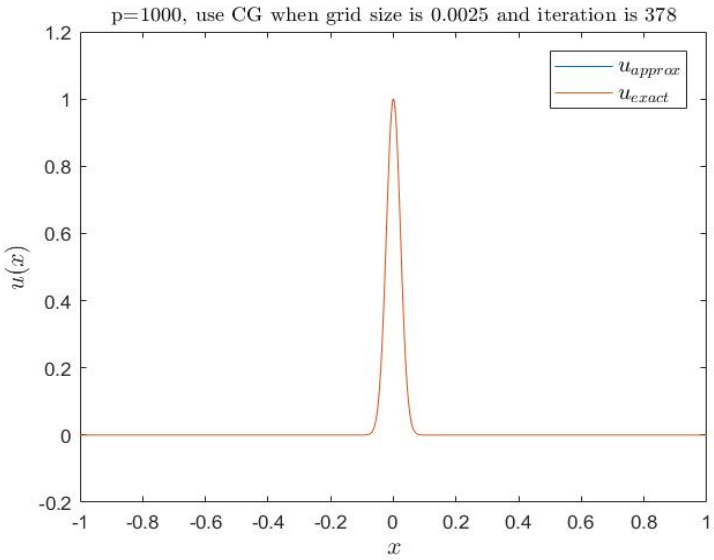

Figure 35: When $p = 1000$ and $h = 0.0025$, the approximation result using unpreconditioned CG.

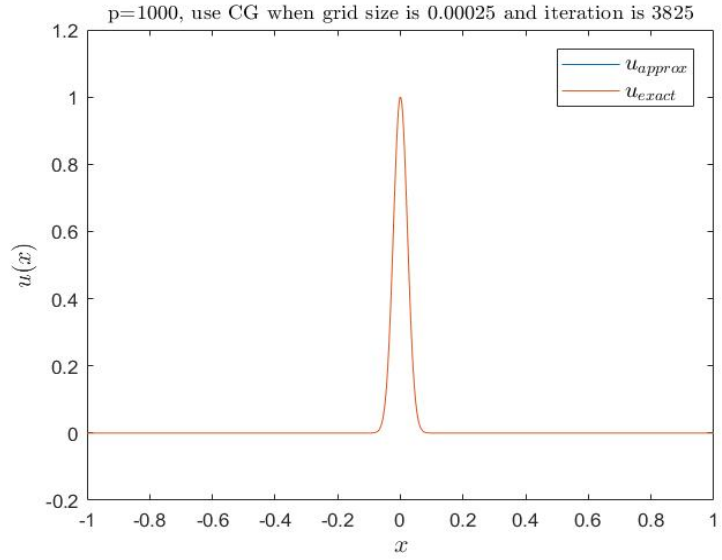

Figure 36: When $p = 1000$ and $h = 0.00025$, the approximation result using unpreconditioned CG.

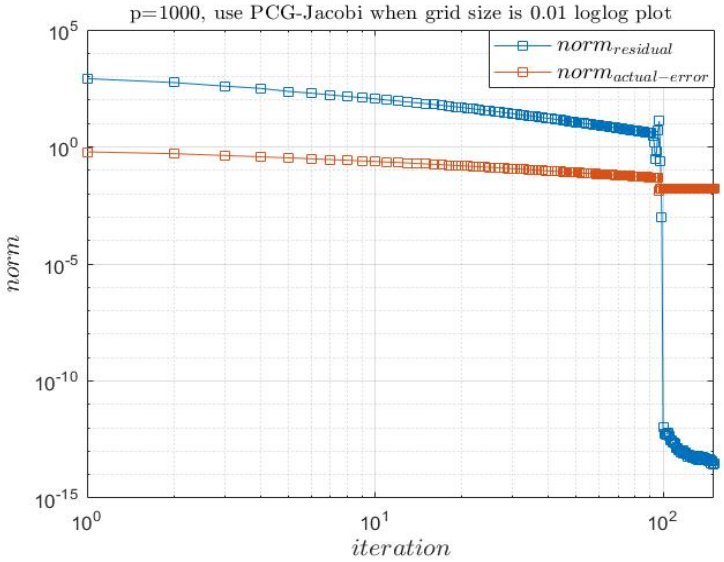

Figure 37: When $p = 1000$ and $h = 0.01$, loglog plot of the iteration verse the infinite norm of the residual vector (denotes as the blue) and the actual error vector (denotes as the orange) using Jacobi as preconditioner for PCG method.

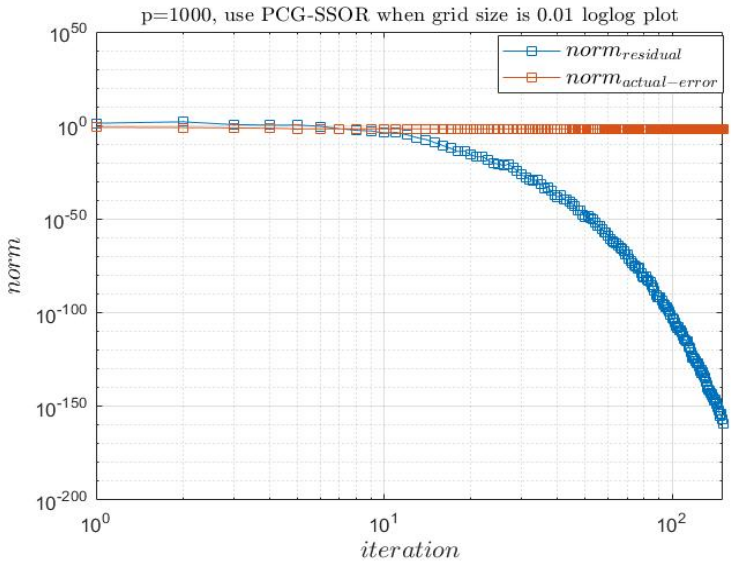

Figure 38: When $p = 1000$ and $h = 0.01$, loglog plot of the iteration verse the infinite norm of the residual vector (denotes as the blue) and the actual error vector (denotes as the orange) using SSOR as preconditioner for PCG method.

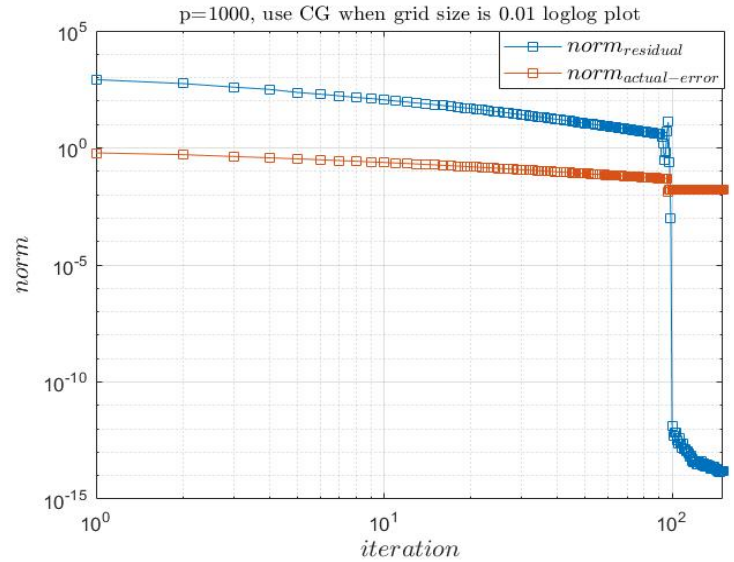

Figure 39: When $p = 1000$ and $h = 0.01$, loglog plot of the iteration verse the infinite norm of the residual vector (denotes as the blue) and the actual error vector (denotes as the orange) using unpreconditioned CG.

4. MATLAB code

4.1 Tridiagonal matrix A multiplication

```
function Apk=A_pk(p_k,h,N)
% Author: Mingfu Liang
```

```
% Date: 2019/05/07
%
% Compute the multiplication between the tridiagonal matrix A and column
% vector p_k without computing the zero element in matrix A and only take
% advantage of the bands of matrix A.
%
% Input:
%          p_k:
%               The column vector p_k
%
%          h:
%                The grid size h
%
%          N:
%                The dimension of the matrix A
%
% Output:
%         Apk:
%                The multiplication of A * p_k, the dimension of Apk
%                should be N * 1


            Apk(1,1)=(1/h^2)*(2*p_k(1,1) - p_k(2,1)); % the first entry of
                the multiplication
            Apk(2:N-1,1)=(1/h^2)*(-p_k(1:end-2,1)+2*p_k(2:end-1,1)-p_k(3:end
                ,1)); % the second to the last but one entry
            Apk(N,1)=(1/h^2)*(-p_k(end-1,1) +2 * p_k(end,1)); % the last
                entry of the multiplication
end
```

4.2 Preconditioned Conjugate Gradient Method

```
function ESAM445_PCG_MingfuLiang(h,p,method)
% Author: Mingfu Liang
% Date: 2019/05/07
%
% Implementation of Preconditioned Conjugate Gradient Method with different
% Relaxation scheme to solve the problem
%
%                       -u_{xx} = f(x), u(-1)=u(1)=0
%
```

```
10 %                   where       u_{exact}(x) = (1 - x^2) * exp(- p * x^2)
11 %
12 % Input:
13 %           h:
14 %                Input different grid size choice for the PCG method
15 %
16 %                grid size of the PCG method.
17 %
18 %
19 %           p:
20 %                 Input different value to determine the problem form
21 %
22 %                 parameter of problem.
23 %
24 %           mehod:
25 %                 Input the preconditioner for the PCG and in this problem
26 %                 we have SSOR and Jacobi relaxtion method for
27 %                 preconditioner, and we can also choose just CG method
28 %
29 %                If you want to choose CG, type 'CG'
30 %                If you want to choose SSOR, type 'PCG-SSOR'
31 %                If you want to choose Jacobi, type 'PCG-Jacobi'
32 %
33 % Example:
34 %
35 %            ESAM445_PCG_MingfuLiang(0.01,1,'PCG-SSOR')
36 %
37 %            means that you are going to set the grid size h=0.01 and p=1
38 %            to do the PCG method with SSOR as preconditioner for solving
39 %            the problem
40 %
41
42 epslion = h^2; % based on the truncation error, here for the convergence
       critiria
43                          % we should make sure that all the error come from
                               the
44                          % truncation error and since the truncation error
                               is proportional to h^2
45                          % when we are using the central difference
                               approximation, therefore we need to
```

```
46                          % reduce the residual error to less than h^2 such
                                that all the error come from the truncation
                                error
47
48 range=-1:h:1; % range of $x$ based on the grid size $h$
49 RHS = @(x) -(10*p*x.*x-2+4*p^2*x.*x-4*p^2*x.^4+-2*p).*exp(-p*x.*x); %
       Anonymous vectorized version of f(x)
50 exact_sol = @(x) (1-x.*x).*exp(-p.*x.*x); % Anonymous vectorized version of
       $u_{exact}$
51 f = RHS(range); % generate the f(x)
52 u_exact = exact_sol(range); % generate the $u_{exact}$
53 f = f'; % transpose the f(x) such that it is a column vector
54 u_exact = u_exact'; %% transpose the $u_{exact}$ such that it is a column
       vector
55 N = round((1-(-1))/h -1); % here I don't add one since I want to be
       consistent with Professor Bayliss Notebook where h= 1/(N+1) in Page 11
56 %method = 'PCG-SSOR'; % decide what preconditioned method you are going to
       use, the choices are: 'CG', 'PCG-Jacobi' and 'PCG-SSOR'
57 w = 2/(1+sqrt(2)*sin(pi*h)); % the optimal choice of the $\omega$ when using
        the SSOR
58
59 %%% test the vectorized version of f(x) and u_exact(x)
60
61 % k=1;
62 % for x=0:h:1
63 %     f_test(k)=-2*exp(-p*x^2)+8*p*(x^2)*exp(-p*x^2)+(1-x^2)*(-2*p*exp(-p*x
       ^2)+4*(p^2)*(x^2)*exp(-p*x^2));
64 %     k=k+1;
65 % end
66
67 % k=1;
68 % for x=-1:h:1
69 %     u_exact_test(k)=(1-x^2)*exp(-p*x^2);
70 %     k=k+1;
71 % end
72
73 %%% Construct Tridiagonal Matrix A
74
75 %upper_diagonal = -1*ones(N-1);
76 %main_diagonal = 2*ones(N);
```

```
77 %lower_diagonal = -1*ones(N-1);
78
79 %%%%%%% un-comment them for debug propose if you want to check the matrix-
       form PCG %%%%%%%
80
81 % A_diag = (1/h^2)*(2*eye(N));
82 % A_upper_vec = (1/h^2)*(-1 *  ones(N-1,1));
83 % A_upper = diag(A_upper_vec,1);
84 % A_lower = diag(A_upper_vec,-1);
85 % A = A_diag + A_upper+A_lower; % define the first block matrix A1
86 % A_S = sparse(A); % construct the sparse form of matrix A to takes
       advantages of the bands of the matrix A
87
88 %%% test the efficiency about using the sparse matrix
89
90 % p_test = rand(N,1);
91 % tic
92 % A_S_p = A_S * p_test;
93 % elapsedTime = toc;
94 % fprintf('\n Using Sparse matrix need %4d seconds \n', elapsedTime);
95 %
96 % tic
97 % Ap = A * p_test;
98 % elapsedTime = toc;
99 % fprintf('\n Using Non-Sparse matrix need %4d seconds \n', elapsedTime);
100
101 %%% Preconditioned Conjugate Gradient Algorithm - PCG %%%%%%%%
102 %%% All the procedure are following Professor Bayliss Note in Page 82-83 %%
103
104 x_0 = zeros(N+2,1); % initial guess of the solution x
105 x_k =x_0; % initialize x
106 r_0 = x_0; % initialize r_0
107 r_0(2:N+1,1) = f(2:N+1,1); % r_0 = b_0;
108 z_0 = x_0; % initialize z_0
109 z_k = z_0; % initialize z_k
110 r_k = r_0; % initialize r_k
111
112 if strcmp(method,'CG') % if the method just uses CG, then using the
       following setup for z_k
113         fprintf('\n Using CG as preconditioner \n');
```

```
114         z_k(2:N+1,1) =r_k(2:N+1,1);
115
116         %%%%%%% un-comment them for debug propose if you want to check the
                matrix-form PCG %%%%%%%
117 %         M = eye(N);
118 %         M = sparse(M);
119 %         z_k = inv(M)*r_k(2:N+1,1);
120 end
121
122 if strcmp(method,'PCG-Jacobi') % if the preconditioned method is Jacobi,
    then using the following setup for z_k
123         fprintf('Using Jacobi as preconditioner');
124         i = 2:N+1;
125
126         %% Since here z_k is initialized as zero before so here we did not
127         %% need to write the z_k(i-1,1) and z_k(i+1,1) again since there
128         %% are all zero here.
129
130         z_k(i,1) = (1/2)* (h*h) * r_k(i,1) ;
131
132         %%%%%%% un-comment them for debug propose if you want to check the
                matrix-form PCG %%%%%%%
133 %     M = A_diag;
134 %     M = sparse(M);
135 %     z_k(2:N+1) = inv(M)*r_k(2:N+1,1);
136
137 end
138
139 if strcmp(method,'PCG-SSOR') % if the preconditioned method is SSOR, then
    using the following setup for z_k
140
141     fprintf('Using SSOR as preconditioner');
142
143     %%% Do one SSOR and since here the z_k is initialized as zero as
144     %%% before so we just use it
145
146      %%%%%%%% forward %%%%%%%%%%
147         for i=2:N+1
148             z_k(i,1) = (1-w)*z_k(i,1) +w*(1/2)*( z_k(i-1,1) + z_k(i+1,1) + (
                h*h)*r_k(i,1));
```

```
149         end
150     %%%%%%%% backward %%%%%%%%%%
151         for i=N+1:-1:2
152             z_k(i,1) = (1-w)*z_k(i,1) +w*(1/2)*( z_k(i-1,1) + z_k(i+1,1)  +
                    (h*h)*r_k(i,1));
153         end
154
155      %%%%%%%  un-comment them for debug propose %%%%%%
156 %      M = (A_diag+w*A_lower)*inv(A_diag)*(A_diag+w*A_upper)/(w*(2-w));
157 %      M =sparse(M);
158 %       z_k_test = inv(M)*r_k(2:N+1,1);
159 end
160
161 p_1 = z_k(2:N+1,1); % initialize the p_1
162 p_k = p_1; % initialize the p_k
163 ZTR = z_k(2:N+1,1)' * r_0(2:N+1,1); % initialize the ZTR
164
165 iter_counter=0; % initialize the loop counter
166 while 1
167     Apk=A_pk(p_k,h,N); % using the function A_pk to compute the A*pk such
          that
168                                     % we can take advantage of the band of
                                          the matrix A and
169                                     % only do the multiplication when the
                                          entry is non-zero in matrix A
170     alpha_k =ZTR/(p_k' * Apk); % compute the $\alpha_{k}$
171     x_k(2:N+1,1) = x_k(2:N+1,1) + alpha_k * p_k;  % update the solution x_k
172     r_k(2:N+1,1) = r_k(2:N+1,1) - alpha_k * Apk;  % update the residual r_k
173
174     %%%%%%%% un-comment it for debug propose if you want to check the
          matrix-form PCG %%%%%%%
175 %      z_k(2:N+1,1) = inv(M)*r_k(2:N+1,1);
176
177     if strcmp(method,'CG') % if the method is just CG, then using the
          following setup for z_k
178         z_k(2:N+1,1) = r_k(2:N+1,1); % here since the inverse of matrix M is
                the identity matrix
179                                                 % so we can save our
                                                      memory and just
                                                      use the r_k
```

```
                                                                directly
180     end
181
182     if strcmp(method,'PCG-Jacobi') % if the preconditioned method is Jacobi,
            then using the following setup for z_k
183         i = 2:N+1; % generate the vectorized index for parellel computation
184         %%% Since here z_k is initialized as zero each time, so here we did
                not
185         %%% need to write the z_k(i-1,1) and z_k(i+1,1) again since there
186         %%% are all zero here.
187         z_k(i,1) = (1/2)* (h*h) * r_k(i,1) ;
188     end
189
190     if strcmp(method,'PCG-SSOR') % if the preconditioned method is SSOR,
            then using the following setup for z_k
191         %%%%%%%%% forward SOR sweep of SSOR%%%%%%%%%%%%
192         z_k =zeros(N+2,1); % initialize z_k as zero each time
193         for i=2:N+1
194             z_k(i,1) = (1-w)*z_k(i,1) +w*(1/2)*( z_k(i-1,1) + z_k(i+1,1) +(h
                    *h)*r_k(i,1));
195         end
196     %%%%%%%%% backward SOR sweep of SSOR%%%%%%%%%%%%
197         for i=N+1:-1:2
198             z_k(i,1) = (1-w)*z_k(i,1) +w*(1/2)*( z_k(i-1,1) + z_k(i+1,1) + (
                    h*h)*r_k(i,1));
199         end
200     end
201
202     iter_counter = iter_counter+1; % update the iteration counter
203
204     %%%%%%%%%%%%% Check convergence%%%%%%%%%%%%%%%%%%%%
205
206     norm(r_k(2:N+1,1),inf)/norm(r_0(2:N+1,1),inf) % print out the norm to
            know exactly what is going on for debug.
207     if norm(r_k(2:N+1,1),inf)/norm(r_0(2:N+1,1),inf)<epslion
208         fprintf('\n Convergence achieve. \n')
209         break
210     end
211
212     %%%%%%% If no convergence then continue %%%%%%%%%%%%%%%%%%%
```

```
213
214         ZTRNEW = z_k(2:N+1,1)' * r_k(2:N+1,1); % update the ZTRNEW
215         beta_k = ZTRNEW/ZTR; % update the $\beta_{k}$
216         p_k = z_k(2:N+1,1) + beta_k * p_k; % update the p_k
217         ZTR = ZTRNEW; % update the ZTR
218
219 end
220
221 %%%%%%% Generate the visualization result of $u_{exact}$ and $x_k$ %%%
222 %%%%%%% $x_k$ is the numerical approximation of the exact solution %%%
223
224 figure;
225 plot(range,x_k);
226 hold on
227 plot(range,u_exact,'-');
228 hold off
229 ylabel('$u(x)$','Interpreter','latex','FontSize',13)
230 xlabel('$x$','Interpreter','latex','FontSize',13)
231 leg1 = legend('$x_{k}$','$u_{exact}$');
232 set(leg1,'Interpreter','latex');
233 set(leg1,'FontSize',12);
234 title(['p=',num2str(p),', use ',method,' when grid size is ',num2str(h),'
       and iteration is ',num2str(iter_counter)],'Interpreter','latex');
235 end
```