Part 2: Functions defined first; see execution loop below functions

%%% write unsteady coefficients in terms of steady coefficients %%%

function unsteady\_coeffs\_LHS

% call global variables needed

global awe aso ano aea ap q den dx dy dt phi tmethod

% globals to be written by this function

global awet asot anot aeat apt

if tmethod == 1; % Explicit Euler

awet = zeros(size(awet));

asot = zeros(size(asot));

anot = zeros(size(anot));

aeat = zeros(size(aeat));

apt = den\*dx\*dy/dt \* ones(size(apt))

elseif tmethod == 2; % Implicit Euler

awet = awe;

asot = aso;

anot = ano;

aeat = aea;

apt = den\*dx\*dy/dt\*ones(size(apt)) + ap;

else; % if tmethod == 3; % Crank-Nicolson

awet = awe/2;

asot = aso/2;

anot = ano/2;

aeat = aea/2;

apt = den\*dx\*dy/dt \* ones(size(apt)) + ap/2;

end

endfunction

%%% end of unsteady\_coeffs\_LHS %%%

%%% write unsteady RHS in terms of steady coefficients %%%

function unsteady\_RHS

% call global variables needed

global awe aso ano aea ap q den dx dy dt phi tmethod n m

% globals to be written by this function

global qt

if tmethod == 1; % Explicit Euler

for i = 2:n+1

for j = 2:m+1

qt(i,j) = q(i,j) + (den\*dx\*dy/dt).\*phi(i,j) - awe(i,j)\*phi(i-1,j) ...

- aso(i,j)\*phi(i,j-1) - ano(i,j)\*phi(i,j+1) ...

- aea(i,j)\*phi(i+1,j) - ap(i,j)\*phi(i,j);

end

end

elseif tmethod == 2; % Implicit Euler

for i = 2:n+1

for j = 2:m+1

qt(i,j) = q(i,j) + (den\*dx\*dy/dt).\*phi(i,j);

end

end

else; % if tmethod == 3; % Crank-Nicolson

for i = 2:n+1

for j = 2:m+1

qt(i,j) = q(i,j) + (den\*dx\*dy/dt).\*phi(i,j) - awe(i,j)\*phi(i-1,j)/2 ...

- aso(i,j)\*phi(i,j-1)/2 - ano(i,j)\*phi(i,j+1)/2 ...

- aea(i,j)\*phi(i+1,j)/2 - ap(i,j)\*phi(i,j)/2;

end

end

end

endfunction

%%% end of unsteady\_RHS %%%

%%% begin gs

function gs

% globals needed

global apt anot asot aeat awet qt phidir n m epsit resmax errmax nitmax xc yc

global phinew iterstore phi tt

iterstore = 0; % storage variable for iteration count

phinew = phidir; % initialize array for "new" phi values

resTemp = zeros(m, n); % initialize local array to store residuals

% set internal nodes to be zero for initial time

if tt == 0;

phinew(2:end-1, 2:end-1) = 0;

end

nit = 0; % iteration counter

ERRMAX = 1; % initialize scalar variable for max error

while ERRMAX > epsit\*max(max(phinew));

nit = nit + 1;

for j = 2:m+1

for i = 2:n+1

phinew(i, j) = (qt(i,j) - anot(i,j)\*phinew(i,j+1) ...

- asot(i,j)\*phinew(i,j-1) ...

- aeat(i,j)\*phinew(i+1,j) ...

- awet(i,j)\*phinew(i-1,j))/apt(i,j);

end

end

% Periodically show results

if mod(nit, 50) == 0;

fprintf('t=%.3f, GS it=%.0f, errmax=%.4e\n', tt, nit, ERRMAX);

end

for j = 2:m+1

for i = 2:n+1

resTemp(i,j) = qt(i,j) - anot(i,j)\*phinew(i,j+1) ...

- asot(i,j)\*phinew(i,j-1) ...

- aeat(i,j)\*phinew(i+1,j) ...

- awet(i,j)\*phinew(i-1,j) ...

- apt(i,j)\*phinew(i,j);

end

end

resmax(nit) = max(max(resTemp));

phiold = phinew;

phi = phinew;

% update RHS vector Q

unsteady\_RHS;

iterstore = nit; % save number of iterations in global var

% Break out of the while loop if maximum number of iterations is reached

if nit == nitmax;

break

end

errmax(nit) = max(max(abs(phidir - phinew))); % calculate max error

ERRMAX = max(max(abs(phidir - phinew))); % calculate max error

end % end while loop

if nit == nitmax;

fprintf('GS solution did not converge in %.0f iterations.\n', nitmax);

else

fprintf('t = %.4f GS solution converged in %.0f iterations. Errmax = %.4e\n',...

tt, nit, ERRMAX);

end

end

%%% end of gs

% Solve for phi with iterative GS solver

numiters = [];

cputimes = [];

tic

tmethod = 3;

dt = 0.01;

tfinal = 1;

unsteady\_coeffs\_LHS;

for tt = dt:dt:tfinal

gs;

phi = phinew;

numiters(end+1) = iterstore;

if tt == tfinal

local\_quantities;

end

end

cputimes(end+1) = toc;

Part 3:

%%% begin function to output local quantities %%%

function local\_quantities

% globals needed

global phi n m xc yc dx dy

% determine indices and coordinates corresponding to center of domain

nic = ( n + 3 ) / 2;

njc = ( m + 3 ) / 2;

xce = xc(nic);

yce = yc(njc);

% 1. Value of phi at center of domain (x=1/2, y=1/2)

phi\_center = phi(nic,njc);

% 2. Value of dphi/dx at the center of the left-hand boundary (x=0, y=1/2)

dphidx\_centerleft = (phi(2, njc) - phi(1, njc))/dx;

% 3. Value of phi at the center of the right-hand boundary (x=1, y=1/2)

phi\_centerleft = phi(end, njc);

% 4. Value of phi at the center of the bottom boundary

phi\_centerbottom = phi(nic, 1);

% 5. Value of dphi/dy at the center of the top boundary (x=1/2, y=1)

dphidy\_centertop = (phi(nic, end) - phi(nic, end-1))/dy;

% print results to file

fileID = fopen('local\_quantites.txt','w');

fprintf('1:%f\t2:%f\t3:%f\t4:%f\t5::%f\n', phi\_center, dphidx\_centerleft, ...

phi\_centerleft, phi\_centerbottom, dphidy\_centertop);

fclose(fileID);

end

%%% end function to output local quantities %%%

Part 4:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Location | (1) | (2) | (3) | (4) | (5) |
| Steady | 0.141725 | -0.791711 | 0.076696 | 0.444688 | -0.015545 |
| Unsteady, t = 10 | 0.141725 | -0.791711 | 0.076696 | 0.444688 | -0.015545 |

Part 5: Explicit Euler appears to give stable solutions up until a value of dt = 0.01. From heuristic arguments, based on the restriction dt ≤ ρ (dx)^2/(2Г) we would expect dt to be at most 0.0136 for a stable solution. Implicit Euler and Crank-Nicolson are stable at all choices of time-step, but are not necessarily accurate at all choices of time-step.

Part 6: The results of phi at the various locations for Explicit Euler (EE), Implicit Euler (IE) and Crank-Nicolson (CN) are summarized below:



The use of Richardson extrapolation was unsuccessful at each point, because the solution had already converged within about 1e-16 of the exact value by t = 10 s. We would expect a convergence rate *p* of 1.0 for both EE and IE, since they are both first-order accurate, and a convergence rate of 2.0 for the Crank-Nicolson case, which is second-order accurate. Based on the maximum stable time step for Explicit Euler, we would not be able to resolve a stable solution if dt = 0.02 or 0.04 was used. The time-step restriction of Explicit Euler caused substantially longer computational time compared to the other methods.