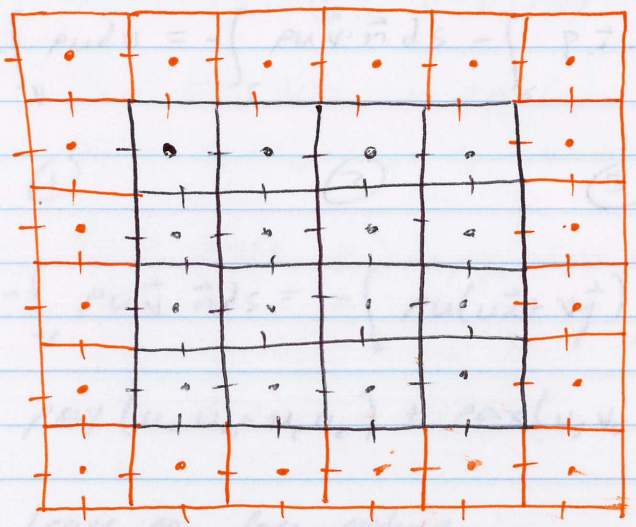
NS Solver — This one works

- 12-1/04
- Do a 2D Soln of NS equations.
- Const ρ, μ, Δx, Δy
- Explicit Euler
- Tested in Lid Driven Cavity, and Slot Jet.



- Staggard Grid
- Use Ghost Cells (orange) all around the Flow Domain,
- U-Cells are Centered at — and have P-cells • on the e, w faces
- V-Cells are Centered at | and have P-cells • on the n,s faces
- Using Ghost Cells is easier Then accounting Just for The flow Domain, and Specifying The Type of free boundary. Its easier to Code, and read and store values at.
- Note ~~Tod~~ That This means we Solve u-cells $i = 3$ to $nx-1$ and $j = 2$ to $ny-2$
  Similarly for v-cells
- ~~the~~ U ij is to the Left of the Corresponding P-Cell
- V ij is below the Corresponding P-Cell.
- P-Cells take all Presedence for Geometry.
  e.g. if a Wall is along The bottom Then V(:,1) and V(:,2) have the wall velocity.

U-Momentum

$$\frac{\partial(\rho\vec{v})}{\partial t} = -\nabla\cdot\rho(\vec{v}\vec{v}) - \nabla p - \nabla\cdot\underline{\underline{\tau}} + \rho\vec{g}$$

$$\frac{\partial(\rho u_\alpha)}{\partial t} = -\frac{\partial}{\partial x_\beta}(\rho u_\alpha u_\beta) - \frac{\partial}{\partial x_\alpha}(p) - \frac{\partial}{\partial x_\beta}(\tau_{\alpha\beta}) + \rho g_\alpha$$

Discretize

$$\frac{\partial}{\partial t}\int_V \rho u\,dV = -\int_S \rho u\vec{v}\cdot\vec{n}\,ds - \int_S p\vec{i}\cdot\vec{n}\,ds - \int_S \vec{\tau}_x\cdot\vec{n}\,ds + \int_V \rho\vec{g}\cdot\vec{i}_V$$

(1) (2) (3) (4) (5)

(2): $$-\int_S \rho u\vec{v}\cdot\vec{n}\,ds = -\int_S \rho u(u\vec{i}+v\vec{j})\cdot\vec{n}\,ds$$

$$\rho\Delta y(u_w u_w - u_e u_e) + \rho\Delta x(u_s v_s - u_n v_n)$$

Leave as face values

(3) $$-\int p\vec{i}\cdot\vec{n}\,ds = \Delta y(p_w - p_e)$$

(4) $$-\int_S \vec{\tau}_x\cdot\vec{n}\,ds = -\int_S (\tau_{xx}\hat{\imath}_x + \tau_{xy}\hat{\imath}_y)\cdot\vec{n}\,ds$$

$$\tau_{xx} = -2\mu\frac{\partial u}{\partial x} \qquad \tau_{xy} = -\mu\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)$$

$$-\int_S \left(-2\mu\frac{\partial u}{\partial x}\vec{i} - \mu\frac{\partial v}{\partial x}\vec{j} - \mu\frac{\partial u}{\partial y}\vec{j}\right)\cdot\vec{n}\,ds$$

$$\int_S \mu\left(\frac{\partial u}{\partial x}\vec{i} + \frac{\partial u}{\partial y}\vec{j}\right)\cdot\vec{n}\,ds + \int \mu\left(\frac{\partial u}{\partial x}\vec{i} + \frac{\partial v}{\partial x}\vec{j}\right)\cdot\vec{n}\,ds$$

(4') (4'')

* Note the form for 4' → general $\nabla\cdot(\Gamma\nabla\phi)$, 4'' → Source

(4') $$\mu\Delta y\left[\left(\frac{\partial u}{\partial x}\right)_e - \left(\frac{\partial u}{\partial x}\right)_w\right] + \mu\Delta x\left[\left(\frac{\partial u}{\partial y}\right)_n - \left(\frac{\partial u}{\partial y}\right)_s\right]$$

(4'') $$\mu\Delta y[\ \text{Same}\ ] + \mu\Delta x\left[\left(\frac{\partial v}{\partial x}\right)_n - \left(\frac{\partial v}{\partial x}\right)_s\right]$$

(5) → $\rho g_x\Delta x\Delta y$

For Central Difference we could write

$$(1) = \sum a_i u_i + \zeta$$

$$a_E = -\rho \Delta y u_e/2 + \mu \Delta y/\Delta x$$

$$a_W = \rho \Delta y u_w/2 + \mu \Delta y/\Delta x$$

$$a_N = -\rho \Delta x v_n/2 + \mu \Delta x \Delta y$$

$$a_S = \rho \Delta x v_s/2 + \mu \Delta x/\Delta y$$

$$a_P = a_E + a_W + a_N + a_S - 4\mu \Delta y/\Delta x - 4\mu \Delta x/\Delta y$$

$$\zeta = \Delta y(P_W - P_e) + \rho g_x \Delta x \Delta y + \frac{\mu \Delta y}{\Delta x}[u_E - 2u_P + u_W] + \mu[v_N - v_{NW} - v_P + v_W]$$

(where we should leave pressure out.)

- For other Algorithms this is a good way, but its way too painful for our explicit Algorithm.
- To be v-mom Just make u's v's, and v's u's, and $e \to n$, $w \to s$, $n \to e$, $s \to w$ $\Delta x \to \Delta y$, etc.
- Note all e w n s are wrt the u-cell. So P_e is P_{ij}
- For centrall Diff, usually $u_e = \frac{u_{i+1,j} + u_{ij}}{\Delta x}$
- To Treat Boundaries its easiest Just To Redefine with face values and Derivatives left out (or known, not approximated.

Write Momentum as:

For Explicit Euler:

$$u_P^{n+1} = \hat{u}^n + \frac{\Delta t}{\rho \Delta x}(P_w^n - P_e^n)$$

$$v_P^{n+1} = \hat{v}^n + \frac{\Delta t}{\rho \Delta x}(P_s^n - P_n^n)$$

~~Continuity~~

- where $\hat{u}^n = u_P^n + \frac{\Delta t}{\rho \Delta x \Delta y}\left[\sum a_u^n u_s^n + S_u^n\right]$
- Note, while we keep the $\sum a_u^n u_s^n$ notation here (which sums over all 5, not just nb) we compute it differently.

  eg we do $\frac{\Delta t}{\rho \Delta x \Delta y}\left[(2) + (4) + (5)\right]$

  where (2) (4) (5) are the face values of u, v, d/dx, which are all computed for every cell.

Continuity: $\nabla \cdot \vec{v} = 0 \rightarrow \int_V \nabla \cdot \vec{v}\, dV = \int_S \vec{v} \cdot \vec{n}\, dS = 0$

$$\Delta y(u_e - u_w) + \Delta x(v_n - v_s) = 0$$

- Sub u_P^{n+1}, v_P^{n+1} into ↗
- Continuity is over P-cell

  So u_e is $u_{i+1,j}$, u_w is u_{ij}, v_n is v_{ij+1}, v_s is v_{ij}

And, P_w in u_e is P_{ij} etc.

So $u_e \rightarrow u_{i+1j}^{n+1} = \hat{u}_{i+1j}^n + \frac{\Delta t}{\rho \Delta x}(P_{ij} - P_{i+1j})$

$u_w \rightarrow u_{ij}^{n+1} = \hat{u}_{ij}^n + \frac{\Delta t}{\rho \Delta x}(P_{i-1j} - P_{ij})$

$v_n \rightarrow v_{ij+1}^{n+1} = \hat{v}_{ij+1}^n + \frac{\Delta t}{\rho \Delta y}(P_{ij} - P_{ij+1})$

$v_s \rightarrow v_{ij}^{n+1} = \hat{v}_{ij}^n + \frac{\Delta t}{\rho \Delta y}(P_{ij-1} - P_{ij})$

Pressure eqn.

$$\frac{\Delta t \Delta y}{\rho \Delta x}\left(P_{i-1,j} - 2P_{ij} + P_{i+1,j}\right) +$$

$$\frac{\Delta t \Delta x}{\rho \Delta y}\left(P_{i,j-1} - 2P_{ij} + P_{i,j+1}\right) =$$

$$\Delta y\left(\hat{u}^{n}_{i+1,j} - \hat{u}^{n}_{ij}\right) + \Delta x\left(\hat{v}^{n}_{i,j+1} - \hat{v}^{n}_{ij}\right)$$

$$\rightarrow \quad A_p P = B_p$$

- where A_p never changes.
- Note diagonals $d_n = d_s = d_e = d_w$ for $\Delta x = \Delta y$
- Note $d_p = (d_n + d_s + d_e + d_w) \cdot (-1)$

- For BC's its easy
- For left wall we Don't Put u_w in since its known
  $\rightarrow$ Kill $P_{i-1,j}$, kill one P_{ij}, Replace $\hat{u}^{n}_{ij}$ with $u_{\text{left wall}}$
- This is how we correct $^{(0)}$ The A_p, B_p matricies.

So, The Algorithm is:

- Initialize The fields
- Compute $\hat{u}$, $\hat{v}$ (only for flow cells but size is all)
- Solve for The Pressure field (only for flow cells, size is flow or all, but Currently flow)
- Get new Velocity.

This is very easy

over →

B.C.'s (a couple words)

- Inlets and walls are easy (just specify face velocities)
- Derivatives are just forward or Backward Diff, but note that w/ Ghost Cells life is easier:
  eg $\left.\frac{du}{dy}\right|_n$ is just $2 \times \left.\frac{du}{dy}\right|_n$ evaluated w/ the Ghost cell (The $\times 2$ is cause $dy = \Delta y/2$)
- Likewise for face vel. $u_n = u_{ghost}$ where the value of the Ghost cell (0) or other extends to the face.
- For Outlets life is harder.
  As for inlets and walls we want to be able to specify the face values of u, v and Derivatives (which are ito u, v)
  - Satisfy Continuity at time zero (and all others) by making the Ghost velocities such that mass out = mass in.

I've Done this by making Tangent Vel zero and $\perp$ velocity = Upwind velocity + V_corr where V_corr is just an average correction to add to all cells next too outlet.

eg if 4 cells in then mass in $\propto V_{in} \times 4$
mass out $\propto V_{out} \times (nx-2)$ or (ΣV_{out}) for nonuniform Vel

$$V_{corr} = \frac{(\text{mass in} - \text{mass out})}{(nx-2)}$$

$$V_{out} = V_{j-1} + V_{corr}$$

- Note This is just an easy extrapolation.
  in General extrapolate to outlet then correct velocity
- This is not the same as saying $\frac{dV}{dx} = 0$
- We could make an nonzero Tangent Vel too.

Note too: This code works with and w/o forcing one $P=0$

eg $A_P(ij) = 1$, $B_P(i) = 0$

But be sure to shift P around some small value (eg $P = P - P_{avg}$) at each iteration or P will Grow and washout ∇P. (its easy to watch this happen)

I'm not sure why I can get away with not setting $A_P(\text{middle}) = 1$, $B_P(i) = 0$ (or other)

Jeremy says maybe its the walls.

See Attached code → next pages.

Next Steps

- Allow for energy Eqn (e.g. Temperature)
- Allow for variable Density (EOS)
- Allow for variable ΔY, ΔX
- Allow for Turbulence
-

```
% David O. Lignell
% 12/27/03
% Constant density, 2-D, laminar flow solver
% Constant viscosity, constant x and y grid
% Time accurate, explicit solver.  (or at least time advance to steady state)
% Explicit Euler is used
% Pressure solve, not pressure correction
% Allow for wall, inlet, outlet BC
% See make grid for setup

clc;
clear;

make_grid;

Re = 500;
mu = 1.0e-3;
nu = 1.0e-6;
rho = mu/nu;
v_in = Re*nu/W;
u_in = 0.0;
mass_in = (v_in^2+u_in^2)^0.5 * 4;       % grid specific
gx = 0.0;
gy = 0.0;

%Simulation control                      (stolen from Jeremy's soln)
ttime = 0.1*W^2/nu;                      %total time iterations
CFL = 1.0;                               %CFL constant
dtc = CFL*0.5*dx/v_in;                   %time step, convective
dtd = CFL*0.5*dx^2/nu;                   %time step, diffusive
dt = min(dtc,dtd)/1.0;                   %choose a dt
rtol = 1.0e-6;

ttime = ttime / 2;
% Create and initialize field variables

u_vel     = zeros(nx,ny);
v_vel     = u_vel;
pressure  = u_vel;
u_cc      = zeros(nx,ny);
v_cc      = u_cc;
v_vel(nx/2-1:nx/2+2,1:2) = v_in;

u_hat = zeros(nx,ny);
v_hat = zeros(nx,ny);

Ap_init = 1;                                % Flag to calc Ap in get_Ap once only
time = 0.0;
titer = 0;

while time < ttime
    time = time + dt;
    titer = titer + 1;

    fprintf('-------------------------------------\n')
    fprintf('       Time = ')
    fprintf('%6.2f\n',time)
    fprintf('Iteration # = ')
    fprintf('%6.0i\n',titer)

    get_uv_hat2;
    get_Ap;                                 % Get pressure equation Ap * P = Bp m
atricies
    P = Ap\Bp;
    P = reshape(P,nx-2,ny-2);               % P size here is nx-2, ny-2
    %P = P - sum(sum(P))/((nx-2)*(ny-2));   % Uncomment if Ap, Bp don't force a r
eference P
                                            % Code works with uncommented and Ap,
 Bp general, not sure why (walls?)
    % get u and v velocities at n+1

    for i=3:nx-1
        u_vel(i,2:ny-1) = u_hat(i,2:ny-1) + dt/rho/dx*( P(i-1-1,1:ny-2) - P(i-1,1:ny
```

```
-2) );
    end
    for j=3:ny-1
        v_vel(2:nx-1,j) = v_hat(2:nx-1,j) + dt/rho/dy*( P(1:nx-2,j-1-1) - P(1:nx-2,j
-1) );
    end

    get_max_divergence;                         % Test continuity and print to screen

    for i=2:nx-1
        u_cc(i,2:ny-1) = 0.5*( u_vel(i,2:ny-1)+u_vel(i+1,2:ny-1) );
    end
    for j=2:ny-1
        v_cc(2:nx-1,j) = 0.5*( v_vel(2:nx-1,j)+v_vel(2:nx-1,j+1) );
    end
    KE(titer,1) = sum(sum( (u_cc.*u_cc + v_cc.*v_cc).^0.5));   % total KE as converg
ence indicator
    fprintf('KE = ');
    fprintf('%6.2e\n', KE(titer));

    %plot the stuff:            ( this part stolen from Jeremy's soln)
    if (mod(titer,30) == 0)
        close all
        subplot(221), quiver(x',y',u_cc(2:nx-1,2:ny-1),v_cc(2:nx-1,2:ny-1));title('v
el vectors')
        axis image
        subplot(222), plot(KE); title('Kinetic Energy');
        subplot(223), streamslice(u_cc',v_cc');title('stream lines')
        axis image
        subplot(224), contourf(v_cc',10);title('u-vel contour');
        axis fill
        pause(0.5)

    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

% get_Ap.m

% David O. Lignell
% Routine returns the Coefficient matrix for the pressure equation
%    As well as the rhs vector Bp
% e.g. Ap * P = Bp,  or P = Ap/Bp
% Ap should not change, so we only have to do this once
% Note for diag -1, the last element of the diagonal is trimmed, while for
% diag 1 the first element is trimmed (if the diagonals are the size of the mat)
% Note we fix a value of P=0 at about the middle cell.  This can be commented out
%   but be sure to uncomment the P shift for the soln in ns.m (e.g. the shift around
 the average).

nxv = nx-2;                                 % number of p cells x-dir
nyv = ny-2;
nvars = nxv*nyv;
Bp = zeros(nx,ny);   % not nxv, nyv

if Ap_init == 1

    ds = repmat(dt*dx/rho/dy, nvars,1);              % south neighbor diagonal
    dn = ds;
    de = repmat(dt*dy/rho/dx, nvars,1);
    dw = de;
    dp = repmat(-2*dt/rho*(dx/dy+dy/dx),nvars,1);

    de([1:nxv:nvars])  = 0.0;     % correct right wall
    dw([nxv:nxv:nvars]) = 0.0;    % correct left wall
    % correct top and bottom
    dp([1:nxv nvars-nxv+1:nvars]) = dp([1:nxv nvars-nxv+1:nvars]) + dt/rho*dx/dy;
    % correct left wall
    dp(1:nxv:nvars) = dp(1:nxv:nvars) + dt/rho*dy/dx;
    % correct right wall
    dp(nxv:nxv:nvars) = dp(nxv:nxv:nvars) + dt/rho*dy/dx;
```

```
    Ap = spdiags([ds dw dp de dn], [-nxv -1 0 1 nxv], nvars, nvars);

    Ap(floor(nvars/2+nxv/2),:) = 0;                    % specify pressure at one poi
nt
    Ap(floor(nvars/2+nxv/2),floor(nvars/2+nxv/2)) = 1.0;

    % get the right hand side here
    Ap_init = 0;
end

for i=2:nx-1
    for j=2:ny-1
        Bp(i,j) = dy*(u_hat(i+1,j)-u_hat(i,j)) + dx*(v_hat(i,j+1)-v_hat(i,j));
    end
end
Bp(:,2)    = Bp(:,2) - dx*v_vel(:,2);              % fix bottom
Bp(:,ny-1) = Bp(:,ny-1) + dx*v_vel(:,ny);          % fix top
Bp(2,:)    = Bp(2,:) - dy*u_vel(2,:);              % fix left
Bp(nx-1,:) = Bp(nx-1,:) + dy*u_vel(nx,:);          % fix right

Bp = reshape(Bp(2:nx-1,2:ny-1),nvars,1);
Bp(floor(nvars/2+nxv/2),1) = 0.0;                   % specifying pressure at one p
oint


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

% get_max_divergence.m

% David O. Lignell
% Compute maximum cell divergence to test satisfaction of the continuity equation

for i=2:nx-1
    for j=2:ny-1
        div(i,j) = (u_vel(i+1,j)-u_vel(i,j))*dy + dx*(v_vel(i,j+1)-v_vel(i,j));
    end
end

maxdiv = max(max(abs(div)));
[imd,jmd] = find(abs(div) == maxdiv);
maxv   = max([max(abs(v_vel)) max(abs(u_vel))]);
fprintf('maxdiv = ')
fprintf('%6.2e\n', maxdiv);
fprintf('maxu = ')
fprintf('%6.2e\n', maxv);
%imd
%jmd

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

% get_uv_hat2.m

% David O. Lignell 12/29/03
% u hat is basically the rhs (all but unsteady term) of the mom eqn, without pressur
e,
%   and with the old velocity
% e.g. u(n+1) = uhat(n) + dt/rho/dx*(Pw-Pe)
% So, the algorithm gets uhat, then plugs the above into continuity and solves P,
%    then solves the above for u(n+1)
%
% This technique is very simple and straight forward.
% Every cell in FLOW domain has the relevant face velocities and derivatives compute
d
% For boundaries, just make sure the face value of vel and dv/dx are correct
% The outlet is treated by assuming u is 0, and v is the adjacent value (upstream) +
 a velocity
%    Correction to force mass conseration.  Note that this is done immediately at ti
me = 0.
% Alternately, we could extrapolate the velocity some way, then apply a correction.
% Note that this is not the same as assuming the gradient of velocity is zero
```

```
v_vel(2:nx-1,ny) = v_vel(2:nx-1,ny-1);               % Set the v-vel at the outlet
u_vel(3:nx-1,ny) = 0.0;
mass_out = sum(v_vel(2:nx-1,ny));
v_corr = (mass_in-mass_out)/(nx-2);
v_vel(2:nx-1,ny) = v_vel(2:nx-1,ny)+v_corr;
mass_out = sum(v_vel(2:nx-1,ny));
if abs((mass_in-mass_out)/mass_in) > 1.0e-6;
    fprintf('error in outlet velocity');
end

for i=3:nx-1                                  % For u cells
    for j=2:ny-1

        ue    = 0.5*(u_vel(i+1,j)+u_vel(i,j));
        uw    = 0.5*(u_vel(i-1,j)+u_vel(i,j));
        un    = 0.5*(u_vel(i,j+1)+u_vel(i,j));
        us    = 0.5*(u_vel(i,j-1)+u_vel(i,j));
        vn    = 0.5*(v_vel(i,j+1)+v_vel(i-1,j+1));
        vs    = 0.5*(v_vel(i-1,j)+v_vel(i,j));
        dudxe = (u_vel(i+1,j)-u_vel(i,j))/dx;
        dudxw = (u_vel(i,j)-u_vel(i-1,j))/dx;
        dudyn = (u_vel(i,j+1)-u_vel(i,j))/dy;
        dudys = (u_vel(i,j)-u_vel(i,j-1))/dy;
        dvdxn = (v_vel(i,j+1)-v_vel(i-1,j+1))/dx;
        dvdxs = (v_vel(i,j)-v_vel(i-1,j))/dx;

        if j == ny-1
            un    = u_vel(i,j+1);
            dudyn = dudyn * 2.0;
        elseif j == 2
            us    = u_vel(i,j-1);
            dudys = dudys*2.0;
        end


        u_hat(i,j) = ( rho*dy*(uw*uw-ue*ue) + rho*dx*(us*vs-un*vn) + ...
                       2.0*mu*dy*(dudxe-dudxw) + mu*dx*(dudyn-dudys) + ...
                       mu*dx*(dvdxn-dvdxs) + ...
                       rho*gx*dx*dy ) * dt/(rho*dx*dy) + u_vel(i,j);

    end
end

for i=2:nx-1                                      % v cells
    for j=3:ny-1

        vn    = 0.5*(v_vel(i,j+1)+v_vel(i,j));
        vs    = 0.5*(v_vel(i,j-1)+v_vel(i,j));
        ve    = 0.5*(v_vel(i+1,j)+v_vel(i,j));
        vw    = 0.5*(v_vel(i-1,j)+v_vel(i,j));
        ue    = 0.5*(u_vel(i+1,j)+u_vel(i+1,j-1));
        uw    = 0.5*(u_vel(i,j)+u_vel(i,j-1));
        dvdyn = (v_vel(i,j+1)-v_vel(i,j))/dy;            % is zero at j=ny-1
        dvdys = (v_vel(i,j)-v_vel(i,j-1))/dy;
        dvdxe = (v_vel(i+1,j)-v_vel(i,j))/dx;
        dvdxw = (v_vel(i,j)-v_vel(i-1,j))/dx;
        dudye = (u_vel(i+1,j)-u_vel(i+1,j-1))/dy;
        dudyw = (u_vel(i,j)-u_vel(i,j-1))/dy;

        if i == 2
            vw    = v_vel(i-1,j);
            dvdxw = dvdxw * 2.0;
        elseif i == nx-1
            ve    = v_vel(i+1,j);
            dvdxe = dvdxe * 2.0;
        end

        v_hat(i,j) = v_vel(i,j) + dt/(rho*dx*dy) * ...
            ( rho*dx*(vs*vs-vn*vn) + rho*dy*(vw*uw-ve*ue) + ...
            2.0*mu*dx*(dvdyn-dvdys) + mu*dy*(dvdxe-dvdxw) + ...
            mu*dy*(dudye-dudyw) + ...
            rho*gy*dx*dy );
```

```
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

% make_grid.m

% David O. Lignell
% 12/27/03
% make_grid.m
% called by ns.m
% specify grid dimensions, cell types
% Code assumes constant delta_x, and delta_y
% Obviously there should be a whole number of cells in each dir so make L/ny whole,
etc.
% This grid is for the slot jet problem of phils cfd class
% Inlet area is 0.2*W, and is 4 cells across
%
%      |   outlet BC   |
%      |               |
%      |               |
%      |               |
%      |   Fluid       |
%      |               | Walls
%      |               |
%      |      ^        |                ^ y , j
%      |      |        |                |
%      |____|in|_____|         x,i    |---->        Lower right corner = (0,0)


L   = 4.0;                         % y-dir grid width
W   = 1.0;                         % x-dir grid length

% Set base flow domain

nx = 80;
ny = nx*L/W;
dx = W/nx;
dy = L/ny;

[x y]  = meshgrid(dx:dx:W, dy:dy:L);
% Add ghost cells (for simple grids this is the boundary cells)

nx = nx + 2;
ny = ny + 2;

% Define four valid cell types (flow, wall, inlet, outlet)
% Flow is 0, inlet is 1, outlet is 2, wall is 3
% Assume flow only in interior cells, inlet and outlet only on boundary (ghost cells
)
% and wall cells anywhere

cell_types = zeros(nx,ny);  % not used in this code

cell_types(1,:)        = 3;
cell_types(end,:)      = 3;
cell_types(2:nx-1,end) = 2;

win = W*0.2;
cell_types(nx/2-1:nx/2+2,1) = 1;
```