HOMEWORK 4 - FRANCISCO CASTILLO

Contents

Defined functions

```
function [phi, LinfR] = SOR (phi,f,h,nIterMax,eps)
    M = size(phi,1)-2;
    N = size(phi,2)-2;
    rho=0.5*(cos(pi/M)+cos(pi/N));
    w=2/(1+sqrt(1-rho^2));
    r = residual(phi,f,h); % residual of the initial guess
    LinfR=zeros(nIterMax,1);
    LinfR(1)=InfNorm(r);
    for n =1:nIterMax
        % Update the interior cells
        for i=2:M+1
            for j=2:N+1
                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(i,j))-phi(i,j));
            end
        end
        % Update the ghost cells
        phi(1,:) = phi(2,:);
        phi(M+2,:) = phi(M+1,:);
        phi(:,1) = phi(:,2);
        phi(:,M+2) = phi(:,M+1);
        % Calculate residuals
        r = residual(phi,f,h);
        LinfR(n+1,1) = InfNorm(r);
        if LinfR(n+1,1)<eps
            break
        end
    end
end


function phi = GaussSeidel (phi,f,h,n)
    for k=1:n
        M=length(phi)-2;
        % Update the interior cells
        for i=2:M+1
            for j=2:M+1
                phi(i,j)=0.25*(phi(i+1,j)+phi(i-1,j)...
                    +phi(i,j+1)+phi(i,j-1)-h^2*f(i,j));
            end
        end
        % Update the ghost cells
        phi(1,:) = phi(2,:);
        phi(M+2,:) = phi(M+1,:);
        phi(:,1) = phi(:,2);
        phi(:,M+2) = phi(:,M+1);
    end
end


function [phi,LinfR] = poisson (phi,f,h,nIterMax,eps)
    [M,N] = size(phi);
    r = residual(phi,f,h);
    LinfR=zeros(nIterMax,1);
    LinfR(1)=InfNorm(r);
    for n=1:nIterMax
        phi = multigrid(phi,f,h);
        r = residual(phi,f,h);
        LinfR(n+1,1) = InfNorm(r);
        if LinfR(n+1,1)<eps
            break
        end
    end
    LinfR(n+2:nIterMax)=[];
end


function phi = multigrid(phi,f,h)
    M=length(phi)-2;
    phi = GaussSeidel(phi,f,h,1);
    if M>2
        rh = residual(phi,f,h);
        r2h = restrict(rh);
        e2h = zeros(M/2+2,M/2+2);
        e2h = multigrid(e2h,r2h,2*h);
        eh = prolong(e2h);
        phi = phi+eh;
        phi = GaussSeidel(phi,f,h,1);
    end
end


function r = residual(phi,f,h)
    M = size(phi,1)-2;
%     M
    r= zeros(size(phi));
%     whos phi f r
%     keyboard
    % Calculate the interior cells
    for i=2:M+1
        for j=2:M+1
            r(i,j)=f(i,j)-(phi(i+1,j)-2*phi(i,j)+phi(i-1,j)+phi(i,j+1)-2*phi(i,j)+phi(i,j-1))/h^2;
        end
    end
    % calculate the ghost cells
    r(1,:) = r(2,:);
    r(M+2,:) = r(M+1,:);
    r(:,1) = r(:,2);
    r(:,M+2) = r(:,M+1);
end
```

```
function eh = prolong(e2h)
    M2h=size(e2h,1)-2;
    Mh=2*M2h;
    eh=zeros(Mh+2,Mh+2);
    e2h(:,[1 end])=[];
    e2h([1 end],:)=[];
    % Interior
    for i=1:M2h
        for j=1:M2h
            eh(2*i-1+1:2*i+1,2*j-1+1:2*j+1) = e2h(i,j);
        end
    end
    % Calculate the ghost cells
        eh(1,:) = eh(2,:);
        eh(Mh+2,:) = eh(Mh+1,:);
    eh(:,1) = eh(:,2);
        eh(:,Mh+2) = eh(:,Mh+1);
end


function r2h = restrict(rh)
    M=size(rh,1)-2;
    M2h=M/2;
    r2h=zeros(M2h+2);
    rh(:,[1 end])=[];
    rh([1 end],:)=[];
    for i=1:M2h
        for j=1:M2h
            r2h(i+1,j+1) = 0.25*sum(sum(rh(2*i-1:2*i,2*j-1:2*j)));
        end
    end
    % Update the ghost cells
        r2h(1,:) = r2h(2,:);
        r2h(M2h+2,:) = r2h(M2h+1,:);
    r2h(:,1) = r2h(:,2);
        r2h(:,M2h+2) = r2h(:,M2h+1);
end


function Linf = InfNorm(x)
    Linf= max(max(abs(x)));
end


function L=ErrorNorm2D(v,v_app,k)
    [M,N]=size(v);
    if k==inf
        L=max(max(abs(v-v_app)));
    elseif k==1
        L=sum(sum(abs(v-v_app)))/(M*N);
    elseif k==2
        L=sqrt(sum(sum((v-v_app).^2))/(M*N));
    end
end
```

Problem 1

First, I will talk about the function prolong. In the drawing attached we can see how the ghost cells of the coarser mesh are not numbered. This is because I get rid of them at the beginning of the function to work only with the inner values (which are the ones to prolong). I thought this was the most consistent with my code, I apologize if this causes the grader any inconvenience. Once we get rid of the ghost cells of the coarser mesh, we define the matrix of the finer mesh but including this time the ghost cells. I did this because I didn't want to deal with index shifting at the end of the function. Then, with the for loop I a associate the values of the coarser mesh cells to the fine mesh. The loops run going through the cells of the coarser mesh and those are associated to different cells of the finer mesh. The index form of the operator is

$$\epsilon^h_{2i:2i+1,2j:2j+1} = e^{2h}_{i,j}, \qquad i = 1, 2, ..., M^{2h}, j = 1, 2, ..., M^{2h}.$$

Once the inner cells values are placed, we impose the zero Neumann boundary conditions at the end of the function to end up with a finer mesh including the ghost cells with the same boundary conditions.

Now we talk about the restrict function. The procedure is very similar, I get rid of the ghost cells of the finer mesh, that's why they are not numbered in the drawing, to work only with the inner values of the finer mesh, the ones to restrict. We define the coarser mesh including the ghost cells, that is why they are numbered for the coarser mesh. In this case, the loops are going to run over indeces of the coarser mesh but we will shift the positions adding 1 to the indeces so we only fill the inner cell values. The index form of the restrict operator is

$$r^{2h}_{i+1,j+1} = \frac{1}{4} \sum_{l=2j-1}^{2j} \sum_{k=2i-1}^{2i} r^h_{k,l}, \qquad i = 1, 2, ..., M^{2h}, j = 1, 2, ..., M^{2h}.$$

which basically places the average of the four cells of the fincer mesh in the one cell of the coarser mesh placed on them. Once the inner cells values are placed, we impose the zero Neumann boundary conditions at the end of the function to end up with a finer mesh including the ghost cells with the same boundary conditions.

Problem 2

In this problem we solve the differential equation

$$\begin{aligned} \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = \quad & -9\pi x \cos(2\pi y^2) \left[2\sin(3\pi x^3) + 9\pi x^3 \cos(3\pi x^3)\right] \\ & -4\pi \cos(3\pi x^3) \left[\sin(2\pi y^2) + 4\pi y^2 \cos(2\pi y^2)\right], \end{aligned}$$

on the square domain $0 \leq x \leq 1, 0 \leq y \leq 1$ with Neumann boundary conditions

$$\frac{\partial \phi}{\partial n} = 0,$$

on all boundaries. We are told to use a cell centered mesh with $M = 256$ elements in both x- and y-directions. As initial guess we use

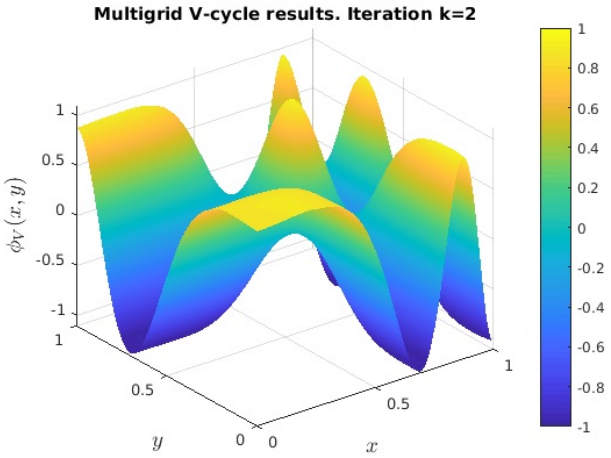
$$\phi^{(0)}(x, y) = 0.$$

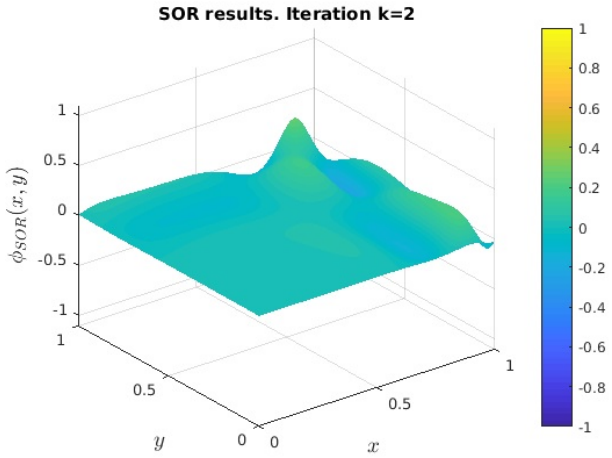
Since we are using a cell centered mesh we will need ghost cells around the domain to impose the boundary conditions, and since the Neumann boundary conditions are zero for every boundary, we will simply equal the solution at the ghost cells to the corresponding ones previously calculated with the iterative methods.

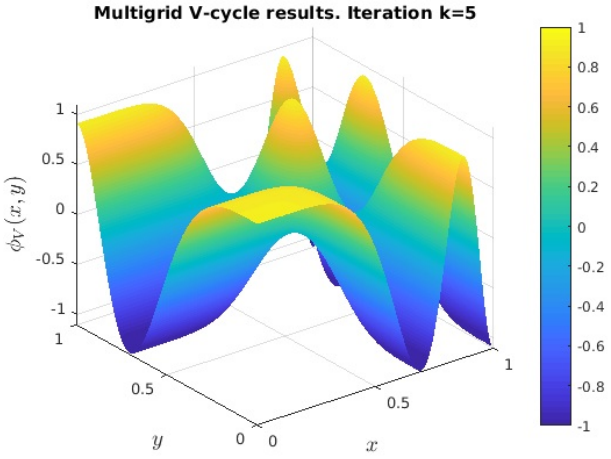
In the next lines we calculate the solution using the V-cycle multigrid method and the SOR method. The solution after 2, 5, 10 and 20 iterations will be ploted and discussed below.

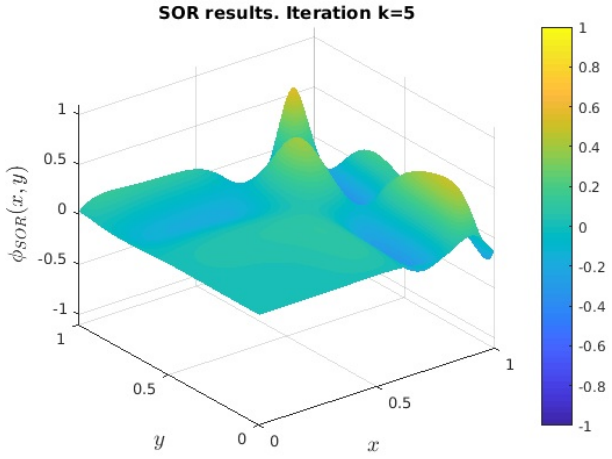
```
clear all;close all;format long;clc
labelfontsize=14;
M=256;
L=1;
h=L/M;
x=-h/2:h:1+h/2;
y=-h/2:h:1+h/2;
nIterMax=[2 3 5 10];

f=zeros(M+2,M+2);
for i=1:length(x)
    for j=1:length(y)
        f(i,j) = -9*pi*x(i).*cos(2*pi*y(j).^2).*(2*sin(3*pi*x(i).^3)+9*pi*x(i).^3.*cos(3*pi*x(i).^3))...
```

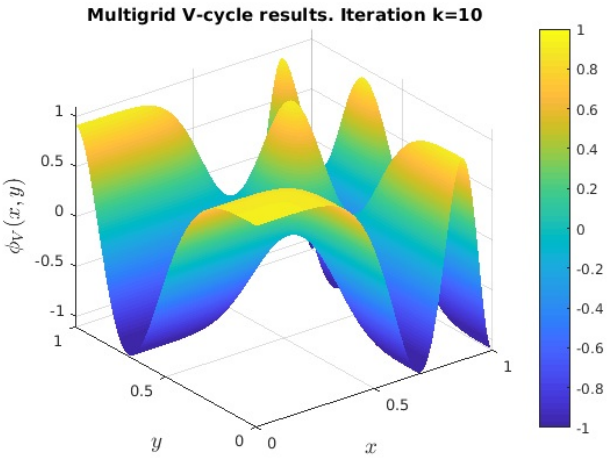
```
    -4*pi*cos(3*pi*x(i).^3).*(sin(2*pi*y(j).^2)+4*pi*y(j).^2.*cos(2*pi*y(j).^2));
    end
end
phi=zeros(M+2,M+2); % +2 to include the ghost cells
phi_SOR=zeros(M+2,M+2); % +2 to include the ghost cells

for n=1:length(nIterMax)
    phi = poisson(phi,f,h,nIterMax(n),eps);
    phiavg=0;
    for i=2:M+1
        for j=2:M+1
            phiavg=phiavg+phi(i,j);
        end
    end
    phiavg=phiavg/(M*M);

    figure
    surf(x(2:end-1),y(2:end-1),(phi(2:end-1,2:end-1)-phiavg)')
    zlim([-1.1 1.1])
    colorbar
    caxis([-1 1])
    xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
    ylabel('$y$','fontsize',labelfontsize,'interpreter','latex')
    zlabel('$\phi_V(x,y)$','fontsize',labelfontsize,'interpreter','latex')
    title(['Multigrid V-cycle results. Iteration k=',num2str(sum(nIterMax(1:n)))])
    shading interp


    phi_SOR = SOR(phi_SOR,f,h,nIterMax(n),eps);
    phiavg_SOR=0;
    for i=2:M+1
        for j=2:M+1
            phiavg_SOR=phiavg_SOR+phi_SOR(i,j);
        end
    end
    phiavg_SOR=phiavg_SOR/(M*M);

    figure
    surf(x(2:end-1),y(2:end-1),(phi_SOR(2:end-1,2:end-1)-phiavg_SOR)')
    zlim([-1.1 1.1])
    colorbar
    caxis([-1 1])
    xlabel('$x$','fontsize',labelfontsize,'interpreter','latex')
    ylabel('$y$','fontsize',labelfontsize,'interpreter','latex')
    zlabel('$\phi_{SOR}(x,y)$','fontsize',labelfontsize,'interpreter','latex')
    title(['SOR results. Iteration k=',num2str(sum(nIterMax(1:n)))])
    shading interp
end
```

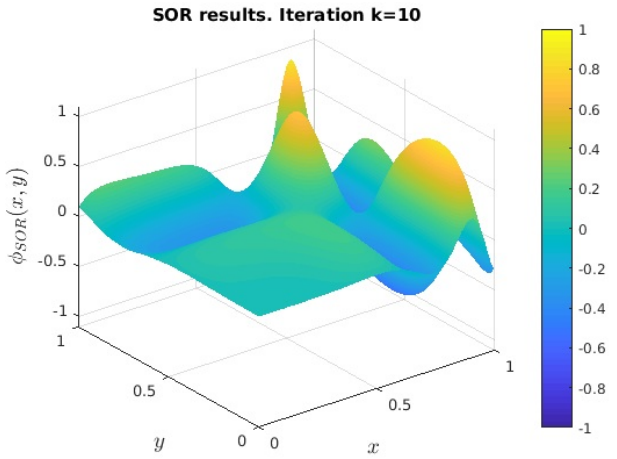


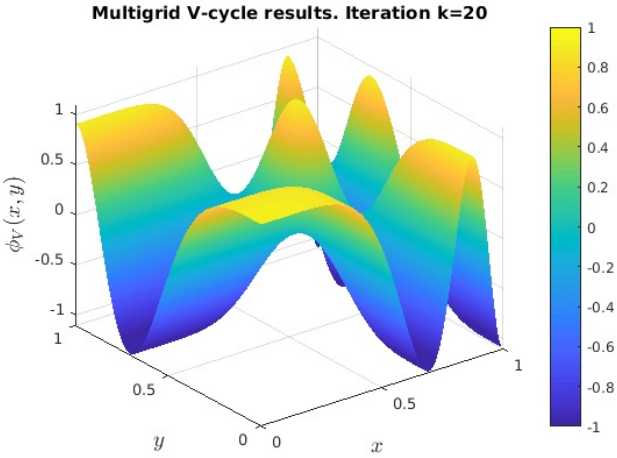


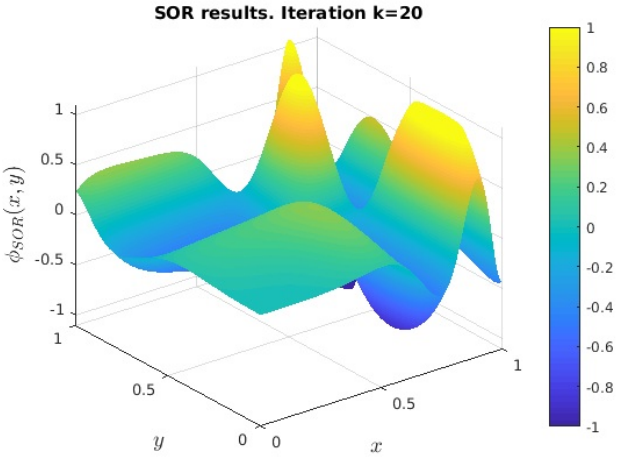






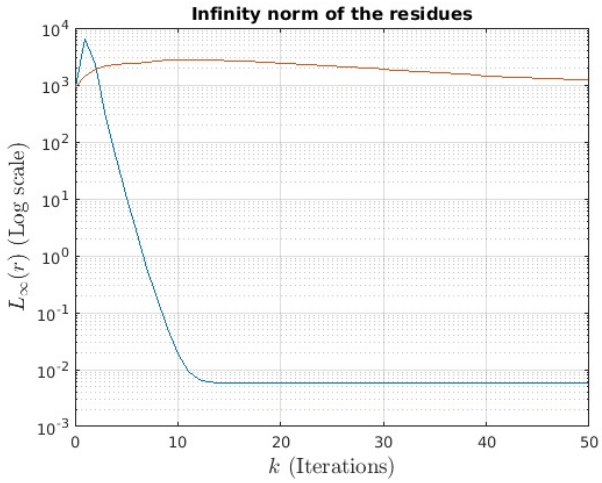






In the previous images we can see at first that the V-cycle multigrid method has converged almost inmediately, its speed is astonishing. The SOR method, that was itself very fast as we saw in the previous homework assignments, is close to be as fast as the multigrid method. We can see in the plots how after just 2 iterations the the solution obtained using the V-cycle multigrid method is mostly the same as the one obtained after 20 iterations, which indicates convergence. However, the SOR method has barely changed the solution from the zero initial guess after 2 iterations. The solution changes progressively and after 20 iterations we can see how it becomes a bit similar to the solution obtained using multigrid. This indicates that the SOR mehtod hasn't converged yet. We can compare the two methods more by looking at the infinity norm of the residuals vs the interation number, see next figure.

```
nIterMax=50;
phi=zeros(M+2,M+2); % +2 to include the ghost cells
phi_SOR=zeros(M+2,M+2); % +2 to include the ghost cells
[phi,LinfR] = poisson(phi,f,h,nIterMax,eps);
[phi_SOR,LinfR_SOR] = SOR(phi_SOR,f,h,nIterMax,eps);
iter=0:nIterMax;
figure
semilogy(iter,LinfR)
hold on
semilogy(iter,LinfR_SOR)
grid on
xlabel('$k$ (Iterations)','fontsize',labelfontsize,'interpreter','latex')
ylabel('$L_{\infty}(r)$  (Log scale)','fontsize',labelfontsize,'interpreter','latex')
title('Infinity norm of the residues')
```



We can see how the V-cycle multigrid mehtod's residues get smaller much faster than the SOR mehtod and convergence is acomplished after around 14 iterations. The residues of the SOR method are not close to reach the point at which the multigrid residues are. More precisely, the residues of the multigrid method are of the order of 10^{-2} after 15 iterations while the SOR residues are of the order of 10^3 after 50 iterations.

Problem 3

In this problem we obtain the partial derivatives of the solution ϕ, obtained in the previous problem, using a second order central differences at each cell center. We just have to apply the finite difference discretization formula for the inner cells. The boundary cells will use the values of the ghost cells to compute the values of ϕ_x and ϕ_y at the boundaries.

We are given the exact derivatives

$$\frac{\partial \phi}{\partial x} = -9\pi x^2 \cos(2\pi y^2)\sin(3\pi x^3),$$

and

$$\frac{\partial \phi}{\partial y} = -4\pi y \sin(2\pi y^2)\cos(3\pi x^3).$$

Thus, we can compute the error of our approximation. By taking the L_∞, L_1 and L_2 norms we can also compute the observed order of convergence, see tables below.

I run the following loop 4 times we different number of elements M. In each iteration, I prealocate the variables and I run the code of problem 2 to solve the PDE. Once I've done that I plot the residuals and store the norms of the error in matrices that will be converted in tables. I have chosen central finite differences to compute the derivatives,

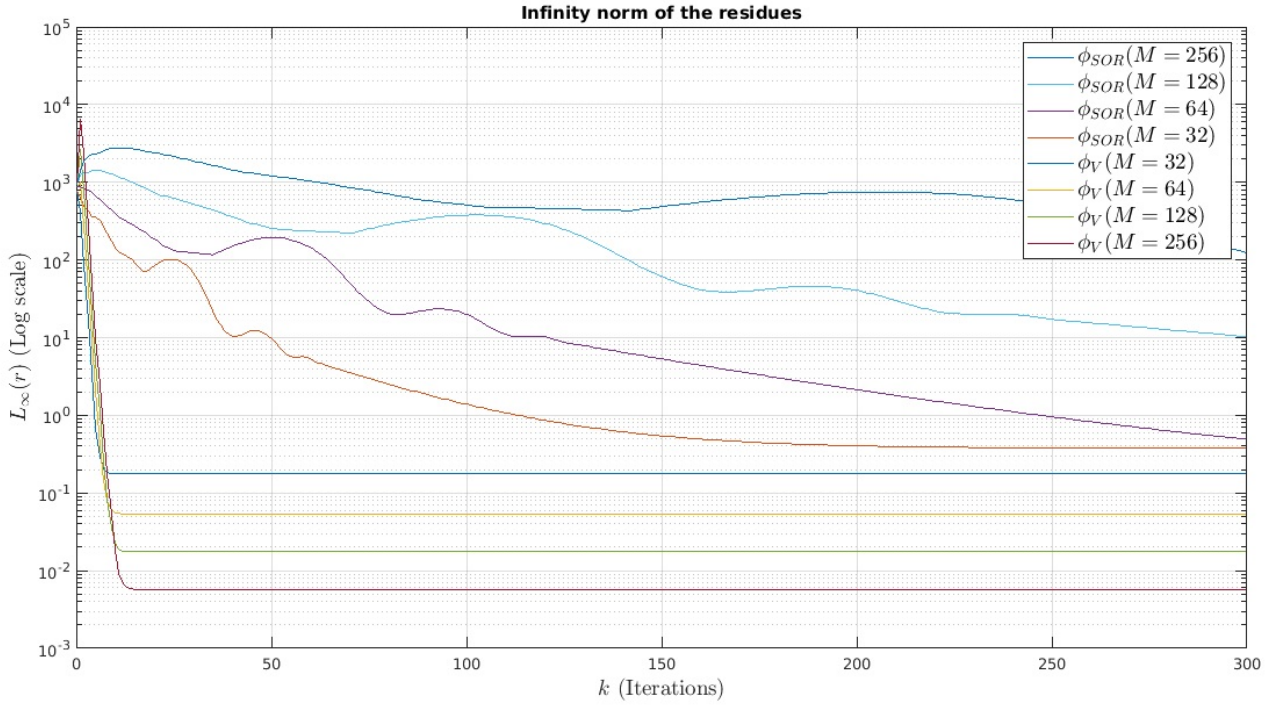
$$\left.\frac{\partial \phi}{\partial x}\right|_{x_{i,j}} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h},$$

and

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_{i,j}} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2h}.$$

We loop the previous discretizations for all the interior cells avoiding the ghost cells. To calculate the values of ϕ_x and ϕ_y at the boundary the scheme will use the ghost cells values.

```
clear all;close all;format long;clc
labelfontsize=14;
L=1;
M=[32 64 128 256];
H=L./M;
nIterMax = 300;
iter=0:nIterMax;
T_x=zeros(length(M),4);
T_y=zeros(length(M),4);
fig1 = figure('units','normalized','outerposition',[0 0 1 1]);
for n=1:length(M)
    h=H(n);
    x=-h/2:h:1+h/2;
    y=-h/2:h:1+h/2;
    phi_x_exact=zeros(M(n),M(n));
    phi_y_exact=zeros(M(n),M(n));
    for i =2:length(x)-1 % Only the interior cells needed
        for j =2:length(y)-1
            phi_x_exact(i-1,j-1)=-9*pi*x(i).^2.*cos(2*pi*y(j).^2).*sin(3*pi*x(i).^3);
            phi_y_exact(i-1,j-1)=-4*pi*y(j).*sin(2*pi*y(j).^2).*cos(3*pi*x(i).^3);
        end
    end
    phi=zeros(M(n)+2,M(n)+2); % +2 to include the ghost cells
    phi_x=zeros(M(n),M(n));
    phi_y=zeros(M(n),M(n));
    phi_SOR=zeros(M(n)+2,M(n)+2); % +2 to include the ghost cells
    f=zeros(M(n)+2,M(n)+2);
    for i=1:length(x)
        for j=1:length(y)
            f(i,j) = -9*pi*x(i).*cos(2*pi*y(j).^2).*(2*sin(3*pi*x(i).^3)+9*pi*x(i).^3.*cos(3*pi*x(i).^3))...
        -4*pi*cos(3*pi*x(i).^3).*(sin(2*pi*y(j).^2)+4*pi*y(j).^2.*cos(2*pi*y(j).^2));
        end
    end

    [phi,LinfR] = poisson(phi,f,h,nIterMax,eps);
    [phi_SOR,LinfR_SOR] = SOR(phi_SOR,f,h,nIterMax,eps);

    % Calculate derivative with respect to x and y
    for i=2:length(phi)-1
        for j=2:length(phi)-1
            phi_x(i-1,j-1)=(phi(i+1,j)-phi(i-1,j))/(2*h);
            phi_y(i-1,j-1)=(phi(i,j+1)-phi(i,j-1))/(2*h);
        end
    end

    e_x_inf=ErrorNorm2D(phi_x_exact,phi_x,inf);
    e_x_1=ErrorNorm2D(phi_x_exact,phi_x,1);
    e_x_2=ErrorNorm2D(phi_x_exact,phi_x,2);
    e_y_inf=ErrorNorm2D(phi_y_exact,phi_y,inf);
    e_y_1=ErrorNorm2D(phi_y_exact,phi_y,1);
    e_y_2=ErrorNorm2D(phi_y_exact,phi_y,2);

    T_x(n,1:4)=[M(n) e_x_inf e_x_1 e_x_2];
    T_y(n,1:4)=[M(n) e_y_inf e_y_1 e_y_2];

    figure(fig1)
    semilogy(iter,LinfR)
    hold on
    semilogy(iter,LinfR_SOR)
    grid on
    xlabel('$k$ (Iterations)','fontsize',labelfontsize,'interpreter','latex')
    ylabel('$L_{\infty}(r)$  (Log scale)','fontsize',labelfontsize,'interpreter','latex')
    title('Infinity norm of the residues')
end

lh = legend({'$\phi_{SOR}(M=256)$','$\phi_{V}(M=256)$','$\phi_{SOR}(M=128)$',...
    '$\phi_{V}(M=128)$','$\phi_{SOR}(M=64)$','$\phi_{V}(M=64)$',...
    '$\phi_{SOR}(M=32)$','$\phi_{V}(M=32)$'},...
    'Interpreter','latex','Fontsize',14);
ylim([1e-3 1e5])
legendEntries = get(gca,'legend');
plotHandles = get(gca,'children');
legendEntries = legendEntries.String;
newOrder = [1,3,5,7,8,6,4,2];
legend(plotHandles(newOrder),legendEntries{newOrder});
```



In the previous figure we see how the SOR method is much slower than the multigrid method, as we discussed in problem 2. We can see how the residuals of the multigrid method get stabilized after around 15 iterations. This is the convergence criterion that we have used, I have done more than just enough iterations for the sake of the discussion about the previous plot. We can see how the cost of the multigrid procedure is not dominant, we can observe how the speed of the multigrid method is a bit lower the higher number of elements by looking at the slope of the straight line. Once the residuals don't decrease anymore regardless of the number of iterations done, we can conclude that the error has then round-off and truncation error components domaining. We see how the horizontal line does decrease in value with the number of elements M. If the residuals decrease with M, so does the error. Therefore we can conclude that, once converged, the method is more precise with higher number of iterations because the truncation error is smaller. This discussion is about the method obtaining ϕ. Once ϕ is obtained, we use second order central finite difference to obtain the partial derivatives. In the following tables we can see how the norms of the error between the exact values of the partial derivatives and the ones obtained by the code decrease with the number of elements, as expected. The observed order of accuracy is calculated and it is 2 for all norms used. In this case, since we use a cell centered method, we don't use a first order method in the boundaries and that's why we don't see the observed order of accuracy equal to 1 when calculated with the infinity norm and between 1 and 2 when calculated with the 2-norm, as we saw in a previous homework assignment. In this case, every cell value is calculated using the second order method, as we can see in the tables.

```
% In the following lines the observed order of accuracy is calculated and
% the matrices T_x and T_y turned into tables to show them.
T_x(1,5:7)=NaN;
T_y(1,5:7)=NaN;
for i=2:length(M)
    T_x(i,5)=log(T_x(i,2)/T_x(i-1,2))/log(H(i)/H(i-1));
    T_x(i,6)=log(T_x(i,3)/T_x(i-1,3))/log(H(i)/H(i-1));
    T_x(i,7)=log(T_x(i,4)/T_x(i-1,4))/log(H(i)/H(i-1));

    T_y(i,5)=log(T_y(i,2)/T_y(i-1,2))/log(H(i)/H(i-1));
    T_y(i,6)=log(T_y(i,3)/T_y(i-1,3))/log(H(i)/H(i-1));
    T_y(i,7)=log(T_y(i,4)/T_y(i-1,4))/log(H(i)/H(i-1));
end
```

The following tables correspond to the calculation of the partial derivative of ϕ with respect to x

```
T_x=array2table(T_x,'VariableNames',{'M','L_inf','L_1','L_2','o_inf','o_1','o_2'});
T_x(:,1:4)
T_x(:,[1 5 6 7])
```

```
ans =

  4×4 table

    M          L_inf              L_1               L_2
    ___    _____________    _____________    _____________

     32     1.68979536277912     0.160661772464168     0.345021370632811
     64    0.429386818835912     0.0391644360751717    0.0847516470596121
    128    0.107380795342888     0.0097715148483807    0.0211009067889418
    256    0.026881158731495    0.00243966323615875    0.005269889329531
```

```
ans =

  4×4 table

    M          o_inf              o_1               o_2
    ___    _____________    _____________    _____________

     32               NaN              NaN              NaN
     64    1.97649873450377    2.03641060750598    2.02537241432421
    128    1.99954190880177    2.00289004400731    2.00593640724671
    256    1.99806876276752    2.00190021997041    2.00146042861105
```

The following tables correspond to the calculation of the partial derivative of ϕ with respect to y

```
T_y=array2table(T_y,'VariableNames',{'M','L_inf','L_1','L_2','o_inf','o_1','o_2'});
T_y(:,1:4)
T_y(:,[1 5 6 7])
```

```
ans =

  4×4 table
```

M	L_inf	L_1	L_2
32	0.336382561851182	0.0490686283619557	0.075637449358093
64	0.0817538078469964	0.0119237609025396	0.0184463564648617
128	0.0203138340334066	0.00296138706488763	0.00458346736139981
256	0.00506936686773951	0.000739234659726147	0.00114412269657658

ans =

4×4 table

M	o_inf	o_1	o_2
32	NaN	NaN	NaN
64	2.04074508419794	2.04096159210481	2.03576483117012
128	2.00882336895853	2.00949437769931	2.00882457716439
256	2.00258507962996	2.00216876313852	2.00219762406633

Published with MATLAB® R2017a