|  |
| --- |
|  |
| Cavity Flow Simulation via SIMPLE |
| ME 811 HW#4 |
|  |
| **Michael Crawley** |
| **5/6/2011** |

|  |
| --- |
|  |

From the following plots of the velocity and pressure contours, the recirculation pattern that develops in the cavity can be clearly observed. It is seen that the location of the center of the vortex changes with Reynolds number, with low Reynolds numbers resulting in a recirculation pattern offset towards the top right side of the cavity and high Reynolds numbers resulting in a circulation pattern that is centered in the cavity. The location of the vortex center corresponds to a low pressure region, as one would expect. Flow velocities in the lower corners of the cavity are found to be quite small. For the Reynolds number simulated in this study, secondary recirculation patterns do not develop in the lower corners. A high pressure node develops in the right top corner, with a corresponding low pressure node in the top left corner. The magnitude of these pressure nodes increases with increasing Reynolds number. Higher Reynolds numbers are shown to result in higher velocity gradients near the cavity lid and walls.

Increasing the grid resolution results in slightly higher magnitudes for the pressure nodes in the top left and right corners, though the velocity vectors and magnitudes are unchanged. These results are similar for all Reynolds numbers simulated in this study. The number of outer iterations required for convergence is found to increase slightly with increasing Reynolds number. However, the number of outer iterations required increases significantly with increasing mesh resolution. As each inner iteration requires more computational time to complete for the finer mesh as well, the required time to reach convergence increases dramatically by doubling the grid resolution. Oscillations in the residuals are clearly evident during the initial iterations during the simulations, particularly for the pressure. The amplitude of the oscillations eventually decays, so that a linear trend is observed in the residuals as they reach convergence. The frequency of the oscillations (with respect to the number of iterations) appears to increase with increasing grid resolution, though it is not clear why this occurred. The velocity residuals are found to be consistently lower than the pressure residuals, by at least an order of magnitude.

|  |  |
| --- | --- |
| Z:\My Documents\Matlab\Coursework\ME 811\HW 4\40x40 mesh, Re = 100 x centerline.png | Z:\My Documents\Matlab\Coursework\ME 811\HW 4\40x40 mesh, Re = 100 y centerline.png |
|  |  |
| Z:\My Documents\Matlab\Coursework\ME 811\HW 4\40x40 mesh, Re = 100 pressure.png | |

Figure : Centerline velocities, velocity and pressure residuals, and pressure contours and velocity vectors for 40 by 40 mesh for Reynolds number flow of 100.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

Figure 2: Centerline velocities, velocity and pressure residuals, and pressure contours and velocity vectors for 40 by 40 mesh for Reynolds number flow of 500.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

Figure 3: Centerline velocities, velocity and pressure residuals, and pressure contours and velocity vectors for 40 by 40 mesh for Reynolds number flow of 1000.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

Figure 4: Centerline velocities, velocity and pressure residuals, and pressure contours and velocity vectors for 80 by 80 mesh for Reynolds number flow of 100.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

Figure 5: Centerline velocities, velocity and pressure residuals, and pressure contours and velocity vectors for 80 by 80 mesh for Reynolds number flow of 500.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

Figure 6: Centerline velocities, velocity and pressure residuals, and pressure contours and velocity vectors for 80 by 80 mesh for Reynolds number flow of 1000.

# Appendix

function [u v p Resx Resy] = SIMPLE(rho,gamma,alpha,omega,Ulid,M,N)

tic

Rtol = 10E-10;

itrmax = 1E4;

L = 0.01;

dx = L/N;

dy = L/M;

x = 0:dx:L;

y = 0:dy:L;

xc = dx/2:dx:L-dx/2;

yc = dy/2:dy:L-dy/2;

%initialize variables

u = zeros(M,N);

v = u; p = u;

Resx = zeros(1,itrmax+1);Resx(1) = 1;

counter = 1;

Resy = Resx; Resp = Resx;

while (Resx(counter) >= Rtol || Resy(counter) >= Rtol) && counter <= itrmax

[AxO AxE AxW AxN AxS Sx] = calcXcoefs(u,v,p,Ulid,rho,gamma,dx,dy);

[AyO AyE AyW AyN AyS Sy] = calcYcoefs(u,v,p,rho,gamma,dx,dy);

[uh Resx(counter+1)] = Xsolver(alpha,AxO,AxE,AxW,AxN,AxS,Sx,u);

uh = u+uh;

[vh Resy(counter+1)] = Ysolver(alpha,AyO,AyE,AyW,AyN,AyS,Sy,v);

vh = v+vh;

[ApO ApE ApW ApN ApS Sp] = calcPcoefs(rho,dx,dy,uh,vh,AxO,AyO);

[pp Resp(counter+1)] = Psolver(ApO,ApE,ApW,ApN,ApS,Sp);

u(:,2:end) = uh(:,2:end)+omega.u\*dy\*(pp(:,1:end-1)-pp(:,2:end))./AxO;

v(1:end-1,:) = vh(1:end-1,:)+omega.v\*dx\*(pp(2:end,:)-pp(1:end-1,:))./AyO;

p = p+omega.p\*pp;

fprintf('counter: %i Resx: %1.2e Resy: %1.2e Resp: %1.2e \n',[counter Resx(counter+1) Resy(counter+1) Resp(counter+1)]);

counter = counter+1;

end

u = [u zeros(M,1)];

u = flipud(u);

v = [zeros(1,N); v];

v = flipud(v);

p = flipud(p);

Resx = Resx(2:counter);

Resy = Resy(2:counter);

Resp = Resp(2:counter);

h(1) = figure;contourf(x,yc,u);title('u velocity');colormap gray;colorbar;

h(2) = figure;contourf(xc,y,v);title('v velocity');colormap gray;colorbar;

h(3) = figure;contourf(xc,yc,p);title('pressure');colormap gray;colorbar;

foldername = [num2str(M),'x',num2str(N),' Re',num2str(round(rho\*Ulid\*L/gamma))];

if exist(foldername,'file') ~= 7

mkdir(pwd,foldername);

end

cd(foldername);

save data.mat;

saveas(h(1),'u velocity','fig'); saveas(h(1),'u velocity','png');

saveas(h(2),'v velocity','fig'); saveas(h(2),'v velocity','png');

saveas(h(3),'p velocity','fig'); saveas(h(3),'p velocity','png');

close(h);

cd ..

toc

end

function [AxO AxE AxW AxN AxS Sx] = calcXcoefs(u,v,p,Ulid,rho,gamma,dx,dy)

%Calculates link coefficients for x-momentum FVM equation

%Note: for an NxMy mesh, u(:,1) is zero while v(:,1) and p(:,1) are not;

%the u and v velocity fields are modified to reflect this.

%Inputs:

% u: x-velocity component (NxM matrix)

% v: y-velocity component (NxM matrix)

% p: pressure (NxM matrix)

% Ulid: lid velocity

% rho: density (constant)

% gamma: viscosity (constant)

% dx: step size in x-direction

% dy: step size in y-direction

[M N] = size(u);

ut = [u zeros(M,1)]; %add known boundary values to simplify computations

vt = [zeros(1,N); v]; %add known boundary values to simplify computations

ce = rho\*0.5\*(ut(:,2:end-1)+ut(:,3:end));

cw = rho\*0.5\*(ut(:,2:end-1)+ut(:,1:end-2));

cn = rho\*0.5\*(vt(1:end-1,2:end)+vt(1:end-1,1:end-1));

cs = rho\*0.5\*(vt(2:end,2:end)+vt(2:end,1:end-1));

dep = 0.5\*(abs(ce)+ce);

dem = 0.5\*(abs(ce)-ce);

dwp = 0.5\*(abs(cw)+cw);

dwm = 0.5\*(abs(cw)-cw);

dsp = 0.5\*(abs(cs)+cs);

dsm = 0.5\*(abs(cs)-cs);

dnp = 0.5\*(abs(cn)+cn);

dnm = 0.5\*(abs(cn)-cn);

%Calculate link coefficients for interior nodes

AxO = (dep+dwm+2\*gamma/dx)\*dy+(dnp+dsm+2\*gamma/dy)\*dx;

AxE = -(dem+gamma/dx)\*dy;

AxW = -(dwp+gamma/dx)\*dy;

AxN = -(dnm+gamma/dy)\*dx;

AxS = -(dsp+gamma/dy)\*dx;

Sx = - (p(:,2:end)-p(:,1:end-1))\*dy;

%Modify link coefficients for North Boundary

AxO(1,:) = (dep(1,:)+dwm(1,:)+2\*gamma/dx)\*dy+(dnp(1,:)+4\*gamma/dy)\*dx;

AxN(1,:) = 0;

AxS(1,:) = -(dsp(1,:)+4\*gamma/dy/3)\*dx;

Sx(1,:) = Sx(1,:)+8\*gamma\*dx\*Ulid/3/dy;

%Modify link coefficients for South Boundary

AxO(M,:) = (dep(M,:)+dwm(M,:)+2\*gamma/dx)\*dy+(dnp(M,:)+4\*gamma/dy)\*dx;

AxS(M,:) = 0;

AxN(M,:) = -(dnm(M,:)+4\*gamma/dy/3)\*dx;

end

function [AyO AyE AyW AyN AyS Sy] = calcYcoefs(u,v,p,rho,gamma,dx,dy)

%Calculates link coefficients for y-momentum FVM equation

%Inputs:

% u: x-velocity component (NxM matrix)

% v: y-velocity component (NxM matrix)

% p: pressure (NxM matrix)

% Ulid: lid velocity

% rho: density (constant)

% gamma: viscosity (constant)

% dx: step size in x-direction

% dy: step size in y-direction

[M N] = size(u);

ut = [u zeros(M,1)]; %add known boundary values to simplify computations

vt = [zeros(1,N); v]; %add known boundary values to simplify computations

ce = rho\*0.5\*(ut(1:end-1,2:end)+ut(2:end,2:end));

cw = rho\*0.5\*(ut(1:end-1,1:end-1)+ut(2:end,1:end-1));

cn = rho\*0.5\*(vt(2:end-1,:)+vt(1:end-2,:));

cs = rho\*0.5\*( vt(2:end-1,:)+vt(3:end,:));

dep = 0.5\*(abs(ce)+ce);

dem = 0.5\*(abs(ce)-ce);

dwp = 0.5\*(abs(cw)+cw);

dwm = 0.5\*(abs(cw)-cw);

dsp = 0.5\*(abs(cs)+cs);

dsm = 0.5\*(abs(cs)-cs);

dnp = 0.5\*(abs(cn)+cn);

dnm = 0.5\*(abs(cn)-cn);

%Calculate link coefficients for interior nodes

AyO = (dep+dwm+2\*gamma/dx)\*dy+(dnp+dsm+2\*gamma/dy)\*dx;

AyE = -(dem+gamma/dx)\*dy;

AyW = -(dwp+gamma/dx)\*dy;

AyN = -(dnm+gamma/dy)\*dx;

AyS = -(dsp+gamma/dy)\*dx;

Sy = -(p(1:end-1,:)-p(2:end,:))\*dx;

%Modify link coefficients for North Boundary

% AyN(1,:) = 0;

%Modify link coefficients for South Boundary - unnecessary

% AyS(M-1,:) = 0;

%Modify link coefficients for East Boundary

AyO(:,N) = (dwm(:,N)+4\*gamma/dx)\*dy+(dnp(:,N)+dsm(:,N)+2\*gamma/dy)\*dx;

AyE(:,N) = 0;

AyW(:,N) = -(dwp(:,N)+4\*gamma/dx/3)\*dy;

%Modify link coefficients for West Boundary

AyO(:,1) = (dep(:,1)+4\*gamma/dx)\*dy+(dnp(:,1)+dsm(:,1)+2\*gamma/dy)\*dx;

AyE(:,1) = -(dem(:,1)+4\*gamma/dx/3)\*dy;

AyW(:,1) = 0;

end

function [ApO ApE ApW ApN ApS Sp] = calcPcoefs(rho,dx,dy,uh,vh,AxO,AyO)

[M,~] = size(AxO);

[~,N] = size(AyO);

AxO = [Inf\*ones(M,1) AxO Inf\*ones(M,1)]; %AxO = (:,1:end-1), AxE = (:,2:end)

AyO = [Inf\*ones(1,N); AyO; Inf\*ones(1,N)]; %AyO = (2:end,:), AyN = (1:end-1,:)

uh = [uh zeros(M,1)];

vh = [zeros(1,N); vh];

%Calculate link coefficients for all nodes

ApO = -rho\*(dy^2./AxO(:,2:end)+dy^2./AxO(:,1:end-1)+dx^2./AyO(1:end-1,:)+dx^2./AyO(2:end,:));

ApE = rho\*dy^2./AxO(:,2:end);

ApW = rho\*dy^2./AxO(:,1:end-1);

ApN = rho\*dx^2./AyO(1:end-1,:);

ApS = rho\*dx^2./AyO(2:end,:);

Sp = rho\*((uh(:,2:end)-uh(:,1:end-1))\*dy+(vh(1:end-1,:)-vh(2:end,:))\*dx);

end

function [uh Resx] = Xsolver(alpha,AxO,AxE,AxW,AxN,AxS,Sx,u)

Rtol = 1E-20;

itrmax = 2;

[M N] = size(Sx);

%Reshape link coefficients

AxO = reshape(AxO,N\*M,1);

AxE = reshape(AxE,N\*M,1);

AxW = reshape(AxW,N\*M,1);

AxN = reshape(AxN,N\*M,1);

AxS = reshape(AxS,N\*M,1);

AxN = circshift(AxN,-1);

AxS = circshift(AxS,1);%need to check to make sure AN,AS are being shifted properly

AxE = circshift(AxE,M);

AxW = circshift(AxW,-M);

Ax = spdiags([AxW AxN AxO AxS AxE],[-M -1 0 1 M],N\*M,N\*M);

%Modify source for correction equation

Rx = reshape(Sx,M\*N,1)-Ax\*reshape(u(:,2:end),M\*N,1);

Sx = reshape(Rx,M,N);

Resx = norm(Rx);

Ax = Ax+diag(AxO\*alpha);

%Solve for u hat

[uh R] = ADIp(Ax,Sx,'-TDMA',Rtol,itrmax);

%append boundary condition

uh = [zeros(M,1) uh];

end

function [vh Resy] = Ysolver(alpha,AyO,AyE,AyW,AyN,AyS,Sy,v)

Rtol = 1E-20;

itrmax = 2;

[M N] = size(Sy);

%Reshape link coefficients

AyO = reshape(AyO,N\*M,1);

AyE = reshape(AyE,N\*M,1);

AyW = reshape(AyW,N\*M,1);

AyN = reshape(AyN,N\*M,1);

AyS = reshape(AyS,N\*M,1);

AyN = circshift(AyN,-1);

AyS = circshift(AyS,1);%need to check to make sure AN,AS are being shifted properly

AyE = circshift(AyE,M);

AyW = circshift(AyW,-M);

Ay = spdiags([AyW AyN AyO AyS AyE],[-M -1 0 1 M],N\*M,N\*M);

%Modify source for correction equation

Ry = reshape(Sy,M\*N,1)-Ay\*reshape(v(1:end-1,:),M\*N,1);

Sy = reshape(Ry,M,N);

Resy = norm(Ry);

Ay = Ay+diag(AyO\*alpha);

%Solve for v hat

[vh R] = ADIp(Ay,Sy,'-TDMA',Rtol,itrmax);

%append boundary condition

vh = [vh; zeros(1,N)];

end

function [pp Rp] = Psolver(ApO,ApE,ApW,ApN,ApS,Sp)

Rtol = 1E-20;

itrmax = 20;

[M N] = size(Sp);

%Reshape link coefficients

ApO = reshape(ApO,N\*M,1);

ApE = reshape(ApE,N\*M,1);

ApW = reshape(ApW,N\*M,1);

ApN = reshape(ApN,N\*M,1);

ApS = reshape(ApS,N\*M,1);

ApN = circshift(ApN,-1);

ApS = circshift(ApS,1);%need to check to make sure AN,AS are being shifted properly

ApE = circshift(ApE,M);

ApW = circshift(ApW,-M);

Ap = spdiags([ApW ApN ApO ApS ApE],[-M -1 0 1 M],N\*M,N\*M);

%Solve for u hat

[pp R] = ADIp(Ap,Sp,'-TDMA',Rtol,itrmax);

Rp = norm(R);

end

function [phi,R] = ADIp(X,S,method,Rtol,itrmax)

%Performs the Alternating Direction Implicit solver for a 2 dimensional

%system.

%Note: conversion from 2-D grid to 1-D array must be done as i,j -> k = M(i-1)+j,

%where there are M nodes in the y direction and N nodes in the x

%direction

%Code Version: 1.0 @ 2011-04-10

%Inputs:

% X: matrix for the x-derivative terms (M\*N,M\*N)

% S: source term, either matrix or scalar (M,N)

% method: '-TDMA' for tridiag systems, '-GaussSeidel' otherwise

% Rtol: Residual tolerance (optional; default 1E-5)

% itrmax: maximum number of iterations (optional; default 1E4)

%Outpus:

% phi: Solution to the problem

% R: L2norm of the Residuals at each iteration

if ~exist('Rtol','var')

Rtol = 1E-5;

end

if ~exist('itrmax','var')

itrmax = 1E4;

end

[M N] = size(S);

phi = zeros(M\*N,1);

S = reshape(S,M\*N,1);

R = zeros(1,itrmax);

R(1) = norm(X\*phi-S);

counter = 0;

AW = [zeros(M,1); diag(X,-M)];

AE = [diag(X,M); zeros(M,1)];

AN = [diag(X,1); 0];

AS = [0; diag(X,-1)];

AO = diag(X);

while (R(counter+1) >= Rtol) && (counter < itrmax)

%Row sweep

k = 1:M:1+M\*(N-1);

b = S(k)-AN(k).\*phi(k+1);

% A = spdiags([AW(k) AO(k) AE(k)],-1:1,length(k),length(k));

% phi(k) = TDMsolver(A,b);

phi(k) = TDMAsolver(AW(k),AO(k),AE(k),b);

for j = 2:M-1

k = j:M:j+M\*(N-1); %determine nodal array points

b = S(k)-AN(k).\*phi(k+1)-AS(k).\*phi(k-1);

% A = spdiags([AW(k) AO(k) AE(k)],-1:1,length(k),length(k));

% phi(k) = TDMsolver(A,b);

phi(k) = TDMAsolver(AW(k),AO(k),AE(k),b);

end

k = M:M:M+M\*(N-1); %determine nodal array points

b = S(k)-AS(k).\*phi(k-1);

% A = spdiags([AW(k) AO(k) AE(k)],-1:1,length(k),length(k));

% phi(k) = TDMsolver(A,b);

phi(k) = TDMAsolver(AW(k),AO(k),AE(k),b);

%Column sweep

k = M\*(1-1)+1:M\*1;

b = S(k)-AE(k).\*phi(k+M);

% A = spdiags([AS(k) AO(k) AN(k)],-1:1,length(k),length(k));

% phi(k) = TDMsolver(A,b);

phi(k) = TDMAsolver(AS(k),AO(k),AN(k),b);

for i = 2:N-1

k = M\*(i-1)+1:M\*i;

b = S(k)-AE(k).\*phi(k+M)-AW(k).\*phi(k-M);

% A = spdiags([AS(k) AO(k) AN(k)],-1:1,length(k),length(k));

% phi(k) = TDMsolver(A,b);

phi(k) = TDMAsolver(AS(k),AO(k),AN(k),b);

end

k = M\*(N-1)+1:M\*N;

b = S(k)-AW(k).\*phi(k-M);

% A = spdiags([AS(k) AO(k) AN(k)],-1:1,length(k),length(k));

% phi(k) = TDMsolver(A,b);

phi(k) = TDMAsolver(AS(k),AO(k),AN(k),b);

counter = counter + 1;

R(counter+1) = norm(X\*phi-S);

end

R = R(1:counter+1);

phi = reshape(phi,M,N);

end

function [x] = TDMAsolver(aL,a,aR,b)

% Solves tridiagonal matrix using Thomas' algorithm.

%Inputs:

% A: Full matrix, where Ax=b (modified)

% b: Column vector

%Outputs:

% x: Solution to the linear equation

n = length(b);

x = zeros(n,1);

aR(1) = aR(1)/a(1);

b(1) = b(1)/a(1);

for i=2:n

t = 1/(a(i)-aR(i-1)\*aL(i));

aR(i) = aR(i)\*t;

b(i) = (b(i)-b(i-1)\*aL(i))\*t;

end

x(n) = b(n);

for i = n-1:-1:1

x(i) = b(i)-aR(i)\*x(i+1);

end

end