HOMEWORK 3 - FRANCISCO CASTILLO

Contents

Defined functions

```
function [PHI,res]=IterativeMethods1D(f,phi_0,x,h,N,opt,iter,iter_res)
% This function calculates the solution of the 1D PDE given the initial
% guess phi_0, the right hand side f and information about the mesh.
% We use the parameters iter and iter_res to know what values to store and
% give back to the main code.
kmax=max([iter iter_res]);
j=1;
phi=phi_0(x);
PHI=zeros(length(phi),length(iter));
res=zeros(length(phi),iter_res+1);
if strcmp(opt,'PJ')
    % This includes the boundary conditions
    phi_new=zeros(size(phi));
    for k=1:kmax
        for i=2:N %Only the middle points will change value
            phi_new(i)=0.5*(phi(i+1)+phi(i-1)-h^2*f(x(i)));
        end
        phi=phi_new;
        if k == iter(j)
            PHI(:,j)=phi;
            j=j+1;
        end
        if k<=iter_res
            for i=2:N
                res(i,k+1)=f(x(i))-((phi(i+1)-2*phi(i)+phi(i-1))/h^2);
            end
        end
    end
elseif strcmp(opt,'GS')
    phi(1)=0;
    phi(end)=0;
    for k=1:kmax
        for i=2:N %Only the middle points will change value
            phi(i)=0.5*(phi(i+1)+phi(i-1)-h^2*f(x(i)));
        end
        if k == iter(j)
            PHI(:,j)=phi;
            j=j+1;
        end
        if k<=iter_res
            for i=2:N
                res(i,k+1)=f(x(i))-((phi(i+1)-2*phi(i)+phi(i-1))/h^2);
            end
        end
    end
elseif strcmp(opt,'SOR')
    phi(1)=0;
    phi(end)=0;
    w=2/(1+sqrt(1-(cos(pi/N))^2));
    for k=1:1:kmax
        for i=2:N %Only the middle points will change value
            phi(i)=phi(i)+w*(0.5*(phi(i+1)+phi(i-1)-h^2*f(x(i)))-phi(i));
        end
        if k == iter(j)
            PHI(:,j)=phi;
            j=j+1;
        end
        if k<=iter_res
            for i=2:N
                res(i,k+1)=f(x(i))-(phi(i+1)-2*phi(i)+phi(i-1))/h^2;
            end
        end
    end
end
for i=2:N
    res(i,1)=f(x(i))-((phi_0(x(i+1))-2*phi_0(x(i))+phi_0(x(i-1)))/h^2);
end
end
```

```
function [PHI,res]=IterativeMethods2D(f,phi_0,x,h,M,N,opt,iter,iter_res)
% This function calculates the solution of the 2D PDE given the initial
% guess phi_0, the right hand side f and information about the mesh.
% We use the parameters iter and iter_res to know what values to store and
% give back to the main code.
kmax=max([iter iter_res]);
l=1;
phi=zeros(N+1,N+1);
for i=1:N+1
    for j=1:N+1
        phi(i,j)=phi_0(x(i),x(j));
    end
end
PHI=zeros(length(phi),length(phi),length(iter));% Includes BCs
res=zeros(length(phi),length(phi),iter_res+1);
if strcmp(opt,'GS')
    for k=1:kmax
        for i=2:N %Only the middle points will change value
            for j=2:N
                phi(i,j)=0.25*(phi(i+1,j)+phi(i-1,j)...
                    +phi(i,j+1)+phi(i,j-1)-h^2*f(x(i),x(j)));
            end
        end
        if k == iter(l)
            PHI(:,:,l)=phi;
            l=l+1;
        end
        if k<=iter_res
            for i=2:N
                for j=2:N
                    res(i,j,k+1)=f(x(i),x(j))-(phi(i+1,j)-2*phi(i,j)+phi(i-1,j))/h^2-...
                        (phi(i,j+1)-2*phi(i,j)+phi(i,j-1))/h^2;
                end
            end
        end
    end
elseif strcmp(opt,'SOR')
    rho=0.5*(cos(pi/M)+cos(pi/N));
    w=2/(1+sqrt(1-rho^2));
    for k=1:kmax
        for i=2:N %Only the middle points will change value
            for j=2:N
                phi(i,j)=phi(i,j)+w*(0.25*(phi(i+1,j)+phi(i-1,j)...
                    +phi(i,j+1)+phi(i,j-1)-h^2*f(x(i),x(j)))-phi(i,j));
            end
        end
        if k == iter(l)
            PHI(:,:,l)=phi;
            l=l+1;
        end
        if k<=iter_res
            for i=2:N
                for j=2:N
                    res(i,j,k+1)=f(x(i),x(j))-(phi(i+1,j)-2*phi(i,j)+phi(i-1,j))/h^2-...
                        (phi(i,j+1)-2*phi(i,j)+phi(i,j-1))/h^2;
                end
            end
        end
    end
end
for i=2:N
    for j=2:N
        res(i,j,1)=f(x(i),x(j))-(phi_0(i+1,j)-2*phi_0(i,j)+phi_0(i-1,j))/h^2-...
                    (phi_0(i,j+1)-2*phi_0(i,j)+phi_0(i,j-1))/h^2;
    end
end
end


function L=ErrorNorm(v,v_app,k)
    N=length(v);
    if k==inf
        L=max(abs(v-v_app));
    elseif k==1
        L=sum(abs(v-v_app))/N;
    elseif k==2
        L=sqrt(sum((v-v_app).^2)/N);
    end
end
```

Problem 1

In this problem we solve the differential equation

$\frac{\partial^2\phi}{\partial x^2} = \cos(\pi x)\left(-\frac{\pi^2}{4}x^3 - 2\pi^2x^2 + \frac{3}{2}x + 4\right) - \sin(\pi x)\left(\frac{3}{2}\pi x^2 + 8\pi x\right),$

on the interval $x \in [-1, 1]$ with boundary conditions $\phi(x = -1) = \phi(x = 1) = 0$ and 257 equally spaced nodes. We use

$\phi^{(0)}(x) = \frac{1}{2}\sin(\pi x) - \frac{1}{10}\sin(32\pi x),$

as initial guess. For this problem we use the function *IterativeMethods1D* to obtain the solution $\phi(x)$ on at certain number of iterations and to obtain the residuals of every iteration up to a certain number introduced as input. The function itself is explained in above in its own file.

We start by defining the anonymous functions f(x), that represents the right hand side of the differential equation, and phi_0(x), that represents the initial guess. This two anonymous functions will be inputs for the IterativeMethods1D function. We also define the rest of the variables that we need to meet the requirements of the problem (explained below).

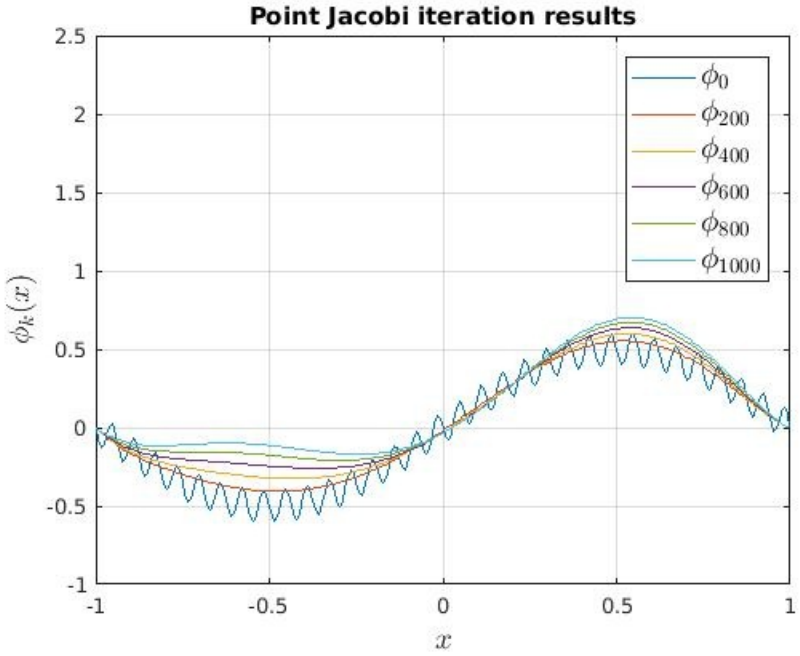
```
clear all; close all; format long; clc
labelfontsize=14;
f = @(x) cos(pi*x).*(-pi^2*x.^3/4-2*pi^2*x.^2+3*x/2+4)-sin(pi*x).*(3*pi*x.^2/2+8*pi*x);
phi_0 = @(x) 0.5*sin(pi*x)-0.1*sin(32*pi*x);
N=256;
L=2;
h=L/N;
x=linspace(-1,1,N+1)';
iter=[200 400 600 800 1000]; % Number of iterations to store the solution.
iter_res=400;  % Number of iterations until which we store the residuals.
```

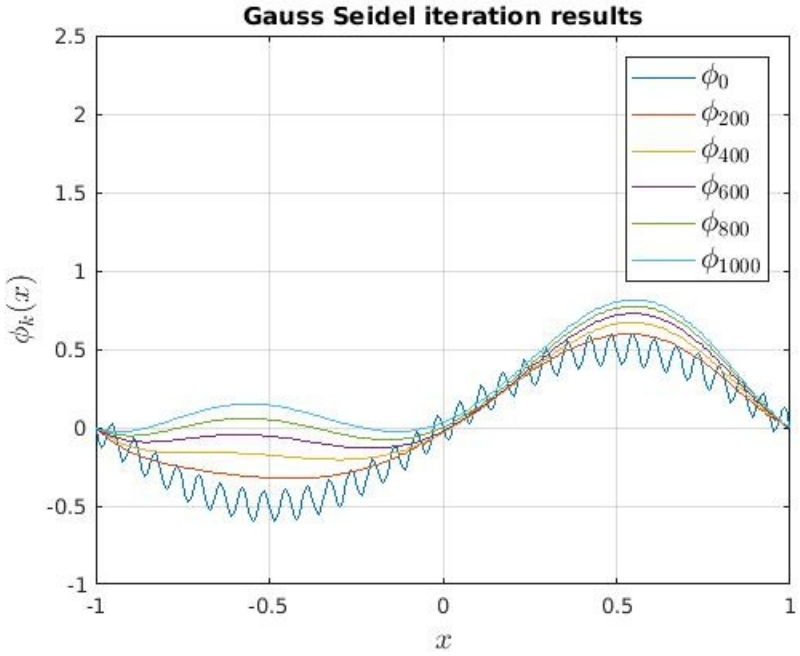
We calculate the solution using different iterative methods and we store the results at the iterations given by the problem in phi_PJ (Point Jacobi), phi_GS (Gauss Seidel) and phi_SOR (SOR). We store the residuals of every iteration until the iteration 400 in the corresponding variables.

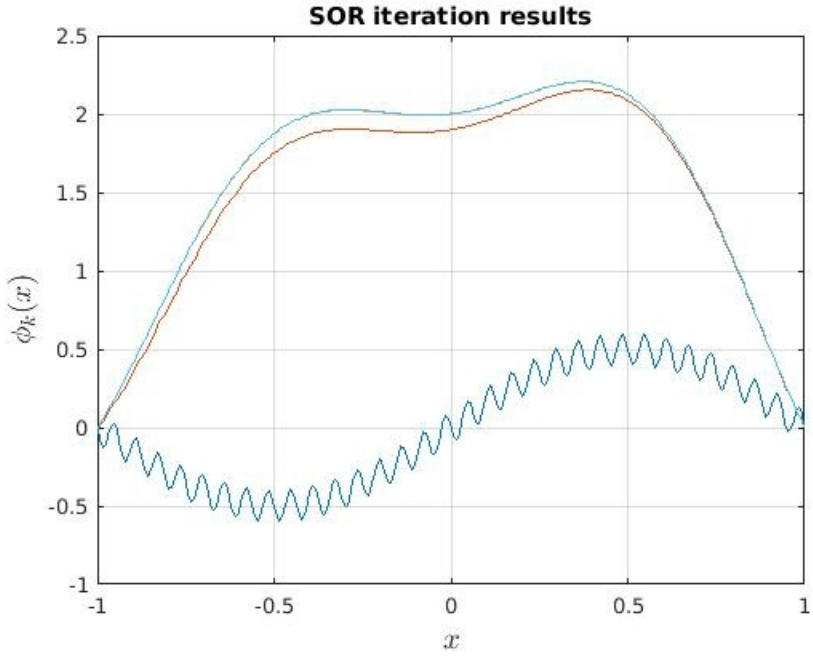
```
[phi_PJ,r_PJ]=IterativeMethods1D(f,phi_0,x,h,N,'PJ',iter,iter_res);
[phi_GS,r_GS]=IterativeMethods1D(f,phi_0,x,h,N,'GS',iter,iter_res);
[phi_SOR,r_SOR]=IterativeMethods1D(f,phi_0,x,h,N,'SOR',iter,iter_res);
```

The following lines of code plot the solutions at different number of iterations as indicated from the problem.

```
figure
plot(x,phi_0(x))
hold on
plot(x,phi_PJ)
grid on
axis([-1 1 -1 2.5])
legend({'$\phi_{0}$','$\phi_{200}$','$\phi_{400}$','$\phi_{600}$','$\phi_{800}$','$\phi_{1000}$'},...
    'fontsize',14,'interpreter','latex')
xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
ylabel('$\phi_k(x)$','fontsize',labelfontsize,'interpreter','latex')
title('Point Jacobi iteration results')

figure
plot(x,phi_0(x))
hold on
plot(x,phi_GS)
grid on
axis([-1 1 -1 2.5])
legend({'$\phi_{0}$','$\phi_{200}$','$\phi_{400}$','$\phi_{600}$','$\phi_{800}$','$\phi_{1000}$'},...
    'fontsize',14,'interpreter','latex')
xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
ylabel('$\phi_k(x)$','fontsize',labelfontsize,'interpreter','latex')
title('Gauss Seidel iteration results')

figure
plot(x,phi_0(x))
hold on
plot(x,phi_SOR)
grid on
axis([-1 1 -1 2.5])
xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
ylabel('$\phi_k(x)$','fontsize',labelfontsize,'interpreter','latex')
title('SOR iteration results')
```







Taking a look at the initial guess we see how it is composed of two sines with two different wave numbers, one of them higher than the other. We can see how the high wave number component converges almost inmediately, with 200 iterations is not in the plots anymore. However, the low wave

number component takes a higher number of iterations to converge. As we saw in class, we want a small M/k factor for fast convergence. The Gauss Seidel method seems a bit faster than Point Jacobi method. The SOR method is significantly faster than the other two. All the figures follow the same code of colors, I didn't put the legend in the third figure because it is in the way of the graph. We can see how the lines of the different iterations are coinciding, we see the initial guess, iteration 200 and all of the following iterations coincide. The SOR is, as expected the fastest method of the three. The optimum over-relaxation factor

$$\omega = \frac{2}{1+\sqrt{1-\cos^2\left(\frac{\pi}{N}\right)}}$$

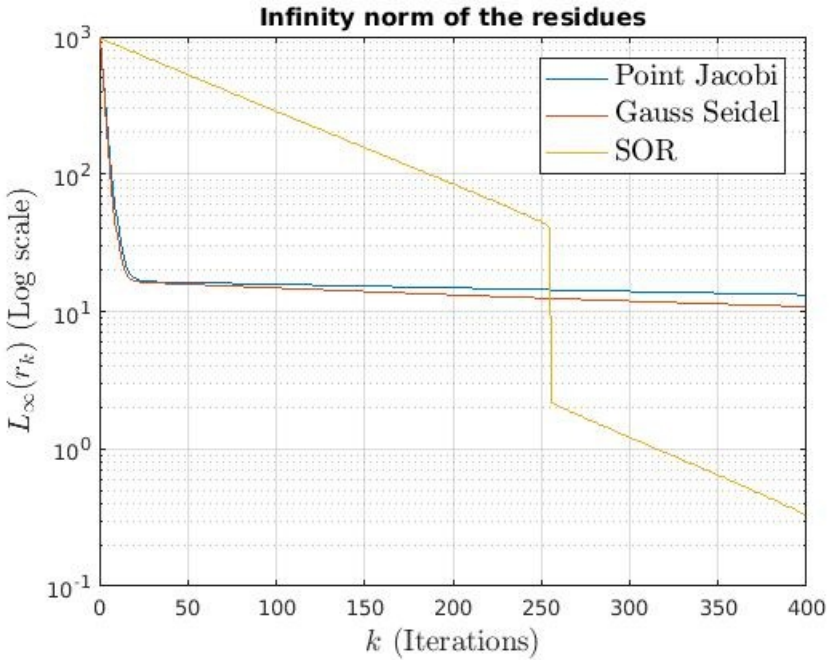
is

```
omega=2/(1+sqrt(1-(cos(pi/N))^2))
```

```
omega =

   1.975754453579712
```

To finish with this problem, we show a log-linear plot of the inifinity norm of the residual vs the number of iterations. We see how Point Jacobi and Gauss Seidel have similar behaviour, being Gauss Seidel a bit faster (the residues decrease faster). Both Gauss Seidel and Point Jacobi have a fast convergence at low number of iterations, probably due to the fast convergence of the high wave number component of the initial guess. We see how the SOR method is not necessarily faster at the beginning. However, this method has sudden drops on the residues that makes the method converge faster than the other two, even when it started slower. It would take many more iterations than 400 for the other two mehtods to reach a residual infinity norm of 1 (I've tried up to 5000 and they don't reach it).

```
figure
iterr=0:iter_res;
semilogy(iterr,max(abs(r_PJ)))
grid on
hold on
semilogy(iterr,max(abs(r_GS)))
semilogy(iterr,max(abs(r_SOR)))
legend({'Point Jacobi','Gauss Seidel','SOR'},'fontsize',14,'interpreter','latex')
xlabel('$k$ (Iterations)','fontsize',labelfontsize,'interpreter','latex')
ylabel('$L_{\infty}(r_k)$  (Log scale)','fontsize',labelfontsize,'interpreter','latex')
title('Infinity norm of the residues')
```



Problem 2

In this problem we solve the differential equation

$$\begin{aligned}\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2} &= \sqrt{y+1}\sin\left(\tfrac{\pi}{2}(y+1)\right)\left[-\pi^2x^3\cos\left(\tfrac{\pi}{2}x\right)-6\pi x^2\sin\left(\tfrac{\pi}{2}x\right)+12x\cos\left(\tfrac{\pi}{2}x\right)\right]\\ &\quad -x^3\cos\left(\tfrac{\pi}{2}x\right)\left[\frac{\sin\left(\frac{\pi}{2}(y+1)\right)}{2(y+1)^{3/2}}-\frac{\pi\cos\left(\frac{\pi}{2}(y+1)\right)}{\sqrt{y+1}}\right]\end{aligned}$$

on the interval $x\in[-1,1]$, $y\in[-1,1]$, with zero boundary conditions and 81 equally spaced nodes. We use

$$\phi^{(0)}(x,y)=\tfrac{2}{5}\sin(8\pi x)\sin(2\pi y),$$

as initial guess. For this problem we use the function *IterativeMethods2D* to obtain the solution $\phi(x,y)$ on at certain number of iterations and to obtain

the residuals of every iteration up to a certain number introduced as input. The function itself is explained in above in its own file.

We start by defining the anonymous functions f(x,y) that represents the right hand side of the differential equation, and phi_0(x,y), that represents the initial guess. This two anonymous functions will be inputs for the IterativeMethods2D function. We also define the rest of the variables that we need to meet the requirements of the problem (explained below).

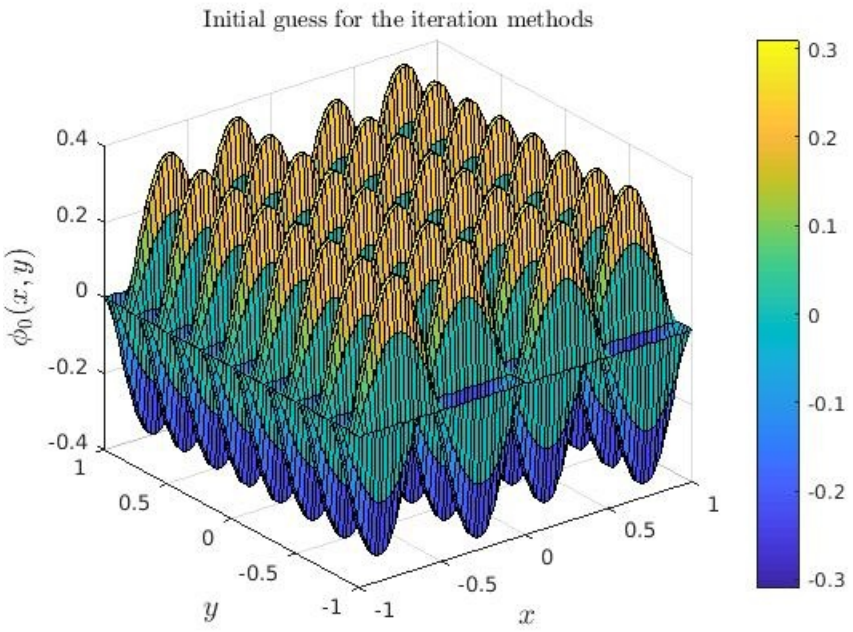
```
clear all; close all; format long; clc
labelfontsize=14;
f = @(x,y) sqrt(y+1).*sin(pi*(y+1)/2).*(-pi^2*x.^3.*cos(pi*x/2)-6*pi*x.^2.*sin(pi*x/2)+12*x.*cos(pi*x/2))...
    -x.^3.*cos(pi*x/2).*(sin(pi*(y+1)/2)./(2*(y+1).^1.5)-pi*cos(pi*(y+1)/2)./sqrt(y+1));
phi_0 = @(x,y) 2*sin(8*pi*x).*sin(2*pi*y)/5;
N=80;
x=linspace(-1,1,N+1)';
h=x(2)-x(1);
iter=[10 20 100 400]; % Number of iterations to store the solution.
iter_res=400;  % Number of iterations until which we store the residuals.
```

We calculate the solution using different iterative methods and we store the results at the iterations given by the problem in phi_GS (Gauss Seidel) and phi_SOR (SOR). We store the residuals of every iteration until the iteration 400 in the corresponding variables.

```
[phi_GS,r_GS_o]=IterativeMethods2D(f,phi_0,x,h,N,N,'GS',iter,iter_res);
[phi_SOR,r_SOR_o]=IterativeMethods2D(f,phi_0,x,h,N,N,'SOR',iter,iter_res);
```

The following lines of code show the surf plots of the solutions at different number of iterations as indicated from the problem. Starting by the initial guess we see the big amount of waves present.

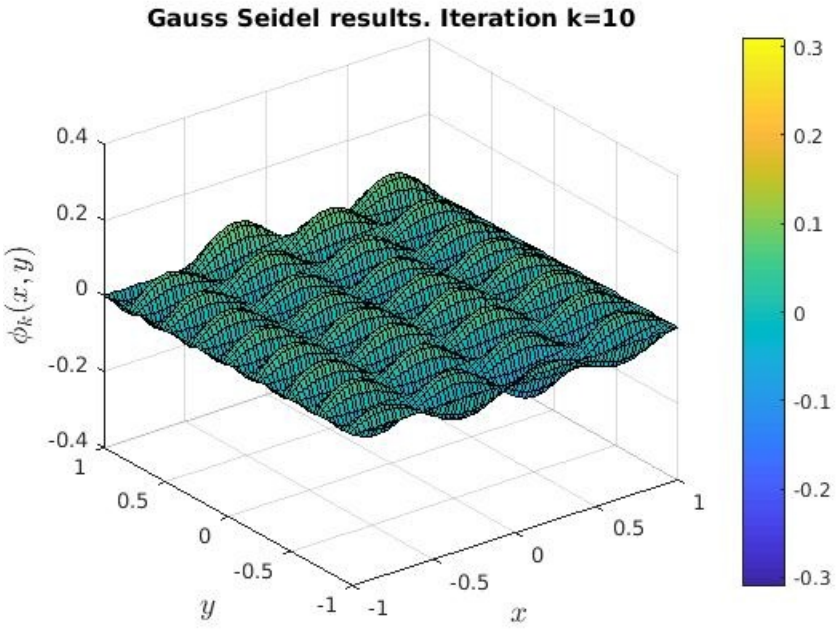
```
phi_0m=zeros(N+1,N+1);
for i=1:N+1 %Only the middle points will change value
    for j=1:N+1
        phi_0m(i,j)=phi_0(x(i),x(j));
    end
end
% figure('units','normalized','outerposition',[0 0 1 1])
figure
surf(x,x,phi_0m)
zlim([-0.4 0.4])
colorbar
caxis([-0.31 0.31])
xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
ylabel('$y$','fontsize',labelfontsize,'interpreter','latex')
zlabel('$\phi_0(x,y)$','fontsize',labelfontsize,'interpreter','latex')
title('Initial guess for the iteration methods','interpreter','latex')
```

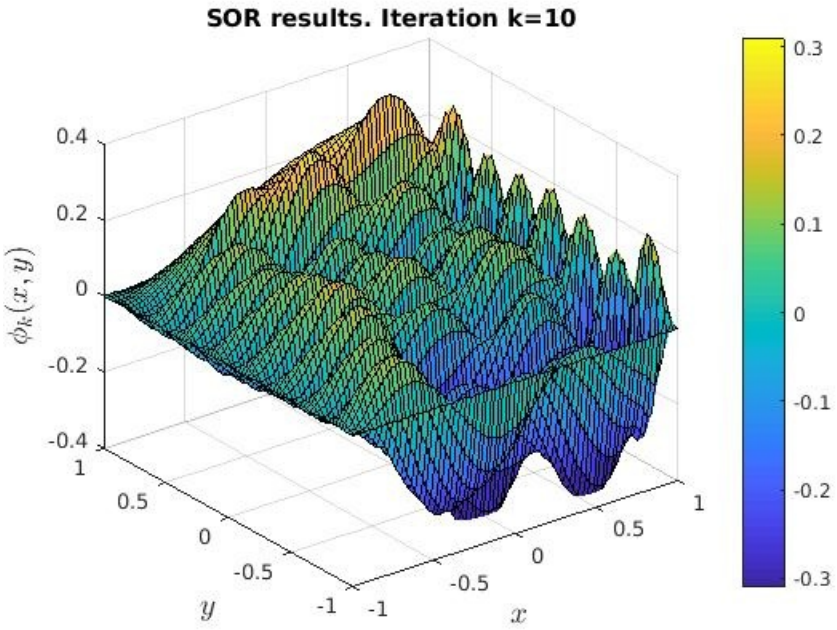


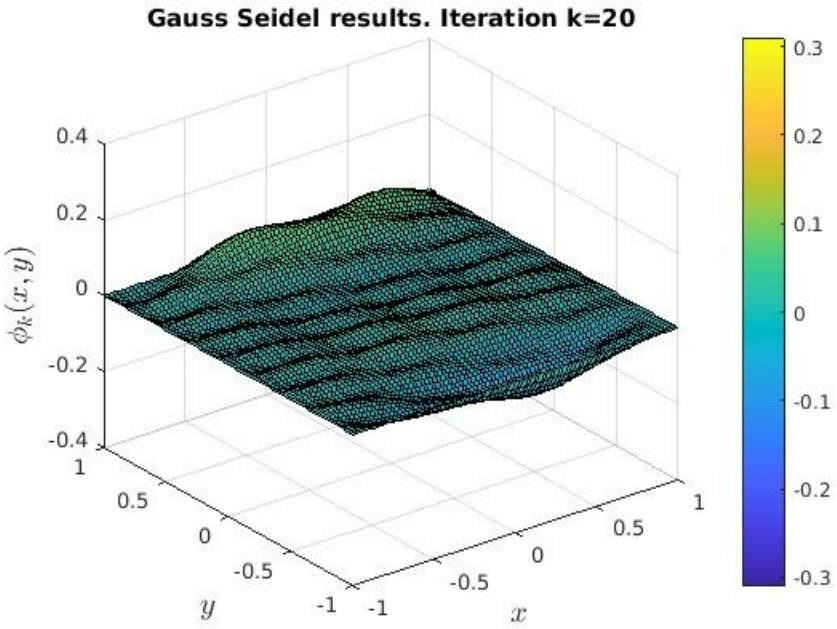
In the following figures we see how the Gauss Seidel dumps the waves inmediately in the first 10-20 iterations while SOR method is a bit caotic. However, at iteration 100, SOR seems to have found the solution while Gauss Seidel still needs more iterations. Both methods seem to agree in the shape of the solution (at least to the naked eye) at 400 iterations. We see how SOR, although starting much worse, reached the solution much faster. We are going to see this again when we look at the residues.

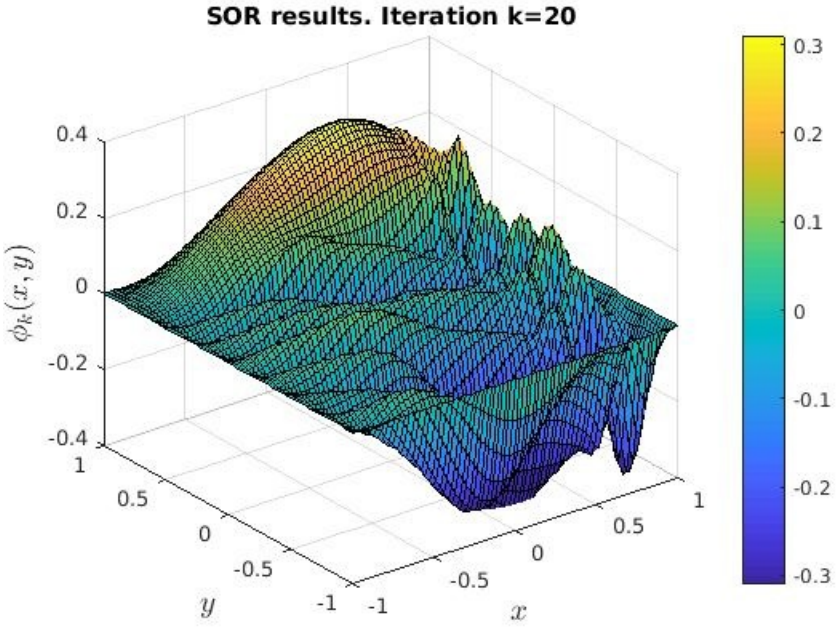
```
for i=1:4
    figure
    surf(x,x,phi_GS(:,:,i))
```

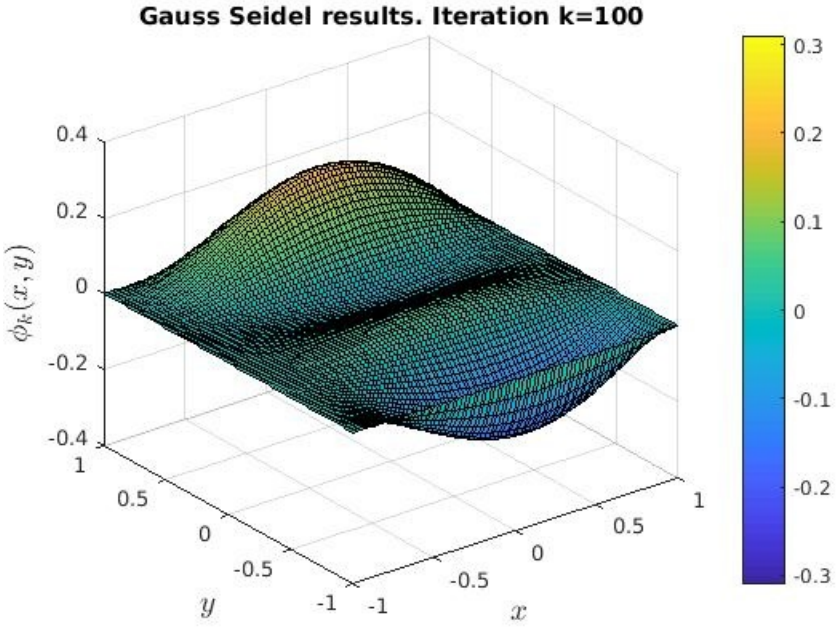
```
    zlim([-0.4 0.4])
    colorbar
    caxis([-0.31 0.31])
    xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
    ylabel('$y$','fontsize',labelfontsize,'interpreter','latex')
    zlabel('$\phi_k(x,y)$','fontsize',labelfontsize,'interpreter','latex')
    title(['Gauss Seidel results. Iteration k=',num2str(iter(i))])

    figure
    surf(x,x,phi_SOR(:,:,i))
    zlim([-0.4 0.4])
    colorbar
    caxis([-0.31 0.31])
    xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
    ylabel('$y$','fontsize',labelfontsize,'interpreter','latex')
    zlabel('$\phi_k(x,y)$','fontsize',labelfontsize,'interpreter','latex')
    title(['SOR results. Iteration k=',num2str(iter(i))])
end
```

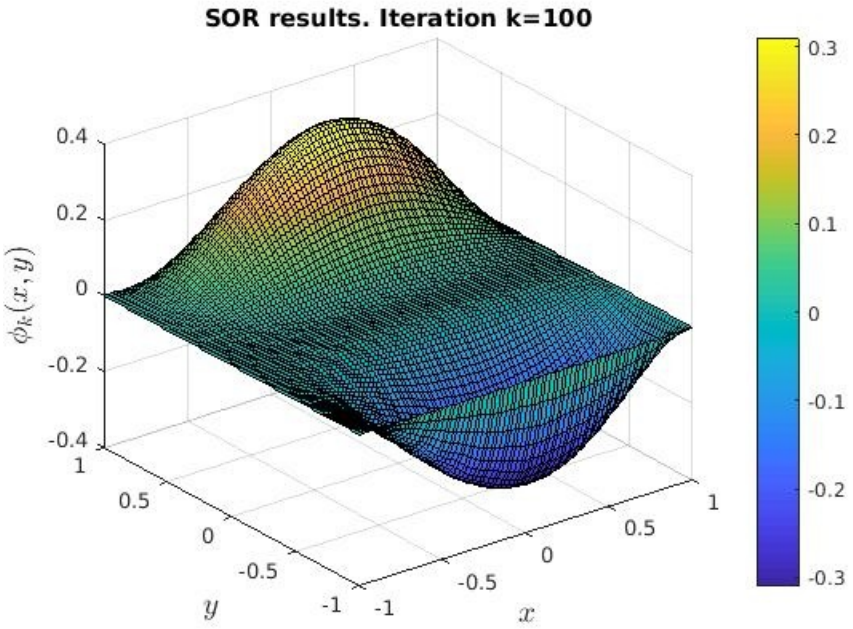


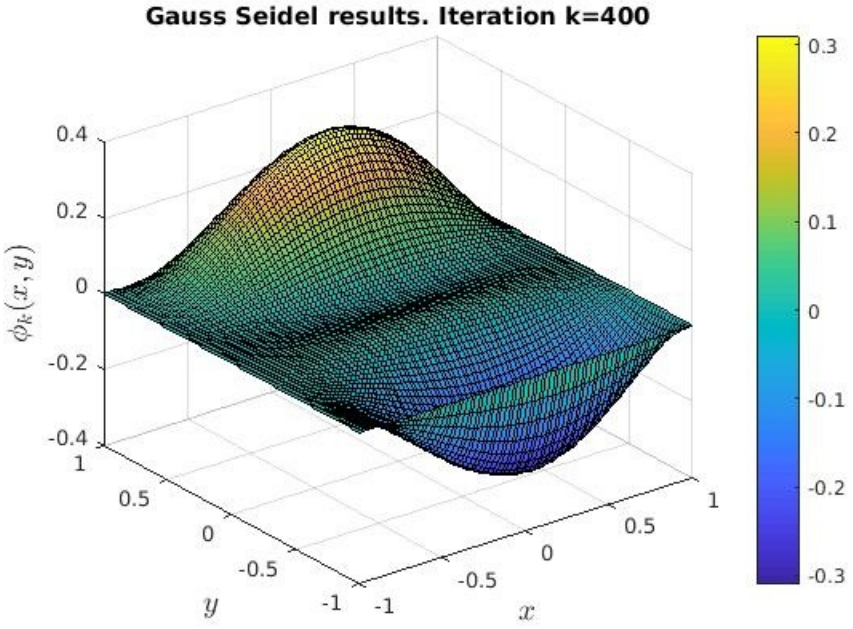


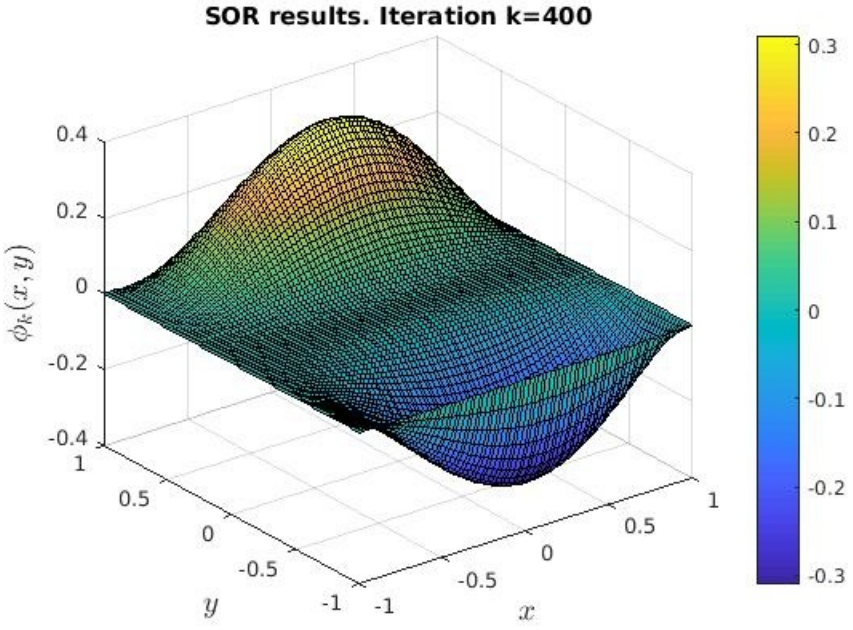

Gauss Seidel results. Iteration k=20
$\phi_k(x,y)$
0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
y
-1
-0.5
0
0.5
1
x
0.3
0.2
0.1
0
-0.1
-0.2
-0.3


SOR results. Iteration k=20
$\phi_k(x,y)$
0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
y
-1
-0.5
0
0.5
1
x
0.3
0.2
0.1
0
-0.1
-0.2
-0.3


Gauss Seidel results. Iteration k=100
$\phi_k(x,y)$
0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
y
-1
-0.5
0
0.5
1
x
0.3
0.2
0.1
0
-0.1
-0.2
-0.3



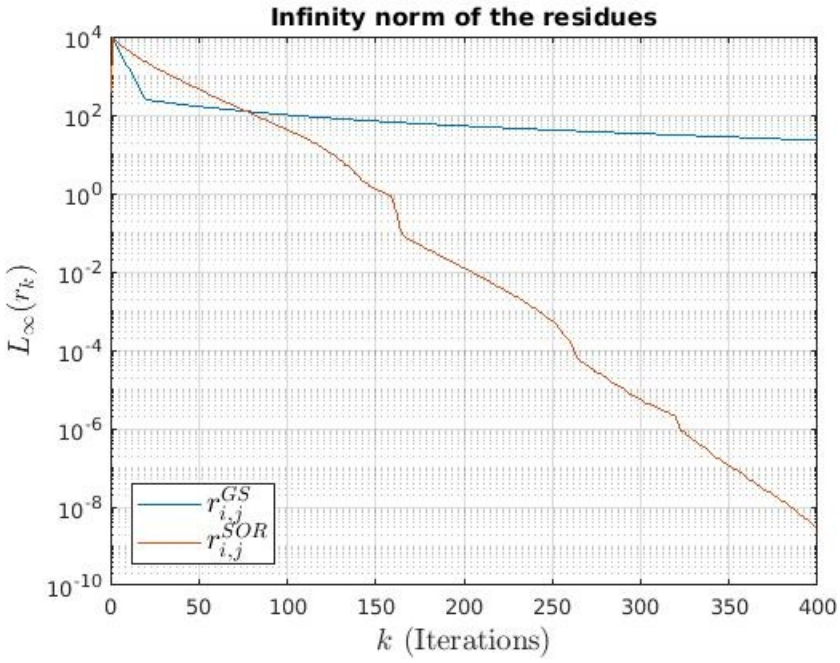




In the following figure we plot the infinity norm of the residues against the number of iterations. We can see, as well as in Problem 1, how the Gauss Seidel method's residues go down faster at the beginning but are rapidly over the residues of the SOR method (at iteration 77). From that point on, the

SOR method has residues much smaller than the Gauss Seidel method, this agrees with our discussion about the solutions above. The SOR method reaches the solution faster and it is able to obtain more precised solution using a reasonable number of iterations. As we see in the figure, the Gauss Seidel would need a huge number of iterations in order to match the residues of the SOR method at 400 iterations.

```
r_GS=zeros(size(r_GS_o,3),1);
r_SOR=zeros(size(r_SOR_o,3),1);
for i=1:size(r_GS_o,3)
    r_GS(i)=norm(r_GS_o(:,:,i));
    r_SOR(i)=norm(r_SOR_o(:,:,i));
end
iterr=0:400;
figure
semilogy(iterr,r_GS)
hold on
semilogy(iterr,r_SOR)
grid on
legend({'$r_{i,j}^{GS}$','$r_{i,j}^{SOR}$'},'fontsize',14,'interpreter','latex','location','southwest')
xlabel('$k$ (Iterations)','fontsize',labelfontsize,'interpreter','latex')
ylabel('$L_{\infty}(r_k)$','fontsize',labelfontsize,'interpreter','latex')
title('Infinity norm of the residues')
```



As we can see, it happens something very similar to the 1D problem. We have a quick convergence at low number of iterations for the Gauss Seidel method, probably due to the quick convergence of the high wave number component of the initial guess. From that point the Gauss Seidel becomes slow and its overcome by the SOR method. The latter reaches a grade of precision that would take a very high number of iterations for the Gauss Seidel method to achieve.

The optimum over-relaxation factor

$$\omega = \frac{2}{1+\sqrt{1-\rho_{PJ}^2}},$$

where

$$\rho_{PJ} = \frac{1}{2}\left[\cos^2\left(\frac{\pi}{N}\right) + \cos^2\left(\frac{\pi}{M}\right)\right].$$

Note that in this problem $N = M = 80$.

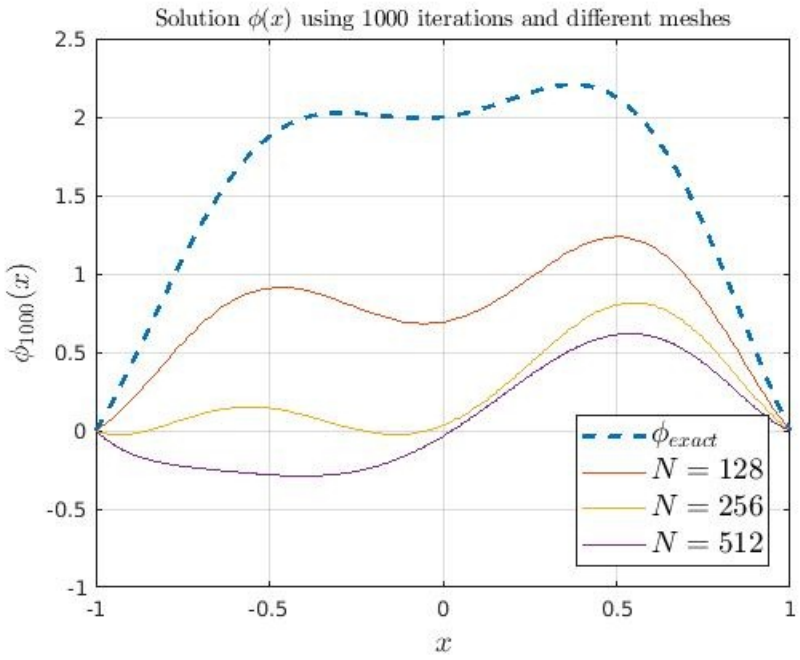
```
rho=0.5*(cos(pi/N)+cos(pi/N));
omega=2/(1+sqrt(1-rho^2))
%
```

```
omega =

   1.924446581761859
```

Problem 3

In this problem we have the exact solution to the differential equation of problem 1 and we are going to obtain the different error norms and the observed order of accuracy by obtaining the error made using the Gauss Seidel method with different number of elements and a 1000 iterations.

```
clear all; close all; format long; clc
labelfontsize=14;
f = @(x) cos(pi*x).*(-pi^2*x.^3/4-2*pi^2*x.^2+3*x/2+4)-sin(pi*x).*(3*pi*x.^2/2+8*pi*x);
phi_exact = @(x) 2+x/4+cos(pi*x).*(x.^3/4+2*x.^2);
xx=linspace(-1,1,1000);
phi_0 = @(x) 0.5*sin(pi*x)-0.1*sin(32*pi*x);
N=[128 256 512];
T=zeros(length(N),7);
T(1,5:7)=NaN;
figure(1)
plot(xx,phi_exact(xx),'--','linewidth',2)
hold on
grid on
axis([-1 1 -1 2.5])
iter=1000;
iter_res=0;
for i=1:length(N)
    x=linspace(-1,1,N(i)+1)';
    h(i)=x(2)-x(1);
    phi_GS=IterativeMethods1D(f,phi_0,x,h(i),N(i),'GS',iter,iter_res);
    e_inf=ErrorNorm(phi_exact(x),phi_GS,inf);
    e_1=ErrorNorm(phi_exact(x),phi_GS,1);
    e_2=ErrorNorm(phi_exact(x),phi_GS,2);
    T(i,1:4)=[N(i) e_inf e_1 e_2];
    figure(1)
    plot(x,phi_GS)
end
legend({'$\phi_{exact}$','$N=128$','$N=256$','$N=512$'},...
    'fontsize',14,'interpreter','latex','location','southeast')
xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
ylabel('$\phi_{1000}(x)$','fontsize',labelfontsize,'interpreter','latex')
title('Solution $\phi(x)$ using 1000 iterations and different meshes','interpreter','latex')
```



We can see in the figure how the solutions get more distant from the exact solution as we increase the number of points N. As we saw in class, for fast convergence we need a N/k factor small. Having the same wave number, increasing the number of nodes we are slowing down our convergence. Thus, it makes sense that we see negative obeserved order of convergence as the more nodes we use the further we are from the exact solution for the same number of iterations. We can solve this by increasing the number of iterations with the number of points, as I do in problem 4.

```
for i=2:length(N)
    T(i,5)=log(T(i,2)/T(i-1,2))/log(h(i)/h(i-1));
    T(i,6)=log(T(i,3)/T(i-1,3))/log(h(i)/h(i-1));
    T(i,7)=log(T(i,4)/T(i-1,4))/log(h(i)/h(i-1));
end
T=array2table(T,'VariableNames',{'N','L_inf','L_1','L_2','o_inf','o_1','o_2'});
T(:,1:4)
T(:,[1 5 6 7])
```

```
ans =

  3×4 table

    N          L_inf                L_1                 L_2
    ___    _____________    _____________    _____________
```

```
    128    1.31006298137877    0.827967093869307    0.923154394414248
    256    2.02834084034356     1.32890048697402      1.4654123257523
    512    2.30712568638202     1.52079923935873      1.6594655203402


ans =

  3×4 table

     N           o_inf                 o_1                 o_2
    ___    _________________    _________________    _________________

    128                  NaN                  NaN                  NaN
    256    -0.630663930622761    -0.682587737733187    -0.666662796542607
    512      -0.1857964988169    -0.194596641250319    -0.179411998515853
```

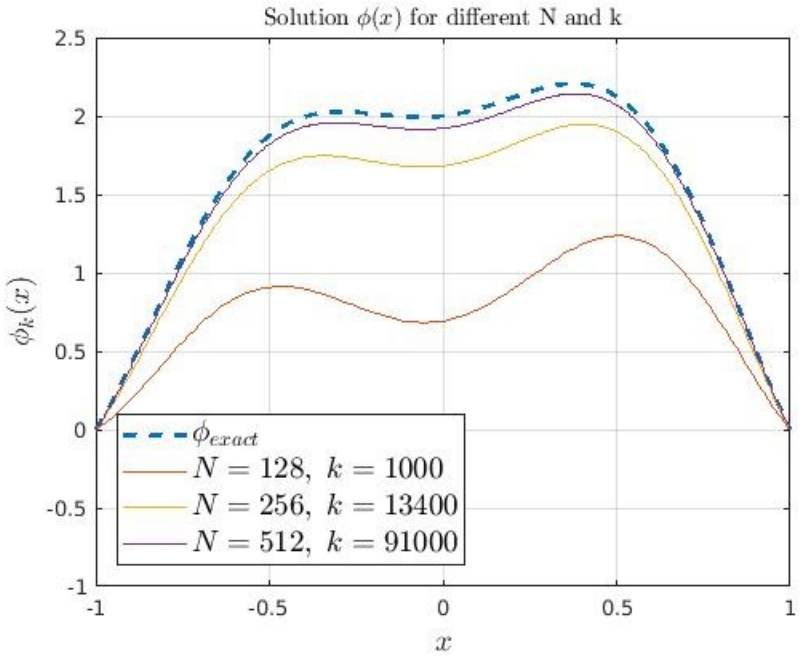
Problem 4

As we have discussed above, the higher number of points, mantaining the same wave number, the slower the convergence. That is causing the negative observed order of convergence. We can obtain the right second order observed order of convergence simply making sure that the truncation error is dominant versus the iterative error, i.e., by increasing the number of iterations. In this case we use 1000 iterations for N=128, 13400 iterations for N=256 and 91000 iterations for N=512. We see in the figure how it takes 91 thousand iterations to be close to the solution with 512 points. In the table we can see how we have obtained the observed order of convergence equal to 2 for all norms.

```
clear all; close all; format long; clc
labelfontsize=14;
f = @(x) cos(pi*x).*(-pi^2*x.^3/4-2*pi^2*x.^2+3*x/2+4)-sin(pi*x).*(3*pi*x.^2/2+8*pi*x);
phi_exact = @(x) 2+x/4+cos(pi*x).*(x.^3/4+2*x.^2);
xx=linspace(-1,1,1000);
phi_0 = @(x) 0.5*sin(pi*x)-0.1*sin(32*pi*x);
N=[128 256 512];
T=zeros(length(N),7);
T(1,5:7)=NaN;
figure(1)
plot(xx,phi_exact(xx),'--','linewidth',2)
hold on
grid on
axis([-1 1 -1 2.5])
iter=[1e3 13.4e3 91e3];
iter_res=0;
for i=1:length(N)
    x=linspace(-1,1,N(i)+1)';
    h(i)=x(2)-x(1);
    phi_GS=IterativeMethods1D(f,phi_0,x,h(i),N(i),'GS',iter(i),iter_res);
    e_inf=ErrorNorm(phi_exact(x),phi_GS,inf);
    e_1=ErrorNorm(phi_exact(x),phi_GS,1);
    e_2=ErrorNorm(phi_exact(x),phi_GS,2);
    T(i,1:4)=[N(i) e_inf e_1 e_2];
    figure(1)
    plot(x,phi_GS)
end
legend({'$\phi_{exact}$','$N=128,~k=1000$','$N=256,~k=13400$','$N=512,~k=91000$'},...
    'fontsize',14,'interpreter','latex','location','southwest')
xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
ylabel('$\phi_k(x)$','fontsize',labelfontsize,'interpreter','latex')
title('Solution $\phi(x)$ for different N and k','interpreter','latex')
for i=2:length(N)
    T(i,5)=log(T(i,2)/T(i-1,2))/log(h(i)/h(i-1));
    T(i,6)=log(T(i,3)/T(i-1,3))/log(h(i)/h(i-1));
    T(i,7)=log(T(i,4)/T(i-1,4))/log(h(i)/h(i-1));
end
T=array2table(T,'VariableNames',{'N','L_inf','L_1','L_2','o_inf','o_1','o_2'});
T(:,1:4)
T(:,[1 5 6 7])
```

```
ans =

  3×4 table

     N           L_inf                 L_1                 L_2
    ___    _________________    _________________    _________________

    128      1.31006298137877     0.827967093869307     0.923154394414248
    256     0.318482289599285     0.201899971798471     0.224713616758665
    512    0.0779819939863411    0.0495324951820341    0.0550752219098151


ans =

  3×4 table
```

N	o_inf	o_1	o_2
128	NaN	NaN	NaN
256	2.04035111774238	2.03593272227362	2.0384844050658
512	2.03000019846322	2.02719350598165	2.02861224081806



Published with MATLAB® R2017a