Advanced Intro to CFD final project report

*Alex Perron*

*12/12/22*

Abstract

The purpose of this project is to edit and debug a coded template to create a functioning 2D finite difference CFD code. The code uses both point Jacobi and symmetric Gauss-Seidel schemes to solve the 2D incompressible Navier-Stokes equations. To do this, these schemes make use of both time derivative preconditioning and artificial viscosity. The cases solved in this project are a manufactured solution and a lid driven cavity. Varying sizes of grids, CFL numbers and Re numbers are used.

Theory

1. Governing equations and Discretization

Text, letter

Description automatically generatedThe base governing equations used in this code are the 2D incompressible N-S equations. These equations can be seen below:

These equations already include both the time preconditioning element as well as the artificial viscosity. The source terms are also included for the manufactured solution case.

The above equations are discretized for a simple explicit method using central in space, forward in time. Below is the discretization for the continuity and x momentum equations.

A picture containing diagram

Description automatically generated

*Continuity discretization*

Graphical user interface, application, Word

Description automatically generated

*X-momentum discretization*

1. Boundary Conditions

The boundary conditions used for the Lid driven cavity were wall boundary conditions. For the bottom, left and right walls, both components of velocity (u,v) are set to zero and the pressure is linearly extrapolated from the inner 2 nodes. For example, the pressure on the bottom wall can be found using the following equation:

For the top wall, the pressure and y component of velocity (v) are treated the same as the other 3 walls, however now the x component of velocity is 1m/s. ie:

utop wall = uinf

1. Artificial viscosity

The artificial viscosity shows up in the Continuity equation as “S”. The expression for S is derived as:

A picture containing diagram

Description automatically generated

The C(4) constant in this case is a chosen constant that can very from 1/128 to 1/16. β2 is the time derivate preconditioning term and is found via the following equation:



Lastly, the 4th derivative of pressure is found using a simple finite difference discretization throughout the domain. For the interior nodes the discretization is a 2nd order 4th derivative central difference scheme as follows:

For the nodes one node away from a wall or corner, where i or j = 2 or Nodemax -1, the 4th derivative was found using a forwards or backwards difference (2nd order accurate) scheme. For example, on the left wall, the 4th derivative of pressure w.r.t x is found via the following equation:

The following equation keeps the same signs regardless if it is forwards or backwards difference. Whether using forwards or backwards difference depends on which side the pressure derivate was calculated on.

1. Time step

The time step for each iteration is determined by the following equation:

Text

Description automatically generated

Where |λmax| is the max between the following:

A picture containing text, antenna

Description automatically generated

Δtc is found at each node, and the minimum value of Δtc is used to determine the next time step. This process of using the minimum value of tc ­across all nodes is called global time stepping.

Results

1. Comparison of Numerical Schemes used

The two numerical schemes used in this project are the point Jacobi and symmetric Gauss-Seidel. Both methods are explicit and used the discretization described in the prior section. The difference between these methods lie in the Gauss-Seidel having a forward and backward sweep where it updates its values twice in one loop, where as the Point Jacobi only uses one sweep, and updates its values from the prior iteration.

Below is a plot comparing the iterative convergence of the two methods as well as the solutions for the MMS with both schemes



1. Discretization error and order of accuracy

The discretization error norms are calculated with the following equations.

Where ε is the discretization error at that node. The observed order of accuracy is calculated using the following equation:

Here is a DE norm, and r is the grid refinement factor, in this case r = 2. With this, the discretization error norms and Observed order of accuracy can be plotted.

Unfortunately, these results do not agree with what a good solution or mesh refinement would show. In fact these results don’t make much sense at all. This could be due to the sizes of the meshes used (33x33 up to 129x129), or bugs in the code, or issues with compiling results. These all being considered, there was no more time to chase down bugs or figure out the underlying reason for these results.

1. The effect of the C(4) constant on discretization error

The C(4) constant can be seen used in the equation for artificial viscosity (reference that section for said equation). Changing the C(4) constant will effect your artificial viscosity and in tern your discretization error. Below is a plot of discretization error at y close to the top of the chamber and x from 0 – 0.01.

As can be seen from the above plot, as you decrease the C(4) constant from 0.02 to 0.001 the overall discretization error decreases. There is large fluctuation in the DE at the higher values and odd-even decoupling can be seen. The sweet spot for the C(4) constant appears to be around 0.001. Once the C(4) constant is decreased too far, the odd even decoupling comes back.

1. The effect of the rkappa (κ) constant

The rkappa constant is a constant that shows up in the β2 time derivative preconditioning term. This constant can be seen in the equation given in the artificial viscosity section. Like the C(4) study, this too will be performed on the MMS solution with a Reynolds number of 100 on a 65x65 grid. rkappa can range from 0.001 to 0.9.

Below are two plots of the pressure, the left figure having an rkappa of 0.9 and the right figure having a rkappa of 0.001



The only difference that is apparent between these two cases, is that the run with the higher rkappa of 0.9 (left), was completed in slightly less iterations than that of the higher rkappa of 0.001 (right).

1. Comparison of the Symmetric Gauss Seidel to the manufactured solution case

Below are the results for a 65x65 SGS run for manufactured solution

**

**

As can be seen from above, the manufactured solutions match very closely with the manufactured solutions given in the project statement. More importantly the iterative residuals converge. Our discretization error norms are low as well (most falling in the magnitude of 10-4).

1. Lid driven cavity results

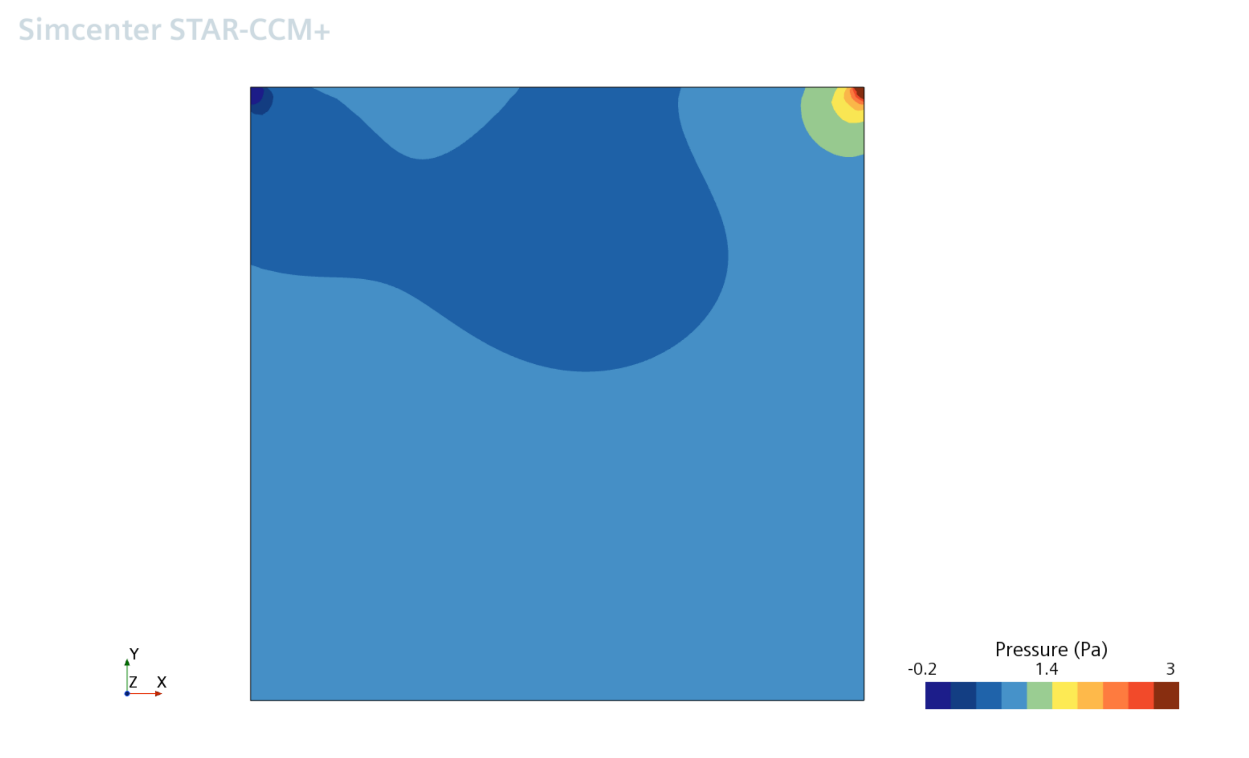
Below are the results of the LDC test case

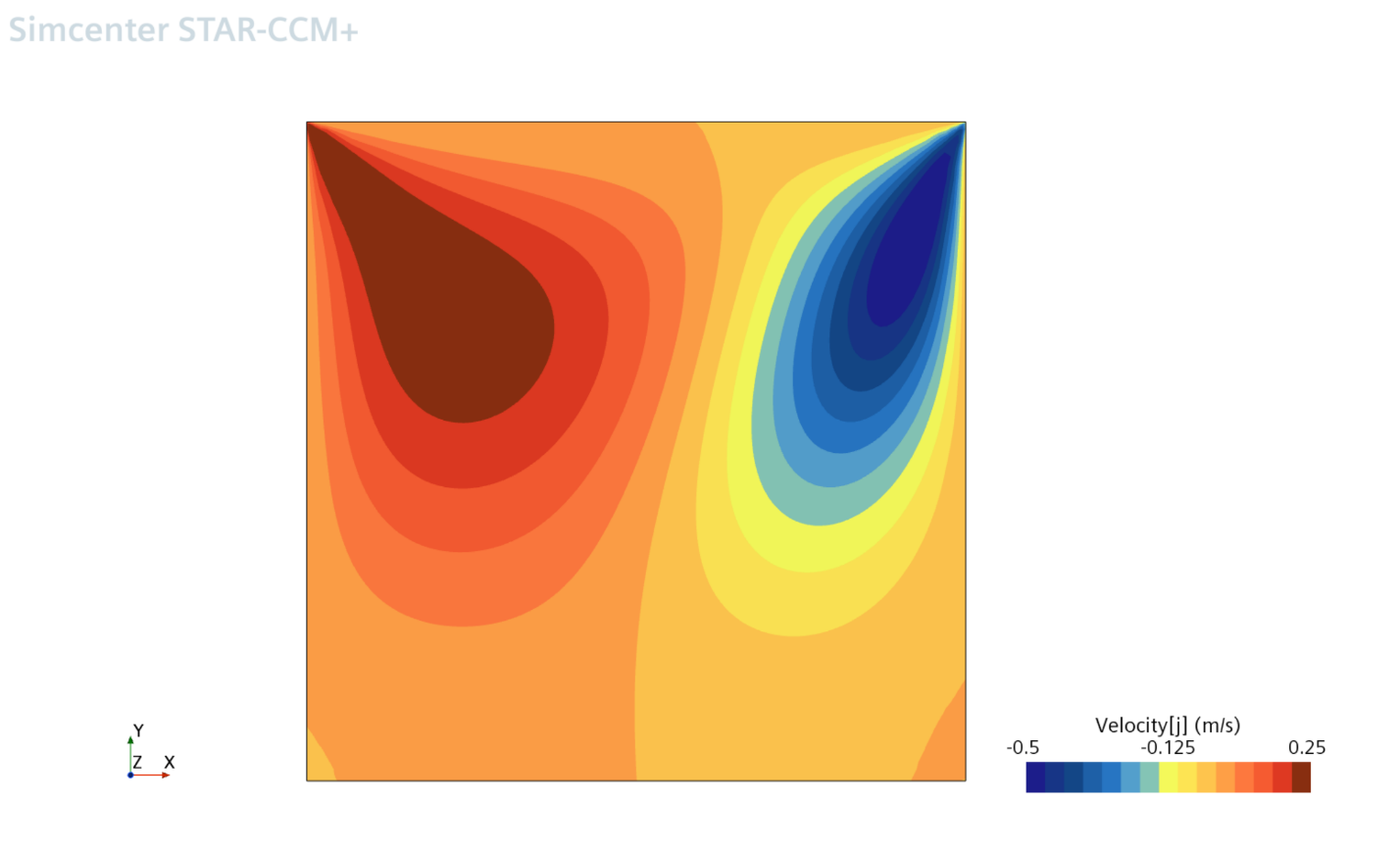
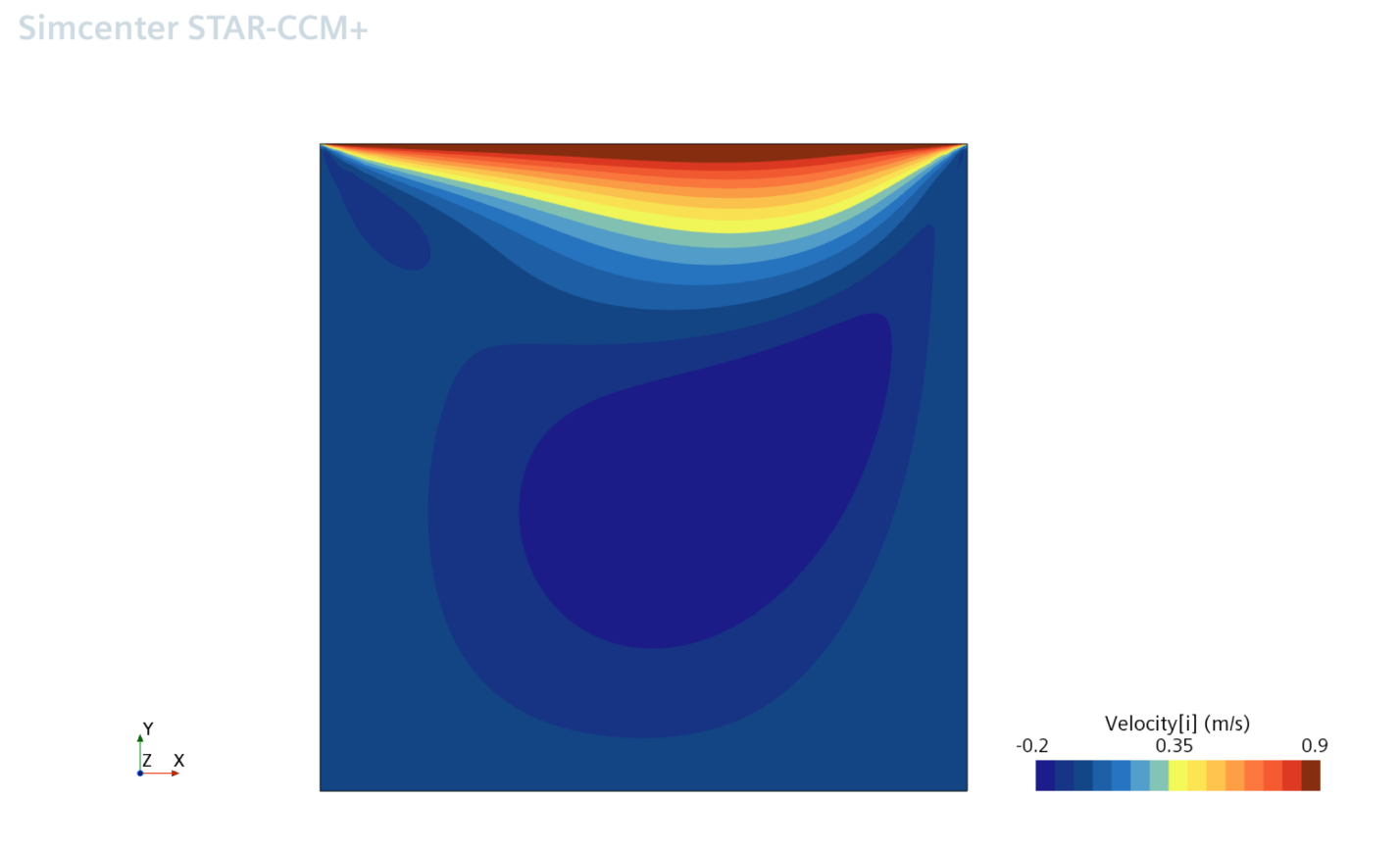








As can be seen from the results above, these compare closely to the results given in the project statement. These also match closely with the below results from StarCCM + run with the same initial and boundary conditions.



1. Higher Reynolds number cases

For these cases the residuals plot and the u velocity plot with stream lines are the only thing shown. However all of the plots can be generated from the raw .mat files in the git-repository. (link provided in apendix). Below are the results from the Higher Reynolds number cases

Re 500:





Re 1000:



Chart, diagram

Description automatically generated

In these higher Reynolds number cases you can start to see the flow separation off of the corners of the cavity. There is now obvious circulation in the bottom right corner, and at Re=1000, the flow starts to circulate in the bottom left corner. In addition, in the highest Reynolds number case, it would appear that there are the beginnings of flow separation and circulation in the top left corner (indicated by the streamlines and velocity contour intrusion).

For the Re of 1000 case I had to take the results from an un-converged solution due to time constraints. Unfortunately the residuals for this case were only brought down 2 orders of magnitude even after 400 thousand iterations (and about 1 day of computational time). In addition, this case was also run before the bug with the artificial viscosity was caught, the results are a bit suspect even just beyond iterative convergence. It should also be mentioned that with this case, due to the high Reynolds number, the cfl number had to be dropped to 0.2 and the grid had to be increased to a 257x257 mesh. This dramatically increased computational time.

1. Remarks

Unfortunately, due to time limitations and computational resources, both the simulation runs, the code development, and the project write up were cut short and not all of the required runs and results could be show, talked about, or finished. A bug in the artificial viscosity was found only days before the due date. Fixing this bug dramatically changed results I was getting, Discretization errors and plots, and would make any results before this fix invalid. Even with that bug being found, the results that were found even after this bug fix had issues to do with another, unfound bug. Unfortunately, time ran out before this and other bugs could be fixed.

Nonetheless I have presented what I have done thus far, the code will be both pasted at the end of this document and attached as a .m file.

1. Github repository

In addition to the raw code, all files, scripts, and data can be found on the Git-hub repository I created for version control and remote storage and access.

The github drectory is available at: <https://github.com/Mufasa6896/CFD_LDC_Project>

This directory will be set as public and will be archived once the project has been turned in.

1. Appendix A - Matlab code (main)

% function [PrsMatrix, uvelMatrix, vvelMatrix] = cavity\_solver(~)

tic %begin timer function

%--- Variables for file handling ---

%--- All files are globally accessible ---

global fp1 % For output of iterative residual history

global fp2 % For output of field data (solution)

% global fp3 % For writing the restart file

% global fp4 % For reading the restart file

% global fp5 % For output of final DE norms (only for MMS)

%$$$$$$ global fp6 % For debug: Uncomment for debugging.

global imax jmax neq nmax

global zero tenth sixth fifth fourth third half one two three four six

global iterout imp\_mat imms isgs irstr ipgorder lim cfl Cx Cy toler rkappa Re pinf uinf rho rhoinv xmin xmax ymin ymax Cx2 Cy2 fsmall

global rlength rmu vel2ref dx dy rpi phi0 phix phiy phixy apx apy apxy fsinx fsiny fsinxy

%\*\*Use these variables cautiously as these are globally accessible from all functions.\*\*

global u; % Solution vector [p, u, v]^T at each node

global uold; % Previous (old) solution vector

global s; % Source term

global dt; % Local time step at each node

global artviscx; % Artificial viscosity in x-direction

global artviscy; % Artificial viscosity in y-direction

global ummsArray; % Array of umms values (funtion umms evaluated at all nodes)

%\*\*\*\*\*\*\*\*\*\*\*\* Following are fixed parameters for array sizes \*\*\*\*\*\*\*\*\*\*\*\*\*

% Number of points in the x-direction (use odd numbers only)

% Number of points in the y-direction (use odd numbers only)

% imax = 257;

% jmax = 257;

% imax = 129;

% jmax = 129;

imax = 65;

jmax = 65;

% imax = 33;

% jmax = 33;

neq = 3; % Number of equation to be solved ( = 3: mass, x-mtm, y-mtm)

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\* All variables declared here. \*\*

%\*\*\*\* These variables SHOULD not be changed \*

%\*\*\*\*\*\*\*\*\* by the program once set. \*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\* The variables declared "" CAN \*\*\*\*

%\*\* not be changed by the program once set \*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%--------- Numerical constants --------

zero = 0.0;

tenth = 0.1;

sixth = 1.0/6.0;

fifth = 0.2;

fourth = 0.25;

third = 1.0/3.0;

half = 0.5;

one = 1.0;

two = 2.0;

three = 3.0;

four = 4.0;

six = 6.0;

%--------- User sets inputs here --------

nmax = 500000; % Maximum number of iterations

% nmax = 1500000;

iterout = 5000; % Number of time steps between solution output

imms = 1; % Manufactured solution flag: = 1 for manuf. sol., = 0 otherwise

isgs = 1; % Symmetric Gauss-Seidel flag: = 1 for SGS, = 0 for point Jacobi

irstr = 0; % Restart flag: = 1 for restart (file 'restart.in', = 0 for initial run

ipgorder = 0; % Order of pressure gradient: 0 = 2nd, 1 = 3rd (not needed)

lim = 1; % variable to be used as the limiter sensor (= 1 for pressure)

cfl = 0.9; % CFL number used to determine time step

Cx = 0.001; % Parameter for 4th order artificial viscosity in x

Cy = 0.001; % Parameter for 4th order artificial viscosity in y

% Cx = 0.001;

% Cy = 0.001;

toler = 1.e-10; % Tolerance for iterative residual convergence

rkappa = 0.1; % Time derivative preconditioning constant

Re = 100.0; % Reynolds number = rho\*Uinf\*L/rmu

pinf = 0.801333844662; % Initial pressure (N/m^2) -> from MMS value at cavity center

uinf = 1.0; % Lid velocity (m/s)

rho = 1.0; % Density (kg/m^3)

xmin = 0.0; % Cavity dimensions...: minimum x location (m)

xmax = 0.05; % maximum x location (m)

ymin = 0.0; % maximum y location (m)

ymax = 0.05; % maximum y location (m)

Cx2 = 0.0; % Coefficient for 2nd order damping (not required)

Cy2 = 0.0; % Coefficient for 2nd order damping (not required)

fsmall = 1.e-20; % small parameter

%-- Derived input quantities (set by function 'set\_derived\_inputs' called from main)----

rhoinv = -99.9; % Inverse density, 1/rho (m^3/kg)

rlength = -99.9; % Characteristic length (m) [cavity width]

rmu = -99.9; % Viscosity (N\*s/m^2)

vel2ref = -99.9; % Reference velocity squared (m^2/s^2)

dx = -99.9; % Delta x (m)

dy = -99.9; % Delta y (m)

rpi = -99.9; % Pi = 3.14159... (defined below)

%-- constants for manufactured solutions ----

phi0 = [0.25, 0.3, 0.2]; % MMS constant

phix = [0.5, 0.15, 1.0/6.0]; % MMS amplitude constant

phiy = [0.4, 0.2, 0.25]; % MMS amplitude constant

phixy = [1.0/3.0, 0.25, 0.1]; % MMS amplitude constant

apx = [0.5, 1.0/3.0, 7.0/17.0]; % MMS frequency constant

apy = [0.2, 0.25, 1.0/6.0]; % MMS frequency constant

apxy = [2.0/7.0, 0.4, 1.0/3.0]; % MMS frequency constant

fsinx = [0.0, 1.0, 0.0]; % MMS constant to determine sine vs. cosine

fsiny = [1.0, 0.0, 0.0]; % MMS constant to determine sine vs. cosine

fsinxy = [1.0, 1.0, 0.0]; % MMS constant to determine sine vs. cosine

% Note: fsin = 1 means the sine function

% Note: fsin = 0 means the cosine function

% Note: arrays here refer to the 3 variables

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Main Function

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%----- Looping indices --------

i = 0; % i index (x direction)

j = 0; % j index (y direction)

k = 0; % k index (# of equations)

n = 0; % Iteration number index

conv = -99.9 ; % Minimum of iterative residual norms from three equations

%--------- Solution variables declaration --------

ninit = 0; % Initial iteration number (used for restart file)

%$$$$$$ u(imax,jmax,neq); % Solution vector (p, u, v)^T at each node

%$$$$$$ uold(imax,jmax,neq); % Previous (old) solution vector

%$$$$$$ s(imax,jmax,neq); % Source term

%$$$$$$ dt(imax,jmax); % Local time step at each node

%$$$$$$ artviscx(imax,jmax); % Artificial viscosity in x-direction

%$$$$$$ artviscy(imax,jmax); % Artificial viscosity in y-direction

res = [0,0,0]; % Iterative residual for each equation

resinit = [0,0,0]; % Initial iterative residual for each equation (from iteration 1)

rL1norm = [0,0,0]; % L1 norm of discretization error for each equation

rL2norm = [0,0,0]; % L2 norm of discretization error for each equation

rLinfnorm = [0,0,0]; % Linfinity norm of discretization error for each equation

rtime = -99.9; % Variable to estimate simulation time

dtmin = 1.0e99; % Minimum time step for a given iteration (initialized large)

x = -99.9; % Temporary variable for x location

y = -99.9; % Temporary variable for y location

% Solution variables initialization with dummy values

% for i=1:imax

% for j=1:jmax

% dt(i,j) = -99.9;

% artviscx(i,j) = -99.9;

% artviscy(i,j) = -99.9;

% for k=1:neq

% u(i,j,k) = -99.9;

% uold(i,j,k) = -99.9;

% s(i,j,k) = -99.9;

% res(k) = -99.9;

% resinit(k) = -99.9;

% res(k) = -99.9;

% rL1norm(k) = -99.9;

% rL2norm(k) = -99.9;

% rLinfnorm(k) = -99.9;

% end

% end

% end

dt = zeros(imax,jmax);

artviscx = zeros(imax,jmax);

artviscy = zeros(imax,jmax);

u = zeros(imax,jmax,neq);

uold = zeros(imax,jmax,neq);

ummsArray = zeros(imax,jmax,neq);

s = zeros(imax,jmax,neq);

res = zeros(neq,1);

resinit = zeros(neq,1);

rL1norm = zeros(neq,1);

rL2norm = zeros(neq,1);

rLinfnorm = zeros(neq,1);

dt(:,:) = -99.9;

artviscx(:,:) = -99.9;

artviscy(:,:) = -99.9;

u(:,:,:) = -99.9;

uold(:,:,:) = -99.9;

s(:,:,:) = -99.9;

res(:) = -99.9;

resinit(:) = -99.9;

rL1norm(:) = -99.9;

rL2norm(:) = -99.9;

rLinfnorm(:) = -99.9;

% Debug output: Uncomment and modify if debugging

%$$$$$$ fp6 = fopen("./Debug.dat","w");

%$$$$$$ fprintf(fp6,"TITLE = \"Debug Data Data\"\n");

%$$$$$$ fprintf(fp6,"variables=\"x(m)\"\"y(m)\"\"visc-x\"\"visc-y\"\n");

%$$$$$$ fprintf(fp6, "zone T=\"n=%d\"\n",n);

%$$$$$$ fprintf(fp6, "I= %d J= %d\n",imax, jmax);

%$$$$$$ fprintf(fp6, "DATAPACKING=POINT\n");

% Set derived input quantities

set\_derived\_inputs();

% Set up headers for output files

output\_file\_headers();

% Set Initial Profile for u vector

[ninit, rtime, resinit] = initial(ninit, rtime, resinit);

% Set Boundary Conditions for u

set\_boundary\_conditions();

% Write out inital conditions to solution file

write\_output(ninit, resinit, rtime);

% Initialize Artificial Viscosity arrays to zero (note: artviscx(i,j) and artviscy(i,j)

artviscx(:,:) = zero;

artviscy(:,:) = zero;

% Evaluate Source Terms Once at Beginning

%(only interior points; will be zero for standard cavity)

compute\_source\_terms();

% Initalizes residual and conv history vectors

rhist\_var=50; %var for how often reshist is stored

[nvec,artn,resvec,convVec]=res\_hist\_init(nmax,rhist\_var);

%========== Main Loop ==========

isConverged = 0;

%% For continuing 129x129 case

imp\_mat =0; % My shit way of continuing/impoting data / restart (restart flag option failed when atempted)

if imp\_mat==1

load('257x257\_LDC\_SGS\_CFL\_002\_Re1000@51324')

ninit=n;rhist\_var=50;

nmax=500000;nmax=n+(nmax\*3); % resets nmax, and changes it to be current n + 500000 itt

artmax=1+ceil(nmax/rhist\_var); % resizes history vecotors

nvec(end+1:artmax)=0;resvec(end+1:artmax,:)=0;convVec(end+1:artmax)=0;

end

for n = ninit:nmax

% Calculate time step

dtmin = compute\_time\_step(dtmin);

% Save u values at time level n (u and uold are 2D arrays)

uold = u;

if isgs==1 % ==Symmetric Gauss Seidel==

% Artificial Viscosity

Compute\_Artificial\_Viscosity();

% Symmetric Gauss-Siedel: Forward Sweep

SGS\_forward\_sweep();

% Set Boundary Conditions for u

set\_boundary\_conditions();

% Artificial Viscosity

Compute\_Artificial\_Viscosity();

% Symmetric Gauss-Siedel: Backward Sweep

SGS\_backward\_sweep();

% Set Boundary Conditions for u

set\_boundary\_conditions();

else

if isgs==0 % ==Point Jacobi==

% Artificial Viscosity

Compute\_Artificial\_Viscosity();

% Point Jacobi: Forward Sweep

point\_Jacobi();

% Set Boundary Conditions for u

set\_boundary\_conditions();

else

fprintf('ERROR: isgs must equal 0 or 1!\n');

return;

end

end

% Pressure Rescaling (based on center point)

pressure\_rescaling();

% Update the time

rtime = rtime + dtmin;

%% store & plot res/conv history

% updates plotting data every 50 itt, updates plot every 500 itt

if ( (mod(n,rhist\_var)==0)||(n==ninit))

[artn,nvec,resvec,convVec]=res\_hist\_vecs(n,conv,res,artn,nvec,resvec,convVec);

end

% plots residuals and conv (updates plot every 500 itt)

if ( (mod(n,500)==0)||(n==ninit) )

figure(1)

plot(nvec(1:artn,:),resvec(1:artn,:))

ylim([0,1])

set(gca, 'YScale', 'log')

figure(2)

plot(nvec(1:artn,:),convVec(1:artn,:),'color','b')

ylim([0,1])

set(gca, 'YScale', 'log')

end

% Check iterative convergence using L2 norms of iterative residuals

[res, resinit, conv] = check\_iterative\_convergence(n, res, resinit, ninit, rtime, dtmin);

if(conv<toler)

fprintf(fp1, '%d %e %e %e %e\n',n, rtime, res(1), res(2), res(3));

isConverged = 1;

break;

end

% Output solution and restart file every 'iterout' steps

if( (mod(n,iterout)==0) )

write\_output(n, resinit, rtime);

end

% save .mat file every 20,000 itt

if ( (mod(n,20000)==0) )

[savename]=SNAME(n);

save(savename)

end

end % ========== End Main Loop ==========

%%

hold off % REMOVE WHEN DONE TROUBLESHOOTING

% resize residual and conv histories

[nvec,resvec,convVec] = Rsize\_res\_hist(artn,nvec,resvec,convVec);

if isConverged == 0

fprintf('Solution failed to converge in %d iterations!!!', nmax);

end

if isConverged == 1

fprintf('Solution converged in %d iterations!!!', n);

end

% Calculate and Write Out Discretization Error Norms (will do this for MMS only)

% Discretization\_Error\_Norms(rL1norm, rL2norm, rLinfnorm);

if imms==1

[DE,rL1norm,rL2norm,rLinfnorm]=Discretization\_Error\_Norms(rL1norm, rL2norm, rLinfnorm);

end

% Output solution and restart file

write\_output(n, resinit, rtime);

% Close open files

fclose(fp1);

fclose(fp2);

%$$$$$$ fclose(fp6); % Uncomment for debug output (

PrsMatrix = u(:,:,1); %output arrays

uvelMatrix = u(:,:,2);

vvelMatrix = u(:,:,3);

toc %end timer function

% save .mat file at end of loop

[savename]=SNAME(n);

save(savename)

% end %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% REMEBER TO

% UN-COMMMENT THIS AND THE FIRST LINE TO MAKE BACK INTO A FUNCTION

% FILE!!!!!!!!

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

%\* All Other Functions \*/

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Function that makes the .mat file save name

function [savename]=SNAME(n)

global imax jmax cfl imms Cx rkappa isgs Re

if imms==1

mmsind='MMS';

else

mmsind='LDC';

end

if isgs==1

sgsind='SGS';

else

sgsind='PJ';

end

cflstr= erase(num2str(cfl),".");

rkstr= erase(num2str(rkappa),".");

c4str= erase(num2str(Cx),".");

savename=[num2str(imax),'x',num2str(jmax),'\_',mmsind,'\_',sgsind,'\_CFL\_',cflstr,'\_Rkap\_',rkstr,'\_C4\_',c4str,'\_Re',num2str(Re),'@',num2str(n),'.mat'];

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Initalizes residual and conv history vectors

function [nvec,artn,resvec,convVec]=res\_hist\_init(nmax,rhist\_var)

%--- Varibles used in res hist function

artmax=1+ceil(nmax/rhist\_var); %length of history vectors

nvec=zeros(artmax,1); % vector that stores the ittoration number

artn=0; % artifical n (artifical collection itteration number)

resvec=zeros(artmax,3); % residual history vector

convVec=zeros(artmax,1); % convergance history vector

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Stores residual and conv histories

function [artn,nvec,resvec,convVec] = res\_hist\_vecs(n,conv,res,artn,nvec,resvec,convVec)

artn=artn+1;

nvec(artn)=n;

resvec(artn,:)=res';

convVec(artn)=conv;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Re-sizes residual and conv histories after sim is done

function [nvec,resvec,convVec] = Rsize\_res\_hist(artn,nvec,resvec,convVec)

resvec=resvec(1:artn,:);

convVec=convVec(1:artn,:);

nvec=nvec(1:artn,:);

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function set\_derived\_inputs(~)

global imax jmax

global one

global Re uinf rho rhoinv xmin xmax ymin ymax

global rlength rmu vel2ref dx dy rpi

rhoinv = one/rho; % Inverse density, 1/rho (m^3/kg) \*/

rlength = xmax - xmin; % Characteristic length (m) [cavity width] \*/

rmu = rho\*uinf\*rlength/Re; % Viscosity (N\*s/m^2) \*/

vel2ref = uinf\*uinf; % Reference velocity squared (m^2/s^2) \*/

dx = (xmax - xmin)/(imax - 1); % Delta x (m) \*/

dy = (ymax - ymin)/(jmax - 1); % Delta y (m) \*/

rpi = acos(-one); % Pi = 3.14159... \*/

fprintf('rho,V,L,mu,Re: %f %f %f %f %f\n',rho,uinf,rlength,rmu,Re);

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function output\_file\_headers(~)

% Uses global variable(s): imms, fp1, fp2

% Note: The vector of primitive variables is:

% u = [p, u, v]^T

% Set up output files (history and solution)

global imms fp1 fp2

fp1 = fopen('./history.dat','w');

fprintf(fp1,'TITLE = "Cavity Iterative Residual History"\n');

fprintf(fp1,'variables="Iteration""Time(s)""Res1""Res2""Res3"\n');

fp2 = fopen('./cavity.dat','w');

fprintf(fp2,'TITLE = "Cavity Field Data"\n');

if (imms==1)

fprintf(fp2,'variables="x(m)""y(m)""p(N/m^2)""u(m/s)""v(m/s)"');

fprintf(fp2,'"p-exact""u-exact""v-exact""DE-p""DE-u""DE-v"\n');

else

if (imms==0)

fprintf(fp2,'variables="x(m)""y(m)""p(N/m^2)""u(m/s)""v(m/s)"\n');

else

fprintf('ERROR! imms must equal 0 or 1!!!\n');

return;

end

end

% Header for Screen Output

fprintf('Iter. Time (s) dt (s) Continuity x-Momentum y-Momentum\n');

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [ninit, rtime, resinit] = initial(ninit, rtime, resinit)

%

%Uses global variable(s): zero, one, irstr, imax, jmax, neq, uinf, pinf

%To modify: ninit, rtime, resinit, u, s

% i % i index (x direction)

% j % j index (y direction)

% k % k index (# of equations)

% x % Temporary variable for x location

% y % Temporary variable for y location

% This subroutine sets inital conditions in the cavity

% Note: The vector of primitive variables is:

% u = (p, u, v)^T

global zero one irstr imax jmax neq uinf pinf xmax xmin ymax ymin

global u s ummsArray

if (irstr==0) % Starting run from scratch

ninit = 1; % set initial iteration to one

rtime = zero; % set initial time to zero

for k = 1:neq

resinit(k) = one;

end

for j = 1:jmax

for i = 1:imax

u(i,j,1) = pinf;

u(i,j,2) = zero;

u(i,j,3) = zero;

s(i,j,1) = zero;

s(i,j,2) = zero;

s(i,j,3) = zero;

end

u(i,jmax,2) = uinf; % Initialize lid (top) to freestream velocity

end

else

if (irstr==1) % Restarting from previous run (file 'restart.in')

fp4 = fopen('./restart.in','r'); % Note: 'restart.in' must exist!

if (fp4==NULL)

fprintf('Error opening restart file. Stopping.\n');

return;

end

% Convert out file to txt file

file\_name = 'restart.out';

[~,f] = fileparts(file\_name);

copyfile(file\_name,[f '.txt']);

% Read data from file

opts = detectImportOptions('restart.txt');

opts.DataLines = [1 1];

ninit\_rtime = readmatrix('restart.txt',opts); % Need to known current iteration # and time value

ninit = ninit\_rtime(1);

rtime = ninit\_rtime(2);

opts.DataLines = [2 2];

resinit\_file = readmatrix('restart.txt',opts); % Needs initial iterative residuals for scaling

resinit(1) = resinit\_file(1);

resinit(2) = resinit\_file(2);

resinit(3) = resinit\_file(3);

opts.DataLines = 3;

A = readmatrix('restart.txt',opts);

uproc = A(:, 3:5);

for j = 1:jmax

u(1:imax,j,1) = uproc((j-1)\*imax+1:j\*imax,1);

u(1:imax,j,2) = uproc((j-1)\*imax+1:j\*imax,2);

u(1:imax,j,3) = uproc((j-1)\*imax+1:j\*imax,3);

end

ninit = ninit + 1;

fprintf('Restarting at iteration %d\n', ninit);

fclose(fp4);

else

printf('ERROR: irstr must equal 0 or 1!\n');

return;

end

end

%initialize the ummsArray with values computed with umms function

for j=1:jmax

for i=1:imax

for k=1:neq

x = (xmax - xmin)\*(i-1)/(imax - 1);

y = (ymax - ymin)\*(j-1)/(jmax - 1);

ummsArray(i,j,k) = umms(x,y,k);

end

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function set\_boundary\_conditions(~)

%

%Uses global variable(s): imms

%To modify: u (via other functions: bndry() and bndrymms())

global imms

% This subroutine determines the appropriate BC routines to call

if (imms==0)

bndry();

else

if (imms==1)

bndrymms();

else

printf('ERROR: imms must equal 0 or 1!\n');

return;

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function bndry(~)

%

%Uses global variable(s): zero, one, two, half, imax, jmax, uinf

%To modify: u

% i % i index (x direction)

% j % j index (y direction)

global zero two half imax jmax uinf

global u

% This applies the cavity boundary conditions

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*Should be good \*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% Left/Right walls (corners will reflect top and bottom bnd cond.)

% Corners not solved in this loop (rely on top/bottom bndcnds to solv)

% remeber u=(p;u;v) NOT (u;v;p)

for j=2:jmax-1

% Right wall

i=imax;

u(i,j,1)=2\*u(i-1,j,1)-u(i-2,j,1);

u(i,j,2)=0;

u(i,j,3)=0;

% Left wall

i=1;

u(i,j,1)=2\*u(i+1,j,1)-u(i+2,j,1);

u(i,j,2)=0;

u(i,j,3)=0;

end

% Top/Bottom walls (corners will reflect top and bottom bnd cond.)

for i=1:imax

% Top wall

j=jmax;

u(i,j,1)=2\*u(i,j-1,1)-u(i,j-2,1);

u(i,j,2)=uinf;

u(i,j,3)=0;

% Bottom wall

j=1;

u(i,j,1)=2\*u(i,j+1,1)-u(i,j+2,1);

u(i,j,2)=0;

u(i,j,3)=0;

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function bndrymms(~)

%

%Uses global variable(s): two, imax, jmax, neq, xmax, xmin, ymax, ymin, rlength

%To modify: u

% i % i index (x direction)

% j % j index (y direction)

% k % k index (# of equations)

% x % Temporary variable for x location

% y % Temporary variable for y location

% This applies the cavity boundary conditions for the manufactured solution

global two imax jmax neq

global u ummsArray

% Side Walls

for j = 2:jmax-1

i = 1;

for k = 1:neq

u(i,j,k) = ummsArray(i,j,k);

end

u(1,j,1) = two\*u(2,j,1) - u(3,j,1); % 2nd Order BC

% u(1,j,1) = u(2,j,1); % 1st Order BC

i=imax;

for k = 1:neq

u(i,j,k) = ummsArray(i,j,k);

end

u(imax,j,1) = two\*u(imax-1,j,1) - u(imax-2,j,1); % 2nd Order BC

% u(imax,j,1) = u(imax-1,j,1); % 1st Order BC

end

% Top/Bottom Walls

for i=1:imax

j = 1;

for k = 1:neq

u(i,j,k) = ummsArray(i,j,k);

end

u(i,1,1) = two\*u(i,2,1) - u(i,3,1); % 2nd Order BC

%$$$$$$ u(i,1,1) = u(i,2,1); % 1st Order BC

j = jmax;

for k = 1:neq

u(i,j,k) = ummsArray(i,j,k);

end

u(i,jmax,1) = two\*u(i,jmax-1,1) - u(i,jmax-2,1); % 2nd Order BC

%$$$$$$ u(i,jmax,1) = u(i,jmax-1,1); % 1st Order BC

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [ummstmp] = umms( x, y, k)

%

%Uses global variable(s): one, rpi, rlength

%Inputs: x, y, k

%To modify: <none>

%Returns: umms

% ummstmp; % Define return value for umms as % precision

% termx % Temp variable

% termy % Temp variable

% termxy % Temp variable

% argx % Temp variable

% argy % Temp variable

% argxy % Temp variable

% This function returns the MMS exact solution

global one rpi rlength

global phi0 phix phiy phixy apx apy apxy fsinx fsiny fsinxy

argx = apx(k)\*rpi\*x/rlength;

argy = apy(k)\*rpi\*y/rlength;

argxy = apxy(k)\*rpi\*x\*y/rlength/rlength;

termx = phix(k)\*(fsinx(k)\*sin(argx)+(one-fsinx(k))\*cos(argx));

termy = phiy(k)\*(fsiny(k)\*sin(argy)+(one-fsiny(k))\*cos(argy));

termxy = phixy(k)\*(fsinxy(k)\*sin(argxy)+(one-fsinxy(k))\*cos(argxy));

ummstmp = phi0(k) + termx + termy + termxy;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function write\_output( n, resinit, rtime)

%

%Uses global variable(s): imax, jmax, new, xmax, xmin, ymax, ymin, rlength, imms

%Uses global variable(s): ninit, u, dt, resinit, rtime

%To modify: <none>

%Writes output and restart files.

% i % i index (x direction)

% j % j index (y direction)

% k % k index (# of equations)

% x % Temporary variable for x location

% y % Temporary variable for y location

global imax jmax xmax xmin ymax ymin imms

global u ummsArray

global fp2 fp3

% Field output

fprintf(fp2, 'zone T="n=%d"\n',n);

fprintf(fp2, 'I= %d J= %d\n',imax, jmax);

fprintf(fp2, 'DATAPACKING=POINT\n');

if (imms==1)

for j=1:jmax

for i=1:imax

x = (xmax - xmin)\*(i-1)/(imax - 1);

y = (ymax - ymin)\*(j-1)/(jmax - 1);

fprintf(fp2,'%e %e %e %e %e %e %e %e %e %e %e\n', x, y, ...

u(i,j,1), u(i,j,2), u(i,j,3), ummsArray(i,j,1), ummsArray(i,j,2), ummsArray(i,j,3), ...

(u(i,j,1)-ummsArray(i,j,1)), (u(i,j,2)-ummsArray(i,j,2)), (u(i,j,3)-ummsArray(i,j,3)));

end

end

else

if (imms==0)

for j=1:jmax

for i=1:imax

x = (xmax - xmin)\*(i-1)/(imax - 1);

y = (ymax - ymin)\*(j-1)/(jmax - 1);

fprintf(fp2,'%e %e %e %e %e\n', x, y, ...

u(i,j,1), u(i,j,2), u(i,j,3));

end

end

else

fprintf('ERROR: imms must equal 0 or 1!\n');

return;

end

end

% Restart file: overwrites every 'iterout' iteration

fp3 = fopen('./restart.out','w');

fprintf(fp3,'%d %e\n', n, rtime);

fprintf(fp3,'%e %e %e\n', resinit(1), resinit(2), resinit(3));

for j=1:jmax

for i=1:imax

x = (xmax - xmin)\*(i-1)/(imax - 1);

y = (ymax - ymin)\*(j-1)/(jmax - 1);

fprintf(fp3,'%e %e %e %e %e\n', x, y, ...

u(i,j,1), u(i,j,2), u(i,j,3));

end

end

fclose(fp3);

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function compute\_source\_terms(~)

%

%Uses global variable(s): imax, jmax, imms, rlength, xmax, xmin, ymax, ymin

%To modify: s (source terms)

% i % i index (x direction)

% j % j index (y direction)

% x % Temporary variable for x location

% y % Temporary variable for y location

% Evaluate Source Terms Once at Beginning (only %erior po%s; will be zero for standard cavity)

global imax jmax imms xmax xmin ymax ymin

global s

for j=2:jmax-1

for i=2:imax-1

x = (xmax - xmin)\*(i-1)/(imax - 1);

y = (ymax - ymin)\*(j-1)/(jmax - 1);

s(i,j,1) = (imms)\*srcmms\_mass(x,y);

s(i,j,2) = (imms)\*srcmms\_xmtm(x,y);

s(i,j,3) = (imms)\*srcmms\_ymtm(x,y);

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [srcmasstmp] = srcmms\_mass( x, y)

%

%Uses global variable(s): rho, rpi, rlength

%Inputs: x, y

%To modify: <none>

%Returns: srcmms\_mass

% srcmasstmp; % Define return value for srcmms\_mass as % precision

% dudx; % Temp variable: u velocity gradient in x direction

% dvdy; % Temp variable: v velocity gradient in y direction

% This function returns the MMS mass source term

global rho rpi rlength

global phix phiy phixy apx apy apxy

dudx = phix(2)\*apx(2)\*rpi/rlength\*cos(apx(2)\*rpi\*x/rlength) ...

+ phixy(2)\*apxy(2)\*rpi\*y/rlength/rlength ...

\* cos(apxy(2)\*rpi\*x\*y/rlength/rlength);

dvdy = -phiy(3)\*apy(3)\*rpi/rlength\*sin(apy(3)\*rpi\*y/rlength) ...

- phixy(3)\*apxy(3)\*rpi\*x/rlength/rlength ...

\* sin(apxy(3)\*rpi\*x\*y/rlength/rlength);

srcmasstmp = rho\*dudx + rho\*dvdy;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [srcxmtmtmp] = srcmms\_xmtm( x, y)

%

%Uses global variable(s): rho, rpi, rmu, rlength

%Inputs: x, y

%To modify: <none>

%Returns: srcmms\_xmtm

% srcxmtmtmp; % Define return value for srcmms\_xmtm as % precision

% dudx; % Temp variable: u velocity gradient in x direction

% dudy; % Temp variable: u velocity gradient in y direction

% termx; % Temp variable

% termy; % Temp variable

% termxy; % Temp variable

% uvel; % Temp variable: u velocity

% vvel; % Temp variable: v velocity

% dpdx; % Temp variable: pressure gradient in x direction

% d2udx2; % Temp variable: 2nd derivative of u velocity in x direction

% d2udy2; % Temp variable: 2nd derivative of u velocity in y direction

%This function returns the MMS x-momentum source term

global rho rpi rmu rlength

global phi0 phix phiy phixy apx apy apxy

termx = phix(2)\*sin(apx(2)\*rpi\*x/rlength);

termy = phiy(2)\*cos(apy(2)\*rpi\*y/rlength);

termxy = phixy(2)\*sin(apxy(2)\*rpi\*x\*y/rlength/rlength);

uvel = phi0(2) + termx + termy + termxy;

termx = phix(3)\*cos(apx(3)\*rpi\*x/rlength);

termy = phiy(3)\*cos(apy(3)\*rpi\*y/rlength);

termxy = phixy(3)\*cos(apxy(3)\*rpi\*x\*y/rlength/rlength);

vvel = phi0(3) + termx + termy + termxy;

dudx = phix(2)\*apx(2)\*rpi/rlength\*cos(apx(2)\*rpi\*x/rlength) ...

+ phixy(2)\*apxy(2)\*rpi\*y/rlength/rlength ...

\* cos(apxy(2)\*rpi\*x\*y/rlength/rlength);

dudy = -phiy(2)\*apy(2)\*rpi/rlength\*sin(apy(2)\*rpi\*y/rlength) ...

+ phixy(2)\*apxy(2)\*rpi\*x/rlength/rlength ...

\* cos(apxy(2)\*rpi\*x\*y/rlength/rlength);

dpdx = -phix(1)\*apx(1)\*rpi/rlength\*sin(apx(1)\*rpi\*x/rlength) ...

+ phixy(1)\*apxy(1)\*rpi\*y/rlength/rlength ...

\* cos(apxy(1)\*rpi\*x\*y/rlength/rlength);

d2udx2 = -phix(2)\*((apx(2)\*rpi/rlength).^2) ...

\* sin(apx(2)\*rpi\*x/rlength) ...

- phixy(2)\*((apxy(2)\*rpi\*y/rlength/rlength).^2) ...

\* sin(apxy(2)\*rpi\*x\*y/rlength/rlength);

d2udy2 = -phiy(2)\*((apy(2)\*rpi/rlength).^2) ...

\* cos(apy(2)\*rpi\*y/rlength) ...

- phixy(2)\*((apxy(2)\*rpi\*x/rlength/rlength).^2) ...

\* sin(apxy(2)\*rpi\*x\*y/rlength/rlength);

srcxmtmtmp = rho\*uvel\*dudx + rho\*vvel\*dudy + dpdx ...

- rmu\*( d2udx2 + d2udy2 );

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [srcymtmtmp] = srcmms\_ymtm( x, y)

%

%Uses global variable(s): rho, rpi, rmu, rlength

%Inputs: x, y

%To modify: <none>

%Returns: srcmms\_ymtm

% srcymtmtmp; % Define return value for srcmms\_ymtm as % precision

% dvdx; % Temp variable: v velocity gradient in x direction

% dvdy; % Temp variable: v velocity gradient in y direction

% termx; % Temp variable

% termy; % Temp variable

% termxy; % Temp variable

% uvel; % Temp variable: u velocity

% vvel; % Temp variable: v velocity

% dpdy; % Temp variable: pressure gradient in y direction

% d2vdx2; % Temp variable: 2nd derivative of v velocity in x direction

% d2vdy2; % Temp variable: 2nd derivative of v velocity in y direction

% This function returns the MMS y-momentum source term

global rho rpi rmu rlength

global phi0 phix phiy phixy apx apy apxy

termx = phix(2)\*sin(apx(2)\*rpi\*x/rlength);

termy = phiy(2)\*cos(apy(2)\*rpi\*y/rlength);

termxy = phixy(2)\*sin(apxy(2)\*rpi\*x\*y/rlength/rlength);

uvel = phi0(2) + termx + termy + termxy;

termx = phix(3)\*cos(apx(3)\*rpi\*x/rlength);

termy = phiy(3)\*cos(apy(3)\*rpi\*y/rlength);

termxy = phixy(3)\*cos(apxy(3)\*rpi\*x\*y/rlength/rlength);

vvel = phi0(3) + termx + termy + termxy;

dvdx = -phix(3)\*apx(3)\*rpi/rlength\*sin(apx(3)\*rpi\*x/rlength) ...

- phixy(3)\*apxy(3)\*rpi\*y/rlength/rlength ...

\* sin(apxy(3)\*rpi\*x\*y/rlength/rlength);

dvdy = -phiy(3)\*apy(3)\*rpi/rlength\*sin(apy(3)\*rpi\*y/rlength) ...

- phixy(3)\*apxy(3)\*rpi\*x/rlength/rlength ...

\* sin(apxy(3)\*rpi\*x\*y/rlength/rlength);

dpdy = phiy(1)\*apy(1)\*rpi/rlength\*cos(apy(1)\*rpi\*y/rlength) ...

+ phixy(1)\*apxy(1)\*rpi\*x/rlength/rlength ...

\* cos(apxy(1)\*rpi\*x\*y/rlength/rlength);

d2vdx2 = -phix(3)\*((apx(3)\*rpi/rlength).^2) ...

\* cos(apx(3)\*rpi\*x/rlength) ...

- phixy(3)\*((apxy(3)\*rpi\*y/rlength/rlength).^2) ...

\* cos(apxy(3)\*rpi\*x\*y/rlength/rlength);

d2vdy2 = -phiy(3)\*((apy(3)\*rpi/rlength).^2) ...

\* cos(apy(3)\*rpi\*y/rlength) ...

- phixy(3)\*((apxy(3)\*rpi\*x/rlength/rlength).^2) ...

\* cos(apxy(3)\*rpi\*x\*y/rlength/rlength);

srcymtmtmp = rho\*uvel\*dvdx + rho\*vvel\*dvdy + dpdy ...

- rmu\*( d2vdx2 + d2vdy2 );

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [dtmin] = compute\_time\_step(dtmin)

%

%Uses global variable(s): one, two, four, half, fourth

%Uses global variable(s): vel2ref, rmu, rho, dx, dy, cfl, rkappa, imax, jmax

%Uses: u

%To Modify: dt, dtmin

% i % i index (x direction)

% j % j index (y direction)

% dtvisc % Viscous time step stability criteria (constant over domain)

% uvel2 % Local velocity squared

% beta2 % Beta squared paramete for time derivative preconditioning

% lambda\_x % Max absolute value eigenvalue in (x,t)

% lambda\_y % Max absolute value eigenvalue in (y,t)

% lambda\_max % Max absolute value eigenvalue (used in convective time step computation)

% dtconv % Local convective time step restriction

global four half fourth

global vel2ref rmu rho dx dy cfl rkappa imax jmax

global u dt

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*Should be mostly coded \*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

for j=2:jmax-1

for i=2:imax-1

uvel2 = (sqrt((u(i,j,2).^2)+(u(i,j,3).^2))^2);

beta2 = max(uvel2,vel2ref\*rkappa);

lambda\_x = half\*(abs(u(i,j,2)+sqrt((u(i,j,2).^2)+(four\*beta2))));

lambda\_y = half\*(abs(u(i,j,3)+sqrt((u(i,j,3).^2)+(four\*beta2))));

lambda\_max = max(lambda\_x,lambda\_y);

dtconv = min(dx,dy)/lambda\_max;

dtvisc = (dx\*dy)/(4\*(rmu/rho));

dt(i,j) = cfl\*min(dtconv,dtvisc);

dtmin = min(dtmin,dt(i,j));

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function Compute\_Artificial\_Viscosity(~)

%

%Uses global variable(s): zero, one, two, four, six, half, fourth

%Uses global variable(s): imax, jmax, lim, rho, dx, dy, Cx, Cy, Cx2, Cy2, fsmall, vel2ref, rkappa

%Uses: u

%To Modify: artviscx, artviscy

% i % i index (x direction)

% j % j index (y direction)

% uvel2 % Local velocity squared

% beta2 % Beta squared paramete for time derivative preconditioning

% lambda\_x % Max absolute value e-value in (x,t)

% lambda\_y % Max absolute value e-value in (y,t)

% d4pdx4 % 4th derivative of pressure w.r.t. x

% d4pdy4 % 4th derivative of pressure w.r.t. y

% % d2pdx2 % 2nd derivative of pressure w.r.t. x [these are not used]

% % d2pdy2 % 2nd derivative of pressure w.r.t. y [these are not used]

% % pfunct1 % Temporary variable for 2nd derivative damping [these are

% not used]

% % pfunct2 % Temporary variable for 2nd derivative damping [these are

% not used]

global two four six half three

global imax jmax lim rho dx dy Cx Cy Cx2 Cy2 fsmall vel2ref rkappa

global u

global artviscx artviscy

frtn=14.0;twntsx=26.0;twntfr=24.0;elevn=11.0;

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*Should be mostly coded \*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

for j=2:jmax-1

for i=2:imax-1

uvel2 = (sqrt((u(i,j,2).^2)+(u(i,j,3).^2))^2); % Velocity squared

beta2 = max(uvel2,vel2ref\*rkappa); % Beta squared parameter for time derivative preconditioning

lambda\_x = half\*(abs(u(i,j,2)+sqrt((u(i,j,2).^2)+(four\*beta2)))); % Max absolute value e-value in (x,t)

lambda\_y = half\*(abs(u(i,j,3)+sqrt((u(i,j,3).^2)+(four\*beta2)))); % Max absolute value e-value in (y,t)

% wall conditions using forward/backward difference for pressure (2nd order 4th derivitive)

if i==2||i==imax-1||j==2||j==jmax-1

if i==2

if j==2 % bottom left corner

d4pdx4 = (three\*u(i,j,1)-frtn\*u(i+1,j,1)+twntsx\*u(i+2,j,1)...

-twntfr\*u(i+3,j,1)+elevn\*u(i+4,j,1)-two\*u(i+5,j,1))/(dx^4);

d4pdy4 = (three\*u(i,j,1)-frtn\*u(i,j+1,1)+twntsx\*u(i,j+2,1)...

-twntfr\*u(i,j+3,1)+elevn\*u(i,j+4,1)-two\*u(i,j+5,1))/(dy^4);

elseif j==jmax-1 % top left corner

d4pdx4 = (three\*u(i,j,1)-frtn\*u(i+1,j,1)+twntsx\*u(i+2,j,1)...

-twntfr\*u(i+3,j,1)+elevn\*u(i+4,j,1)-two\*u(i+5,j,1))/(dx^4);

d4pdy4 = (three\*u(i,j,1)-frtn\*u(i,j-1,1)+twntsx\*u(i,j-2,1)...

-twntfr\*u(i,j-3,1)+elevn\*u(i,j-4,1)-two\*u(i,j-5,1))/(dy^4);

else % left wall (not corner)

d4pdx4 = (three\*u(i,j,1)-frtn\*u(i+1,j,1)+twntsx\*u(i+2,j,1)...

-twntfr\*u(i+3,j,1)+elevn\*u(i+4,j,1)-two\*u(i+5,j,1))/(dx^4);

d4pdy4 = (u(i,j+2,1)-four\*u(i,j+1,1)+six\*u(i,j,1)-four\*u(i,j-1,1)...

+u(i,j-2,3))/(dy^4);

end

end

if i==imax-1

if j==2 % bottom right corner

d4pdx4 = (three\*u(i,j,1)-frtn\*u(i-1,j,1)+twntsx\*u(i-2,j,1)...

-twntfr\*u(i-3,j,1)+elevn\*u(i-4,j,1)-two\*u(i-5,j,1))/(dx^4);

d4pdy4 = (three\*u(i,j,1)-frtn\*u(i,j+1,1)+twntsx\*u(i,j+2,1)...

-twntfr\*u(i,j+3,1)+elevn\*u(i,j+4,1)-two\*u(i,j+5,1))/(dy^4);

elseif j==jmax-1 % top right corner

d4pdx4 = (three\*u(i,j,1)-frtn\*u(i-1,j,1)+twntsx\*u(i-2,j,1)...

-twntfr\*u(i-3,j,1)+elevn\*u(i-4,j,1)-two\*u(i-5,j,1))/(dx^4);

d4pdy4 = (three\*u(i,j,1)-frtn\*u(i,j-1,1)+twntsx\*u(i,j-2,1)...

-twntfr\*u(i,j-3,1)+elevn\*u(i,j-4,1)-two\*u(i,j-5,1))/(dy^4);

else % right wall

d4pdx4 = (three\*u(i,j,1)-frtn\*u(i-1,j,1)+twntsx\*u(i-2,j,1)...

-twntfr\*u(i-3,j,1)+elevn\*u(i-4,j,1)-two\*u(i-5,j,1))/(dx^4);

d4pdy4 = (u(i,j+2,1)-four\*u(i,j+1,1)+six\*u(i,j,1)-four\*u(i,j-1,1)...

+u(i,j-2,1))/(dy^4);

end

end

if i~=2 && i~=imax-1

if j==2 % bottom wall % must be bottom or top wall now

d4pdx4 = (u(i+2,j,1)-four\*u(i+1,j,1)+six\*u(i,j,1)-four\*u(i-1,j,1)...

+u(i-2,j,1))/(dx^4);

d4pdy4 = (three\*u(i,j,1)-frtn\*u(i,j+1,1)+twntsx\*u(i,j+2,1)...

-twntfr\*u(i,j+3,1)+elevn\*u(i,j+4,1)-two\*u(i,j+5,1))/(dy^4);

end

if j==jmax-1 % top wall

d4pdx4 = (u(i+2,j,1)-four\*u(i+1,j,1)+six\*u(i,j,1)-four\*u(i-1,j,1)...

+u(i-2,j,1))/(dx^4);

d4pdy4 = (three\*u(i,j,1)-frtn\*u(i,j-1,1)+twntsx\*u(i,j-2,1)...

-twntfr\*u(i,j-3,1)+elevn\*u(i,j-4,1)-two\*u(i,j-5,1))/(dy^4);

end

end

else

% interior points

d4pdx4 = (u(i+2,j,1)-four\*u(i+1,j,1)+six\*u(i,j,1)-four\*u(i-1,j,1)...

+u(i-2,j,1))/(dx^4); % 4th derivative of pressure w.r.t. x

d4pdy4 = (u(i,j+2,1)-four\*u(i,j+1,1)+six\*u(i,j,1)-four\*u(i,j-1,1)...

+u(i,j-2,1))/(dy^4); % 4th derivative of pressure w.r.t. y

end

artviscx(i,j) = ((-lambda\_x\*Cx\*(dx^3))/beta2)\*d4pdx4;

artviscy(i,j) = ((-lambda\_y\*Cy\*(dy^3))/beta2)\*d4pdy4;

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function SGS\_forward\_sweep(~)

%

%Uses global variable(s): two, three, six, half

%Uses global variable(s): imax, imax, jmax, ipgorder, rho, rhoinv, dx, dy, rkappa, ...

% xmax, xmin, ymax, ymin, rmu, vel2ref

%Uses: artviscx, artviscy, dt, s

%To Modify: u

% i % i index (x direction)

% j % j index (y direction)

% dpdx % First derivative of pressure w.r.t. x

% dudx % First derivative of x velocity w.r.t. x

% dvdx % First derivative of y velocity w.r.t. x

% dpdy % First derivative of pressure w.r.t. y

% dudy % First derivative of x velocity w.r.t. y

% dvdy % First derivative of y velocity w.r.t. y

% d2udx2 % Second derivative of x velocity w.r.t. x

% d2vdx2 % Second derivative of y velocity w.r.t. x

% d2udy2 % Second derivative of x velocity w.r.t. y

% d2vdy2 % Second derivative of y velocity w.r.t. y

% beta2 % Beta squared parameter for time derivative preconditioning

% uvel2 % Velocity squared

global two half

global imax jmax rho rhoinv dx dy rkappa rmu vel2ref

global artviscx artviscy dt s u

% NOTE: s is "Sorce term" not artifical viscosity

% Symmetric Gauss-Siedel: Forward Sweep

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\* Should be mostly coded \*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

tdx=two\*dx;tdy=two\*dy;dxsqd=dx^2;dysqd=dy^2;

for j=2:jmax-1

for i=2:imax-1

dpdx = (u(i+1,j,1)-u(i-1,j,1))/(tdx); % First derivative of pressure w.r.t. x

dudx = (u(i+1,j,2)-u(i-1,j,2))/(tdx); % First derivative of x velocity w.r.t. x

dvdx = (u(i+1,j,3)-u(i-1,j,3))/(tdx); % First derivative of y velocity w.r.t. x

dpdy = (u(i,j+1,1)-u(i,j-1,1))/(tdy); % First derivative of pressure w.r.t. y

dudy = (u(i,j+1,2)-u(i,j-1,2))/(tdy); % First derivative of x velocity w.r.t. y

dvdy = (u(i,j+1,3)-u(i,j-1,3))/(tdy); % First derivative of y velocity w.r.t. y

d2udx2 = (u(i+1,j,2)-(two\*u(i,j,2))+u(i-1,j,2))/(dxsqd); % Second derivative of x velocity w.r.t. x

d2vdx2 = (u(i+1,j,3)-(two\*u(i,j,3))+u(i-1,j,3))/(dxsqd); % Second derivative of y velocity w.r.t. x

d2udy2 = (u(i,j+1,2)-(two\*u(i,j,2))+u(i,j-1,2))/(dysqd); % Second derivative of x velocity w.r.t. y

d2vdy2 = (u(i,j+1,3)-(two\*u(i,j,3))+u(i,j-1,3))/(dysqd); % Second derivative of y velocity w.r.t. y

uvel2 = (sqrt((u(i,j,2).^2)+(u(i,j,3).^2))^2); % Velocity squared

beta2 = max(uvel2,vel2ref\*rkappa); % Beta squared parameter for time derivative preconditioning

% discretization (mass,x-mnt,y-mnt)

u(i,j,1) = u(i,j,1)-(beta2\*dt(i,j)\*((rho\*dudx)+(rho\*dvdy)-artviscx(i,j)...

-artviscy(i,j)-s(i,j,1))); % Continuity (mass)

u(i,j,2) = u(i,j,2)-(dt(i,j)\*rhoinv\*((rho\*u(i,j,2)\*dudx)+(rho\*u(i,j,3)\*dudy)...

+dpdx-(rmu\*d2udx2)-(rmu\*d2udy2)-s(i,j,2))); % X - Momentum

u(i,j,3) = u(i,j,3)-(dt(i,j)\*rhoinv\*((rho\*u(i,j,2)\*dvdx)+(rho\*u(i,j,3)\*dvdy)...

+dpdy-(rmu\*d2vdx2)-(rmu\*d2vdy2)-s(i,j,3))); % Y - Momentum

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function SGS\_backward\_sweep(~)

%

%Uses global variable(s): two, three, six, half

%Uses global variable(s): imax, imax, jmax, ipgorder, rho, rhoinv, dx, dy, rkappa, ...

% xmax, xmin, ymax, ymin, rmu, vel2ref

%Uses: artviscx, artviscy, dt, s

%To Modify: u

% i % i index (x direction)

% j % j index (y direction)

% dpdx % First derivative of pressure w.r.t. x

% dudx % First derivative of x velocity w.r.t. x

% dvdx % First derivative of y velocity w.r.t. x

% dpdy % First derivative of pressure w.r.t. y

% dudy % First derivative of x velocity w.r.t. y

% dvdy % First derivative of y velocity w.r.t. y

% d2udx2 % Second derivative of x velocity w.r.t. x

% d2vdx2 % Second derivative of y velocity w.r.t. x

% d2udy2 % Second derivative of x velocity w.r.t. y

% d2vdy2 % Second derivative of y velocity w.r.t. y

% beta2 % Beta squared parameter for time derivative preconditioning

% uvel2 % Velocity squared

global two half

global imax jmax rho rhoinv dx dy rkappa rmu vel2ref

global artviscx artviscy dt s u

% Symmetric Gauss-Siedel: Backward Sweep

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\* Should be mostly coded \*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

tdx=two\*dx;tdy=two\*dy;dxsqd=dx^2;dysqd=dy^2;

for j=jmax-1:2

for i=imax-1:2

dpdx = (u(i+1,j,1)-u(i-1,j,1))/(tdx); % First derivative of pressure w.r.t. x

dudx = (u(i+1,j,2)-u(i-1,j,2))/(tdx); % First derivative of x velocity w.r.t. x

dvdx = (u(i+1,j,3)-u(i-1,j,3))/(tdx); % First derivative of y velocity w.r.t. x

dpdy = (u(i,j+1,1)-u(i,j-1,1))/(tdy); % First derivative of pressure w.r.t. y

dudy = (u(i,j+1,2)-u(i,j-1,2))/(tdy); % First derivative of x velocity w.r.t. y

dvdy = (u(i,j+1,3)-u(i,j-1,3))/(tdy); % First derivative of y velocity w.r.t. y

d2udx2 = (u(i+1,j,2)-(two\*u(i,j,2))+u(i-1,j,2))/(dxsqd); % Second derivative of x velocity w.r.t. x

d2vdx2 = (u(i+1,j,3)-(two\*u(i,j,3))+u(i-1,j,3))/(dxsqd); % Second derivative of y velocity w.r.t. x

d2udy2 = (u(i,j+1,2)-(two\*u(i,j,2))+u(i,j-1,2))/(dysqd); % Second derivative of x velocity w.r.t. y

d2vdy2 = (u(i,j+1,3)-(two\*u(i,j,3))+u(i,j-1,3))/(dysqd); % Second derivative of y velocity w.r.t. y

uvel2 = (sqrt((u(i,j,2).^2)+(u(i,j,3).^2))^2); % Velocity squared

beta2 = max(uvel2,vel2ref\*rkappa); % Beta squared parameter for time derivative preconditioning

% discretization (mass,x-mnt,y-mnt)

u(i,j,1) = u(i,j,1)-(beta2\*dt(i,j)\*((rho\*dudx)+(rho\*dvdy)-artviscx(i,j)...

-artviscy(i,j)-s(i,j,1))); % Continuity (mass)

u(i,j,2) = u(i,j,2)-(dt(i,j)\*rhoinv\*((rho\*u(i,j,2)\*dudx)+(rho\*u(i,j,3)\*dudy)...

+dpdx-(rmu\*d2udx2)-(rmu\*d2udy2)-s(i,j,2))); % X - Momentum

u(i,j,3) = u(i,j,3)-(dt(i,j)\*rhoinv\*((rho\*u(i,j,2)\*dvdx)+(rho\*u(i,j,3)\*dvdy)...

+dpdy-(rmu\*d2vdx2)-(rmu\*d2vdy2)-s(i,j,3))); % Y - Momentum

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function point\_Jacobi(~)

%

%Uses global variable(s): two, three, six, half

%Uses global variable(s): imax, imax, jmax, ipgorder, rho, rhoinv, dx, dy, rkappa, ...

% xmax, xmin, ymax, ymin, rmu, vel2ref

%Uses: uold, artviscx, artviscy, dt, s

%To Modify: u

% i % i index (x direction)

% j % j index (y direction)

% dpdx % First derivative of pressure w.r.t. x

% dudx % First derivative of x velocity w.r.t. x

% dvdx % First derivative of y velocity w.r.t. x

% dpdy % First derivative of pressure w.r.t. y

% dudy % First derivative of x velocity w.r.t. y

% dvdy % First derivative of y velocity w.r.t. y

% d2udx2 % Second derivative of x velocity w.r.t. x

% d2vdx2 % Second derivative of y velocity w.r.t. x

% d2udy2 % Second derivative of x velocity w.r.t. y

% d2vdy2 % Second derivative of y velocity w.r.t. y

% beta2 % Beta squared parameter for time derivative preconditioning

% uvel2 % Velocity squared

global two half

global imax jmax rho rhoinv dx dy rkappa rmu vel2ref

global u uold artviscx artviscy dt s

% Point Jacobi method

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*ADD CODING HERE FOR INTRO CFD STUDENTS\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

tdx=two\*dx;tdy=two\*dy;dxsqd=dx^2;dysqd=dy^2;

for j=2:jmax-1

for i=2:imax-1

dpdx = (uold(i+1,j,1)-uold(i-1,j,1))/(tdx); % First derivative of pressure w.r.t. x

dudx = (uold(i+1,j,2)-uold(i-1,j,2))/(tdx); % First derivative of x velocity w.r.t. x

dvdx = (uold(i+1,j,3)-uold(i-1,j,3))/(tdx); % First derivative of y velocity w.r.t. x

dpdy = (uold(i,j+1,1)-uold(i,j-1,1))/(tdy); % First derivative of pressure w.r.t. y

dudy = (uold(i,j+1,2)-uold(i,j-1,2))/(tdy); % First derivative of x velocity w.r.t. y

dvdy = (uold(i,j+1,3)-uold(i,j-1,3))/(tdy); % First derivative of y velocity w.r.t. y

d2udx2 = (uold(i+1,j,2)-(two\*uold(i,j,2))+uold(i-1,j,2))/(dxsqd); % Second derivative of x velocity w.r.t. x

d2vdx2 = (uold(i+1,j,3)-(two\*uold(i,j,3))+uold(i-1,j,3))/(dxsqd); % Second derivative of y velocity w.r.t. x

d2udy2 = (uold(i,j+1,2)-(two\*uold(i,j,2))+uold(i,j-1,2))/(dysqd); % Second derivative of x velocity w.r.t. y

d2vdy2 = (uold(i,j+1,3)-(two\*uold(i,j,3))+uold(i,j-1,3))/(dysqd); % Second derivative of y velocity w.r.t. y

uvel2 = (sqrt((uold(i,j,2).^2)+(uold(i,j,3).^2))^2); % Velocity squared

beta2 = max(uvel2,vel2ref\*rkappa); % Beta squared parameter for time derivative preconditioning

% discretization (mass,x-mnt,y-mnt)

u(i,j,1) = uold(i,j,1)-(beta2\*dt(i,j)\*((rho\*dudx)+(rho\*dvdy)-artviscx(i,j)...

-artviscy(i,j)-s(i,j,1))); % Continuity (mass)

u(i,j,2) = uold(i,j,2)-(dt(i,j)\*rhoinv\*((rho\*uold(i,j,2)\*dudx)+(rho\*uold(i,j,3)\*dudy)...

+dpdx-(rmu\*d2udx2)-(rmu\*d2udy2)-s(i,j,2))); % X - Momentum

u(i,j,3) = uold(i,j,3)-(dt(i,j)\*rhoinv\*((rho\*uold(i,j,2)\*dvdx)+(rho\*uold(i,j,3)\*dvdy)...

+dpdy-(rmu\*d2vdx2)-(rmu\*d2vdy2)-s(i,j,3))); % Y - Momentum

end

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function pressure\_rescaling(~)

%

%Uses global variable(s): imax, jmax, imms, xmax, xmin, ymax, ymin, rlength, pinf

%To Modify: u

% i % i index (x direction)

% j % j index (y direction)

% iref % i index location of pressure rescaling point

% jref % j index location of pressure rescaling point

% x % Temporary variable for x location

% y % Temporary variable for y location

% deltap % delta\_pressure for rescaling all values

global imax jmax imms xmax xmin ymax ymin pinf

global u

iref = (imax-1)/2+1; % Set reference pressure to center of cavity

jref = (jmax-1)/2+1;

if (imms==1)

x = (xmax - xmin)\*(iref-1)/(imax - 1);

y = (ymax - ymin)\*(jref-1)/(jmax - 1);

deltap = u(iref,jref,1) - umms(x,y,1); % Constant in MMS

else

deltap = u(iref,jref,1) - pinf; % Reference pressure

end

j=1:jmax;

i=1:imax;

u(i,j,1) = u(i,j,1) - deltap;

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [res, resinit, conv] = check\_iterative\_convergence...

(n, res, resinit, ninit, rtime, dtmin)

%

%Uses global variable(s): zero

%Uses global variable(s): imax, jmax, neq, fsmall

%Uses: n, u, uold, dt, res, resinit, ninit, rtime, dtmin

%To modify: conv

% i % i index (x direction)

% j % j index (y direction)

% k % k index (# of equations)

global zero

global imax jmax neq fsmall

global u uold dt fp1

% Compute iterative residuals to monitor iterative convergence

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*Think that this might need to change...\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

R=zeros(imax,jmax,neq);

for j=1:jmax

for i=1:imax

for k=1:neq

R(i,j,k)=(u(i,j,k)-uold(i,j,k))/dt(i,j);

end

end

end

% R NEEDS TO BE CONVERTED INTO A VECTOR CONTAING ALL NODAL INFO

% nvm this should do

res(1)=rms(R(:,:,1),"all");

res(2)=rms(R(:,:,2),"all");

res(3)=rms(R(:,:,3),"all");

if n<=10

%Resinit rescalling here

if res(1)>resinit(1) ||res(2)>resinit(2) || res(3)>resinit(3)

if res(1)>resinit(1)

resinit(1)=res(1);

end

if res(2)>resinit(2)

resinit(2)=res(2);

end

if res(3)>resinit(3)

resinit(3)=res(3);

end

end

end

conv=max(abs(res./resinit));

% Write iterative residuals every 10 iterations

if ( (mod(n,10)==0)||(n==ninit) )

fprintf(fp1, '%d %e %e %e %e %e\n',n, rtime, res(1), res(2), res(3), conv );

fprintf('%d %e %e %e %e %e %e\n',n, rtime, dtmin, res(1), res(2), res(3), conv );

% Maybe a need to format this better

end

% Write header for iterative residuals every 200 iterations

if ( (mod(n,200)==0)||(n==ninit) )

fprintf('Iter. Time (s) dt (s) Continuity x-Momentum y-Momentum Conv\n');

end

end

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [DE,rL1norm,rL2norm,rLinfnorm]=Discretization\_Error\_Norms(rL1norm, rL2norm, rLinfnorm)

%

%Uses global variable(s): zero

%Uses global variable(s): imax, jmax, neq, imms, xmax, xmin, ymax, ymin, rlength

%Uses: u

%To modify: rL1norm, rL2norm, rLinfnorm

% i % i index (x direction)

% j % j index (y direction)

% k % k index (# of equations)

% x % Temporary variable for x location

% y % Temporary variable for y location

% DE % Discretization error (absolute value)

global zero imax jmax neq imms xmax xmin ymax ymin u

if imms==1

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*should be ehh? \*\*\*\*\*\*\*\*\*\*\*\* \*/

% !\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

% re making ummsArray incase inital array was changed since begining of

% code

% this part of the code is coppied from function initial earlier in the code

ummsArray\_2=zeros(imax,jmax,neq);

for j=1:jmax

for i=1:imax

for k=1:neq

x = (xmax - xmin)\*(i-1)/(imax - 1);

y = (ymax - ymin)\*(j-1)/(jmax - 1);

ummsArray\_2(i,j,k) = umms(x,y,k);

end

end

end

% Test to add this back or just call the global array ummsArray

%% look into why L1norm is giving negative values

N=imax\*jmax;

DE=zeros(imax,jmax,neq);

for k=1:neq

DE(:,:,k)=u(:,:,k)-ummsArray\_2(:,:,k);

end

rL1norm = [sum(u(:,:,1)-ummsArray\_2(:,:,1),'all')/N;sum(u(:,:,2)-ummsArray\_2(:,:,2),'all')/N;...

sum(u(:,:,3)-ummsArray\_2(:,:,3),'all')/N];

rL2norm = [rms(u(:,:,1)-ummsArray\_2(:,:,1),"all");rms(u(:,:,2)-ummsArray\_2(:,:,2),"all");...

rms(u(:,:,3)-ummsArray\_2(:,:,3),"all")];

rLinfnorm = [max(u(:,:,1)-ummsArray\_2(:,:,1),[],'all');max(u(:,:,2)-ummsArray\_2(:,:,2),[],'all');...

max(u(:,:,3)-ummsArray\_2(:,:,3),[],'all')];

end

end