

# Problem Set 7 Initial Value Problems

## 10.34 Fall 2005, KJ Beers

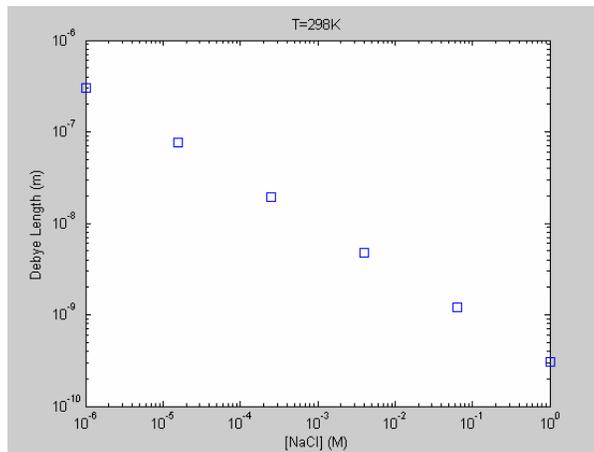
*Ben Wang, Mark Styczynski*  
*November 7, 2005*

### Problem 1: 6.B.3

This is basically just a 1D BVP which we can solve using the method of finite differences. We will come up with an equation corresponding to each grid point and solve the entire system of equations with FSOLVE, since we have a nonlinear system of equations with the exponential terms. We are also asked to plot a few ancillary plots, introduce comparisons, and make some comments about the physicalness of the Poisson-Boltzmann equation. We begin by looking at the how the Debye length,  $\lambda$ , varies with  $[NaCl]$ :

$$\lambda = \left[ \frac{\epsilon_o \epsilon_r k_B T}{q_e^2 N_{av} (2[NaCl])} \right]^{1/2} \quad (1)$$

Plotting this:



We are asked to answer whether or not it makes sense to have a Debye length of 0.1 nm, and we answer that it is not. The Debye length is the characteristic length scale of this problem and it doesn't make sense to have a continuum model to describe the phenomenon when the length is on the order of the charged species themselves.

Next we are asked to solve this boundary value problem (BVP) numerically. This is relatively straightforward after all the non-dimensionalization and derivation that professor Beers goes through. He arrives at:

$$-\frac{d^2\phi}{d\xi^2} = \frac{1}{2} \left( e^{-\phi(\xi)} - e^{\phi(\xi)} \right) \quad (2)$$

with the following Dirichlet type boundary conditions:

$$\phi(0) = \frac{\Phi_0 q_e}{k_B T} = K \quad (3)$$

$$\phi(\infty) = 0 \quad (4)$$

To derive the finite difference equations required for this equation we first look at how we are to space our grid points. Why do we even care about this? Well if we look at equation (4) which states that our boundary condition is defined at infinity, we quickly realize that it is difficult to come up with a numerical solution that includes an infinite number of grid points. Instead we come up with a numerical approximation to infinity. Because we know we have a characteristic length associated with this problem,  $\lambda$ , we can assume that most of what happens occurs within one characteristic length. Therefore if we set infinity at say  $10\lambda$ , this will likely be sufficiently far enough from what occurs to represent infinity. Now we have a range over which we can set up a grid.

However we have just said that, most likely, all of the fun occurs within the span of one  $\lambda$ , we want a dense grid over this region where we can observe how the solution varies here. Similarly we might want to put a sparse grid in the region for length  $> \lambda$ , to save on computation time. And this is what we do here in this solution. However if you chose not to include this different grid spacing, it is fine.

As long as you write the correct finite difference equations for your differential equation you should be all good. First we look at the Laplacian term, and we derive the finite difference terms generally for any grid-spacing:

$$\frac{d\phi_i}{d\xi} = \frac{\phi_{i+1/2} - \phi_{i-1/2}}{\xi_{i+1/2} - \xi_{i-1/2}} \quad (5)$$

$$\frac{d^2\phi_i}{d\xi^2} = \frac{\left. \frac{d\phi}{d\xi} \right|_{i+1/2} - \left. \frac{d\phi}{d\xi} \right|_{i-1/2}}{\xi_{i+1/2} - \xi_{i-1/2}} = \frac{\frac{\phi_{i+1} - \phi_i}{\xi_{i+1} - \xi_i} - \frac{\phi_i - \phi_{i-1}}{\xi_i - \xi_{i-1}}}{2} \quad (6)$$

So combining this with the exponential terms we arrive at:

$$0 = \frac{\frac{\phi_{i+1} - \phi_i}{\xi_{i+1} - \xi_i} - \frac{\phi_i - \phi_{i-1}}{\xi_i - \xi_{i-1}}}{\frac{\xi_{i+1} - \xi_{i-1}}{2}} + \frac{1}{2} (e^{-\phi_i} - e^{\phi_i}) \quad (7)$$

This equation is valid for all points  $i$ , not at  $i = 1$ ,  $i = N+1$  (for example  $10\lambda$ ). We now treat each boundary condition individually.

For Dirichlet type boundary conditions, we just replace the value of  $\phi$  at the 'imaginary' grid point, with the boundary value. For our problem, we look at our first equation and replace the value of  $\phi_0$  with  $K$  (as in equation (3)). We also know that  $\xi_0 = 0$  at the boundary.

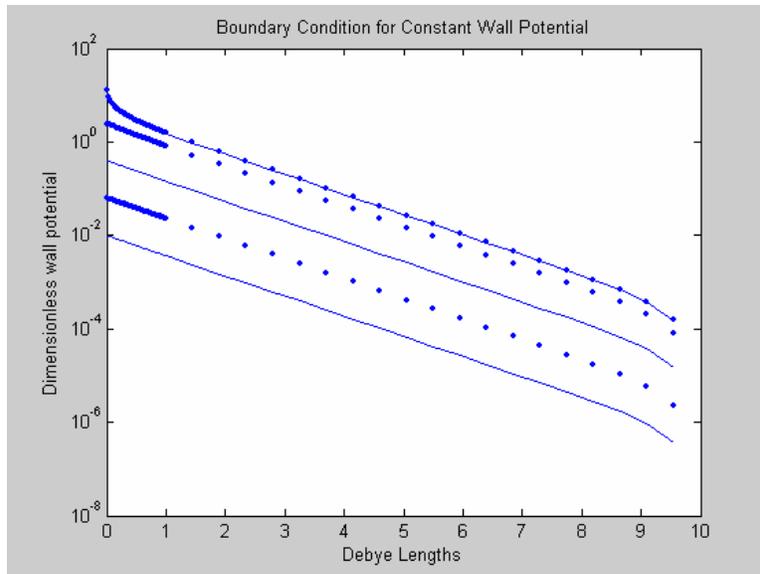
$$0 = \frac{\frac{\phi_2 - \phi_1}{\xi_2 - \xi_1} - \frac{\phi_1 - K}{\xi_1 - 0}}{\frac{\xi_2 - 0}{2}} + \frac{1}{2} (e^{-\phi_1} - e^{\phi_1}) \quad (8)$$

For our boundary condition at infinity, we know that the value of  $\phi = 0$  at grid point  $N+1$ . We write the following as:

$$0 = \frac{\frac{\phi_{N+1} - \phi_N}{\xi_{N+1} - \xi_N} - \frac{\phi_N - \phi_{N-1}}{\xi_N - \xi_{N-1}}}{\frac{\xi_{N+1} - \xi_{N-1}}{2}} + \frac{1}{2} (e^{-\phi_N} - e^{\phi_N}) \quad (9)$$

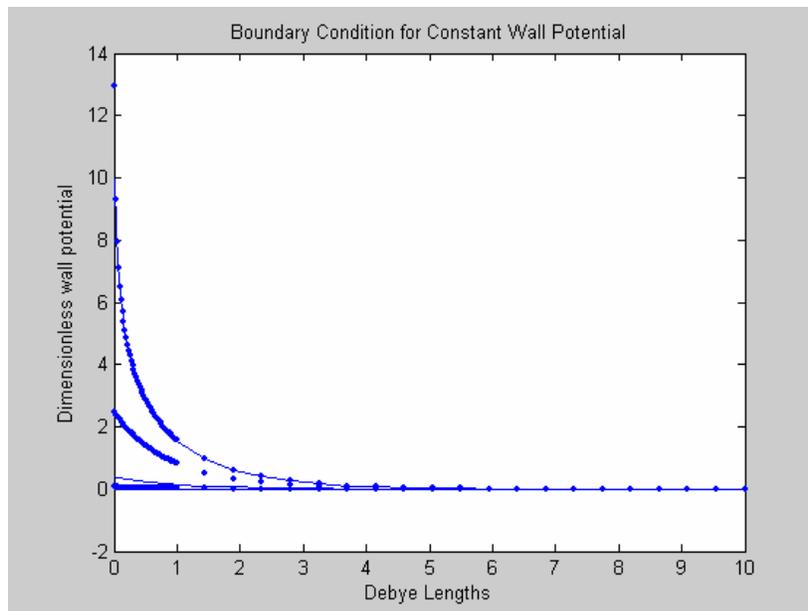
We treat  $\xi_{N+1} = \xi_N + \Delta\xi$ . But there are other ways you can treat this.

We get a plot like the following:



A few notes about this graph: we plot it using semilogy because it is easier to see the full range of different orders of magnitudes of wall potential. We can also see that for values of  $\frac{\Phi_0 q_e}{k_B T} = K \ll 1$ ,  $\phi(\xi)$  shows an exponential decay, which shows up as a linear plot in semilogy. As we increase  $K$  we observe a deviation from that purely exponential behavior. At high values of  $K$ , the potential curves tend to collapse onto one another, as the behavior is dominated by the exponential term. At high debye lengths we observe an artifact as the curves tend to diverge towards zero. This is because at  $10\lambda$ , we set the value of  $\phi = 0$ .

If we did not plot this in semilogy, your curves would look something like:



```

% benwang_P6B3.m
% Ben Wang
% HW#7 Problem #1
% due 11/7/05 9 am

% This routine will use the finite difference approximation to generate a
% system of nonlinear equations, solving the poisson-boltzmann equation

% ===== main routine benwang_P6B3.m
function iflag_main = benwang_P6B3();

iflag_main = 0;

%PDL> clear graphs, screen etc. general initialization
clear all; close all; clc;

%PDL> specify parameters
param.qe = 1.602e-19;           % [C]      charge of electron
param.Nav = 6.022e23;          % []      Avogadro's number
param.eps_0 = 8.854e-12;       % [C^2/J-m]  permittivity of free space
param.eps_r = 78;              % []      dielectric constant for water
param.kB = 1.38e-23;           % [J/K]    Boltzmann's constant
param.T = 298;                 % [K]      room temp
param.totaldebye = 10;         %          total debye lengths of simulation

%PDL> specify grid parameters

grid.N1 = 50;                  % number of grid points close to the wall
grid.N2 = 10;                  % number of grid points far away from wall

%PDL> specify NaCl concentration

num_curves = 10;
conc = logspace(-6,0,num_curves); % [M]
NaCl = conc*1000;              % [mol/m^3]

%PDL> specify wall potential

big_phi = logspace(-2,2,num_curves)/param.qe*param.kB*param.T;
matrix = zeros(grid.N1+grid.N2,num_curves);

% plot debye length vs. [NaCl]
figure;
for i = 1:length(NaCl)
    param.debye = sqrt(param.eps_0*param.eps_r*param.kB*param.T/...
        (param.qe^2*param.Nav*(2*NaCl(i))));
    loglog(conc(i),param.debye,'s');
    hold on;
end

hold off;
xlabel('[NaCl] (M)');
ylabel('Debye Length (m)');

```

**%PDL> specify discretization of grid**

```
grid.domain = linspace(0,1,grid.N1+1);  
grid.domain = grid.domain(2:end);  
a = linspace(1,param.totaldebye, grid.N2+1);  
grid.domain = [grid.domain a(2:end)];
```

**%PDL> call fsolve to solve system for nonlinear equations**

```
for j = 1:length(big_phi)  
    param.wallpotent = big_phi(j);  
    guess = ones(grid.N1+grid.N2, 1);  
    [phi,f] = fsolve(@calc_func_P6B3, guess, [], param, grid);  
    matrix(:,j) = phi;  
end
```

**% plot curves**

```
figure;  
for m = 1:num_curves  
    x = grid.domain;  
    y = matrix(:,m);  
    semilogy(x,y,'-');  
    hold on;  
end  
xlabel('Debye Lengths');  
ylabel('Dimensionless wall potential');  
title('Boundary Condition for Constant Wall Potential');
```

```
figure;  
for m = 1:num_curves  
    x = grid.domain;  
    y = matrix(:,m);  
    plot(x,y,'-');  
    hold on;  
end  
xlabel('Debye Lengths');  
ylabel('Dimensionless wall potential');  
title('Boundary Condition for Constant Wall Potential');
```

**return;**

**%===== subroutine calc\_func\_P6B3.m**  
**function [f] = calc\_func\_P6B3(phi,param,grid)**

```
N = length(phi);  
f = zeros(N, 1);  
zeta = grid.domain;  
phi;
```

**% first calculate interior points**

```
for k = 2:N-1  
    delta_zeta_mid = 0.5*(zeta(k+1)-zeta(k-1));
```

```

A_lo = ((zeta(k)-zeta(k-1))*delta_zeta_mid)^-1;
A_hi = ((zeta(k+1)-zeta(k))*delta_zeta_mid)^-1;
A_mid_1 = -1/((zeta(k+1)-zeta(k))*delta_zeta_mid);
A_mid_2 = -1/((zeta(k)-zeta(k-1))*delta_zeta_mid);
f(k) = A_lo*phi(k-1)+(A_mid_1+A_mid_2)*phi(k)+A_hi*phi(k+1)+0.5*(exp(-phi(k))...
-exp(phi(k)));
end

% boundary condition close to the wall. since this is a fixed number, we
% can just specify the value of "phi(0)"

phi_wall = param.wallpotent*param.qe/param.kB/param.T;
delta_zeta_mid = 0.5*(zeta(2)-0);
A_lo = 1/((zeta(1) - 0)*delta_zeta_mid);
A_hi = 1/((zeta(2)-zeta(1))*delta_zeta_mid);
A_mid_1 = -1/((zeta(2)-zeta(1))*delta_zeta_mid);
A_mid_2 = -1/((zeta(1)-0)*delta_zeta_mid);
f(1) = A_lo*phi_wall+(A_mid_1+A_mid_2)*phi(1)+A_hi*phi(2)+0.5*(exp(-phi(1))-
exp(phi(1)));

% boundary condition close at infinity. this is also a fixed number = 0.
% This is significantly different however, since at very far away, the
% derivative also approaches 0. We will try them both.

phi_equil = 0;
delta_zeta_mid = 0.5*(phi_equil+zeta(N-1));
A_lo = 1/((zeta(N-1) - zeta(N))*delta_zeta_mid);
A_hi = 1/((0-zeta(N))*delta_zeta_mid);
A_mid_1 = -1/((0-zeta(N))*delta_zeta_mid);
A_mid_2 = -1/((zeta(N)-zeta(N-1))*delta_zeta_mid);
f(N) = A_lo*phi(N-1)+(A_mid_1+A_mid_2)*phi(N)+A_hi*phi_equil+0.5*(exp(-phi(N))-
exp(phi(N)));
return;

```

## Problem 2: P6B4

This problem is very similar to the problem P6B4, with the exception that we now change the boundary condition at the wall to one of fixed charge density. The way charge density is related to the wall potential is by the following equation:

$$\sigma_0 = -\epsilon_0 \epsilon_r \left. \frac{d\Phi}{dz} \right|_{z=0} \quad (10)$$

If we non-dimensionalize we get our boundary condition that states:

$$-\frac{k_B T \lambda \sigma_0}{q_e \epsilon_0 \epsilon_r} = -K_2 = \left. \frac{d\phi}{d\xi} \right|_{\xi=0} \quad (11)$$

So it looks like we just have a fixed Neumann condition, in which we can vary  $\sigma_0$ .

With this in mind we now have to go back to our first equation and find a new value for  $\phi_0$

We look in Beers, page 376 on the treatment of Neumann boundary conditions. Just to reiterate we use this polynomial interpolation to approximate the derivative to get better error performance. Assuming that we have a locally uniform grid (we bypass the tedious algebra associated with the Lagrange polynomials) and we see that we can use the equation:

$$\left. \frac{d\phi}{d\xi} \right|_{\xi=0} = -K_2 = \frac{-3\phi_0 + 4\phi_1 - \phi_2}{2(\Delta\xi)} \quad (12)$$

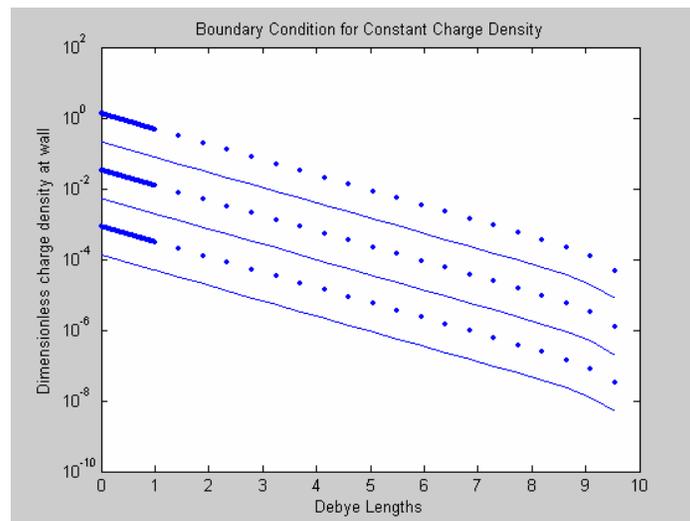
We can solve for  $\phi_0$  and arrive at:

$$\phi_0 = \frac{2K_2(\Delta\xi) + 4\phi_1 - \phi_2}{3} \quad (13)$$

We can now plug this into equation (7) and arrive at:

$$0 = \frac{\phi_2 - \phi_1}{\xi_2 - \xi_1} - \frac{\phi_1 - \left( \frac{2K_2(\Delta\xi) + 4\phi_1 - \phi_2}{3} \right)}{\frac{\xi_2}{2}} + \frac{1}{2}(e^{-\phi_1} - e^{\phi_1}) \quad (14)$$

Now everything else remains the same from the code in P6B3. We run it to get the following plot:



You notice that you get very similar results to problem 6B3.

```
% benwang_P6B4.m
% Ben Wang
% HW#7 Problem #1
% due 11/7/05 9 am

% This routine will use the finite difference approximation to generate a
% system of nonlinear equations, solving the poisson-boltzmann equation

% ===== main routine benwang_P6B4.m
function iflag_main = benwang_P6B4();

iflag_main = 0;

%PDL> clear graphs, screen etc. general initialization
clear all; close all; clc;

%PDL> specify parameters
param.qe = 1.602e-19;      % [C]      charge of electron
param.Nav = 6.022e23;     % []      Avogadro's number
param.eps_0 = 8.854e-12;  % [C^2/J-m]  permittivity of free space
param.eps_r = 78;        % []      dielectric constant for water
param.kB = 1.38e-23;     % [J/K]   Boltzmann's constant
param.T = 298;          % [K]     room temp
param.totaldebye = 10;   %         total debye lengths of simulation

%PDL> specify grid parameters

grid.N1 = 50;            % number of grid points close to the wall
grid.N2 = 20;           % number of grid points far away from wall

%PDL> specify NaCl concentration
num_curves = 6;
conc = logspace(-6,0,num_curves);      % [M]
NaCl = conc*1000;                       % [mol/m^3]

%PDL> wall charge density

sigma_0 = logspace(-2,2,num_curves);
matrix = zeros(grid.N1+grid.N2,num_curves);

%PDL> specify discretization of grid

grid.domain = linspace(0,1,grid.N1+1);
grid.domain = grid.domain(2:end);
a = linspace(1,param.totaldebye, grid.N2+1);
grid.domain = [grid.domain a(2:end)]';

for j = 1:length(sigma_0)
    param.sigma_wall = sigma_0(j);
    guess = ones(grid.N1+grid.N2,1);
end
```

```

        [phi,f] = fsolve(@calc_func_P6B4, guess, [], param, grid);
        matrix(:,j) = phi;
end

% plot curves
funky = ['-','-','-','-','-'];
figure;
for m = 1:num_curves
    x = grid.domain;
    y = matrix(:,m);
    semilogy(x,y,funky(m));
    hold on;
end
xlabel('Debye Lengths');
ylabel('Dimensionless charge density at wall');
title('Boundary Condition for Constant Charge Density');

return

%===== subroutine calc_func_P6B4.m
function [f] = calc_func_P6B4(phi,param,grid)

N = length(phi);
f = zeros(N,1);
zeta = grid.domain;
phi;

% first calculate interior points
for k = 2:N-1
    delta_zeta_mid = 0.5*(zeta(k+1)-zeta(k-1));
    A_lo = ((zeta(k)-zeta(k-1))*delta_zeta_mid)^-1;
    A_hi = ((zeta(k+1)-zeta(k))*delta_zeta_mid)^-1;
    A_mid_1 = -1/((zeta(k+1)-zeta(k))*delta_zeta_mid);
    A_mid_2 = -1/((zeta(k)-zeta(k-1))*delta_zeta_mid);

    f(k) = A_lo*phi(k-1)+(A_mid_1+A_mid_2)*phi(k)+A_hi*phi(k+1)+0.5*(exp(-phi(k))...
        -exp(phi(k)));
end

% boundary condition close to the wall. since this is a fixed number, we
% can just specify the value of "phi(0)"

constant = param.sigma_wall*2/3*(zeta(2)-zeta(1));
delta_zeta_mid = 0.5*(zeta(2)-0);
A_lo = 1/((zeta(1) - 0)*delta_zeta_mid);
A_hi = 1/((zeta(2)-zeta(1))*delta_zeta_mid);
A_mid_1 = -1/((zeta(2)-zeta(1))*delta_zeta_mid);
A_mid_2 = -1/((zeta(1)-0)*delta_zeta_mid);

f(1) = A_lo*constant+(A_mid_1+A_mid_2+4/3)*phi(1)+(A_hi-1/3)*phi(2)+0.5*(exp(-phi(1))-
exp(phi(1)));

% boundary condition close at infinity. this is also a fixed number = 0.
% This is significantly different however, since at very far away, the

```

% derivative also approaches 0. We will try them both.

```
phi_equil = 0;
```

```
delta_zeta_mid = 0.5*(phi_equil+zeta(N-1));
```

```
A_lo = 1/((zeta(N-1) - zeta(N))*delta_zeta_mid);
```

```
A_hi = 1/((0-zeta(N))*delta_zeta_mid);
```

```
A_mid_1 = -1/((0-zeta(N))*delta_zeta_mid);
```

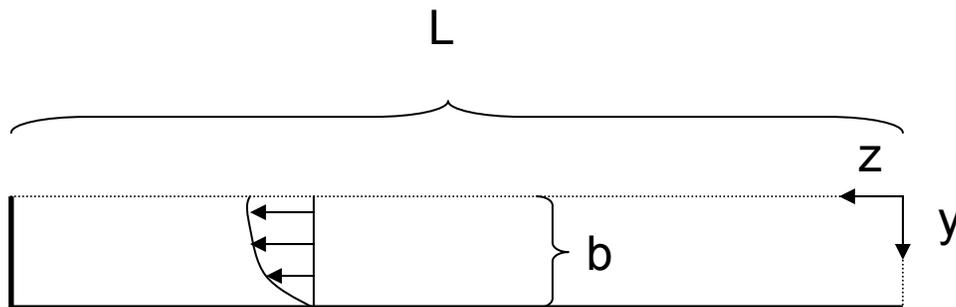
```
A_mid_2 = -1/((zeta(N)-zeta(N-1))*delta_zeta_mid);
```

```
f(N) = A_lo*phi(N-1)+(A_mid_1+A_mid_2)*phi(N)+A_hi*phi_equil+0.5*(exp(-phi(N))-  
exp(phi(N)));
```

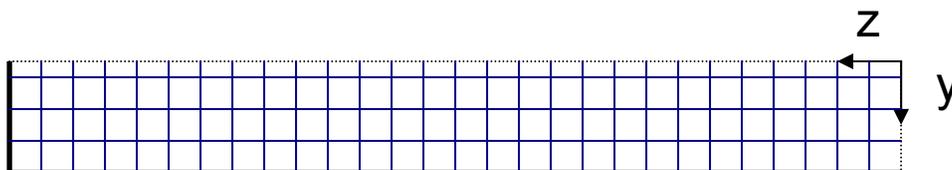
```
return;
```

### Problem 3: P6C5

This problem involves using finite differences to solve a 2D convection-diffusion-reaction problem involving three different species. We begin by looking at the geometry of the problem (with the axis rotated 10 degrees):



Since we have convection in our system, we need to solve for the flow profile. We first discretize our system, schematically shown in the figure below:



Therefore if we discretize the y-axis into  $N_y$  grid points, we should be able to calculate a local velocity at each grid point in the y-direction. We make a few assumptions: that there is no convective momentum transfer, that our flow is fully developed and we neglect entrance and exit effects, as well as edge effects orthogonal to the z-axis and y-axis. Furthermore we assume that we have laminar flow and that there is no change in the thickness of the film so that the

only velocity is along the z-axis. Lastly we ignore any potential pressure gradient such that our flow is purely gravity driven. Thus we have a very simple flow problem which can be solved analytically.

### *Velocity*

Beginning with Navier-Stokes in 2D, which we can take from Deen p.229, we can eliminate the terms to have:

$$0 = \rho g \cos(\theta) + \mu \frac{d^2 v_z}{dy^2} \quad (15)$$

With BC's

$$\text{No slip: } v_z(y = b) = 0$$

$$\text{No flux of momentum: } \left. \frac{dv_z}{dy} \right|_{y=0} = 0$$

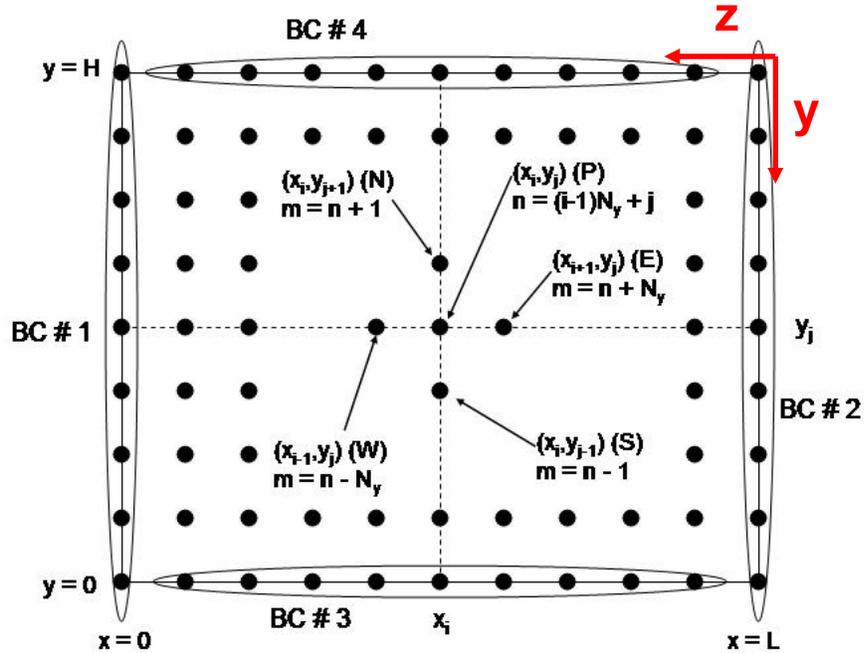
This can be solved analytically to yield:

$$v_z = \frac{b^2 \rho g \cos(\theta)}{2\mu} \left( 1 - \frac{y^2}{b^2} \right) \quad (16)$$

Now we have the velocity which we can plug into our convective diffusion reaction equations.

### *Concentration Field*

To solve the 2D concentration profile in this flowing film, we can use the same superimposed grid over this domain, and use finite difference approximations to make a calculation for each species at each given grid point. We define three 2-D matrices, one for each species in real space, meaning that these matrices will contain the concentration information in the geometry of the film. The one difficulty that arises is that since FSOLVE requires initial guesses and equations in the form of a vector, we will have to create an indexing method for when we interchange between real and vector space. We look at Fig 6.2 from Beers:



So here we look at a 2D grid that captures the physical space of the problem. In this solution we use a similar indexing method; except that we begin with an index of 1 in the top right corner, move right to left until index = Nz, where Nz is the number of grid points along the z-axis. We then begin on the subsequent row at Nz + 1 and then move from right to left until we reach an index of 2Nz. This continues until we reach the bottom left corner with the index = NyNz, where Ny is the total number of grid points along the y-axis. But because we physically have 3 equivalent matrices we need to incorporate an additional difference to our indexing method. Our total number of indices will be on the order of 3(NzNy).

Thus our final indexing method should be:

$$index = 3((y-1)N_z + (z-1)) + n \quad (17)$$

Where  $y = 1:Ny$ ,  $z = 1:Nz$ , and  $n = 1$  for cA,  $n = 2$  for cB and  $n = 3$  for cAB. We label  $n$  in the following manner to decrease the bandwidth of the Jacobian, keeping the matrix sparse and improving computational time. Remember that every time an additional dimension we do the following (using an example x-axis): add  $3*(x-1)NzNy$ , for  $x = 1:Nx$  and that would allow you to incorporate a third dimension into your index. To add a 4<sup>th</sup> dimension you add  $3*(k-1)*NxNyNz$ , etc.

### *Convection Diffusion Equation (with Reaction)*

Now that we have a way of getting between real space and vector space, in a method we call stacking and unstacking a vector, let's look at the equations we are trying to solve. To get the conservation equations we turn again to trusty Deen, this time p. 46. Again we look for a steady state solution, and look at the

system in 2D. Eliminating irrelevant terms, and filling in for the reaction term from the problem statement, we arrive at 3 PDEs describing each of our species:

$$v_z \frac{\partial C_A}{\partial z} = D \left[ \frac{\partial^2 C_A}{\partial y^2} + \frac{\partial^2 C_A}{\partial z^2} \right] - k \left( C_A C_B - \frac{C_{AB}}{K} \right) \quad (18)$$

$$v_z \frac{\partial C_B}{\partial z} = D \left[ \frac{\partial^2 C_B}{\partial y^2} + \frac{\partial^2 C_B}{\partial z^2} \right] - k \left( C_A C_B - \frac{C_{AB}}{K} \right) \quad (19)$$

$$v_z \frac{\partial C_{AB}}{\partial z} = D \left[ \frac{\partial^2 C_{AB}}{\partial y^2} + \frac{\partial^2 C_{AB}}{\partial z^2} \right] + k \left( C_A C_B - \frac{C_{AB}}{K} \right) \quad (20)$$

We need to solve for their concentrations at each grid point but they are all coupled due to the reaction term. Furthermore the boundary conditions for each species are different. However the equations for the internal grid points are mostly of the same form, so in this region we look solely at species A and the same equations can be treated similarly for species B and AB.

To numerically solve these coupled ODEs, we use a finite difference approximation. We imagine that we have a matrix  $C_A$  that contains that value of concentration at each discretized grid point. We further have a vector  $y$ , of length  $N_y$ , that stores the real-space value of  $y$ , a vector  $z$ , of length  $N_z$ , that stores the real-space value of  $z$ , and a vector  $v_z$  that stores the real value of velocity at that some specified value of  $y$ .

We begin with the convection term: As discussed in class, we learned that upwind differencing has an important impact on the numerical stability of our solution for certain local Peclet numbers. Thus for our convection term, using  $C_A$ , as an initial example, we will write:

$$v_z \frac{\partial C_A}{\partial z} \Big|_{(m,n)} = v_z(m) \left( \frac{C_{A(m,n)} - C_{A(m,n-1)}}{z_n - z_{n-1}} \right) \quad (21)$$

Where  $m$  is the index associated with the  $y$ -axis and  $n$  is the index associated with the  $z$ -axis. Similar equations can be derived for species B and AB. We also derive the second derivative diffusion equations for both the  $z$  and  $y$  axes:

$$D \frac{\partial^2 C_A}{\partial z^2} \Big|_{(m,n)} = D \left[ \frac{C_{A(m,n+1)} - C_{A(m,n)}}{(z_{n+1} - z_n)(z_{n+1} - z_{n-1})} - \frac{C_{A(m,n)} - C_{A(m,n-1)}}{(z_n - z_{n-1})(z_{n+1} - z_{n-1})} \right] \quad (22)$$

$$D \frac{\partial^2 C_A}{\partial y^2} \Big|_{(m,n)} = D \left[ \frac{C_{A(m+1,n)} - C_{A(m,n)}}{(y_{m+1} - y_m)(y_{m+1} - y_{m-1})} - \frac{C_{A(m,n)} - C_{A(m-1,n)}}{(y_m - y_{m-1})(y_{m+1} - y_{m-1})} \right] \quad (23)$$

These can also be done for the other species as well. Lastly we incorporate the reaction term:

$$r_R = k \left( C_{A(m,n)} C_{B(m,n)} - \frac{C_{AB(m,n)}}{K} \right) \quad (24)$$

Substituting these representations into the terms in the PDEs, we arrive at nonlinear algebraic equations in the form of the following

$$0 = A_1 C_{A(m,n)} + A_2 C_{A(m+1,n)} + A_3 C_{A(m-1,n)} + A_4 C_{A(m,n+1)} + A_5 C_{A(m,n-1)} - r_R \quad (25)$$

We make some simple assignment calls:

$$\begin{aligned} dz\_mid &= z_{n+1} - z_{n-1} \\ dz\_lo &= z_n - z_{n-1} \\ dz\_hi &= z_{n+1} - z_n \\ dy\_mid &= y_{m+1} - y_{m-1} \\ dy\_lo &= y_m - y_{m-1} \\ dy\_hi &= y_{m+1} - y_m \end{aligned}$$

And now to define some of the A\_coefficients:

$$A_1 = \frac{-v_z(m)}{dz\_lo} - \frac{D}{dz\_mid} \left( \frac{1}{(dz\_hi)} + \frac{1}{(dz\_lo)} \right) - \frac{D}{dy\_mid} \left( \frac{1}{(dy\_hi)} + \frac{1}{(dy\_lo)} \right) \quad (26)$$

$$A_2 = \frac{D}{dy\_mid} \left( \frac{1}{(dy\_hi)} \right) \quad (27)$$

$$A_3 = \frac{D}{dy\_mid} \left( \frac{1}{(dy\_lo)} \right) \quad (28)$$

$$A_4 = \frac{D}{dz\_mid} \left( \frac{1}{(dz\_hi)} \right) \quad (29)$$

$$A_5 = \frac{v_z(m)}{dz\_lo} + \frac{D}{dz\_mid} \left( \frac{1}{(dz\_lo)} \right) \quad (30)$$

These coefficients are easy to calculate if we are passed the value of the indices  $m$  and  $n$ , given the vectors  $z$  and  $y$  and we write a function to take care of this. Similar equations can be written for the matrices of  $c_B$  and  $c_{AB}$  with the one difference in  $c_{AB}$ , that the reaction term  $r_R$  is added rather than subtracted.

We need to remember to pass along values of the Jacobian, because this will significantly reduce the amount of computation time. Luckily the Jacobian is easy to calculate. Since each equation is dependent on only at most seven variables, it is fairly straight forward to calculate these values. We go through and determine the non-zero elements for the equation  $f(\text{index}(m,n))$ :

$$J_{(\text{index},\text{index})} = \frac{\partial f_{\text{index}}}{\partial C_{A(m,n)}} = A_1 - kC_{B(m,n)} \quad (31)$$

$$J_{(\text{index},\text{index}+1)} = \frac{\partial f_{\text{index}}}{\partial C_{B(m,n)}} = -kC_{A(m,n)} \quad (32)$$

$$J_{(\text{index},\text{index}+2)} = \frac{\partial f_{\text{index}}}{\partial C_{AB(m,n)}} = \frac{-k}{K} \quad (33)$$

$$J_{(\text{index},\text{index}+3)} = \frac{\partial f_{\text{index}}}{\partial C_{A(m,n+1)}} = A_4 \quad (34)$$

$$J_{(\text{index},\text{index}-3)} = \frac{\partial f_{\text{index}}}{\partial C_{A(m,n-1)}} = A_5 \quad (35)$$

$$J_{(\text{index},\text{index}+3N_z)} = \frac{\partial f_{\text{index}}}{\partial C_{A(m+1,n)}} = A_2 \quad (36)$$

$$J_{(\text{index},\text{index}-3N_z)} = \frac{\partial f_{\text{index}}}{\partial C_{A(m-1,n)}} = A_3 \quad (37)$$

For species B and AB, there will be slight modifications to equations (32) and (33), but it remains fairly straightforward. In this manner we can handle all of the interior boundary points.

### *Boundary Conditions*

Like most PDEs, the boundary conditions dictate what the solution winds up looking like. In our problem here we have boundary conditions for each species at each edge of the domain. The problem is further complicated by at the corners where multiple boundary conditions arise from the joining of boundaries

along both axes. We begin by looking at the edges, neglecting the corners for now and go into detail into how we handle the boundaries.

Let's begin by looking at the right edge, where the flow of material enters at concentrations of  $c_A = 0$ ,  $c_B = 1$  M,  $c_{AB} = 0$ . This is a Dirichlet boundary condition for all species and is relatively easy to handle. We simply replace the value of concentration at the imaginary grid point to the right of our domain, with the specified concentration. For example:

$$f(\text{index}) = A_1 C_{A(m,1)} + A_2 C_{A(m,2)} + A_3(0) + A_4 C_{A(m+1,1)} + A_5 C_{A(m-1,1)} - r_R \quad (38)$$

$$f(\text{index}) = A_1 C_{B(m,1)} + A_2 C_{B(m,2)} + A_3(1M) + A_4 C_{B(m+1,1)} + A_5 C_{B(m-1,1)} - r_R \quad (39)$$

The equation for  $c_{AB}$  would be something similar to (38). It is relatively easy to handle Dirichlet conditions. Of course the Jacobian matrix will have to be modified to reflect this change. In the above examples illustrated in equation (38) and (39), the terms which are now set to constants have Jacobian values which are equal to zero there.

Now at the left edge, we specify that there will be no change in the concentration as a function of  $z$  any more. This condition is somewhat klugy because we don't know if in fact this is the case, but because we do not know the final concentrations a priori, we substitute Neumann conditions in as a reasonable guess. In practice we might invoke Danckwert's BCs and let the parameters of the problem dictate the behavior at the edge. We invoke the same treatment of boundary conditions that we used to arrive at eqn (12). However the use of Lagrange polynomials gets much more complicated. If we assume that we have a locally uniform grid spacing, we can still use the same analysis and arrive at a similar expression:

$$\frac{dC_{A(m,Nz)}}{dz} = 0 = \frac{-3C_{A(m,Nz+1)} + 4C_{A(m,Nz)} - C_{A(m,Nz-1)}}{2(dz)} \quad (40)$$

We can rearrange again to get a value for the imaginary grid point to the right of left of the grid. So the effect of the treatment of the Neumann BC, is to solve for a value of  $C_{A(m,Nz+1)}$  and replace this value, which depends solely on the values of  $C_{A(m,Nz)}$  and  $C_{A(m,Nz-1)}$ , into the equation for that grid point. This is similarly done for  $C_B$  and  $C_{AB}$  since they all have Neumann conditions here.

The Jacobians are similarly modified with the exceptions that the term relating to the imaginary grid point  $(m,Nz+1)$ , is now set equal to zero, and the Jacobian terms involving grid points  $(m,Nz)$  and  $(m,Nz-1)$  now have an additional factor of  $4/3(D/dz^2)$  and  $-1/3(D/dz^2)$ , respectively.

Next we examine the top boundary conditions. In class, professor Beers discussed some of the difficulties in handling this particular boundary condition. A truly physical problem would either invoke a Neumann condition which reflected volatilization of species AB and B into the air, or invoke a Dirichlet condition which would take into account the partial pressure of AB and B in the atmosphere. We choose to look at the simplified version in which the concentrations of B and AB are = 0 (thus Dirichlet conditions). However choosing to use no flux (Neumann conditions) will also be considered correct.

We've already seen both of these conditions now. The Dirichlet conditions are easy to implement, in that for the imaginary grid point above the grid, you replace the concentration with the constant value specified at the boundary condition. The No flux condition is similarly implemented and is comparable to what we see in equation (40). The key thing to remember now is that you have a no-flux condition in the y direction so the additional factors that are added to grid points (1,n) and (2,n) are now  $4/3(D/dy^2)$  and  $-1/3(D/dy^2)$ , respectively. The bottom boundary conditions are handled as no-flux conditions (no penetration into the plate) and can be treated exactly the same as the top condition if you choose to use the Neumann condition.

So what about the corners? The corners just happen to have boundary conditions in both axes and you can incorporate them sequentially. The only one that is a little tricky and you need to remember, is that if you have two Neumann conditions, you need to remember that there are two contributions to the central grid point:  $4/3(D/dy^2)$  and  $4/3(D/dz^2)$ . With these in place, you will have specified all the internal grid points along with all of the edge points.

Code that could handle this might be something like:

```
% benwang_P6C5.m
% Ben Wang
% HW#7 Problem #3
% due 11/7/05 9 am

% Holy crap. Solving a couple of fluid and transport problem

% ===== main routine benwang_P6C5.m
function iflag_main = benwang_P6C5();

iflag_main = 0;

%PDL> clear graphs, screen etc. general initialization
clear all; close all; clc;

%PDL> specify parameters
param.pA = 1e-4;           % [atm]
param.HA = 1e3;           % [M/atm]
param.k = 1e-2;          % [L/mol-s]
param.K = 1e3;
```

```

param.D = 1e-5/(1e2)^2;           % [m^2/s]
param.b = 1e-3;                   % [m]
param.L = 50e-2;                  % [m]
param.mu = 1e-3;                  % [Pa]
param.rho = 1e3;                  % [kg/m^3]
param.theta = 80/180*pi;         % [rad]
param.g = 9.8;                    % [m/s^2]
param.CB0 = 1;                    % [M]

```

**%PDL> specify grid parameters**

```

grid.z = 10;                      % number of grid points along the flow direction
grid.y1 = 20;                     % number of grid points in the orthogonal to flow
grid.y2 = 11 ;
grid.y = grid.y1 + grid.y2 -1;

```

```

grid.dz = param.L/(grid.z+1);

```

```

y1 = linspace(0,param.b/30,grid.y1);
y2 = linspace(param.b/30, param.b, grid.y2);

```

**%PDL> initialize 3D matrix, where out of the plane we have the different  
%speciesd efine index for 2D grid, here we move from right to left, top to bottom,  
% to keep the convention of the diagram**

```

c = zeros(grid.y,grid.z,3);
cA = c(:, :, 1);
cB = c(:, :, 2);
cAB = c(:, :, 3);

```

**%PDL> calculate flow velocity**

```

z = linspace(0, param.L, grid.z);
grid.y_vector = [y1 y2(2:end)];
grid.z_vector = linspace(0,param.L,grid.z);
y = grid.y_vector;
param.vel = calc_vel(grid.y_vector,param);

```

**%PDL> collapse matrix into a single vector**

```

guess = zeros(3*grid.y*grid.z,1);

```

```

for i = 1:grid.y
    for j = 1:grid.z
        for k = 1:3
            index = 3*((i-1)*grid.z + (j-1)) + k;
            if k == 2
                guess(index) = param.CB0;
            else
            end
        end
    end
end

```

```

%PDL> Call Fsolve
options = optimset('Jacobian','on');
[x, f] = fsolve(@calc_func_P6C5, guess, options, param, grid);

%PDL> extract results

for i = 1:grid.y
    for j = 1:grid.z

        index_cA = 3*(i-1)*grid.z + 3*(j-1) + 1;
        index_cB = 3*(i-1)*grid.z + 3*(j-1) + 2;
        index_cAB = 3*(i-1)*grid.z + 3*(j-1) + 3;

        cA(i,j) = x(index_cA);
        cB(i,j) = x(index_cB);
        cAB(i,j) = x(index_cAB);
    end
end

figure;
pcolor(param.L-z,param.b-grid.y_vector,cA);
shading('interp');
colorbar;
title('Concentration of cA with Dirichlet Condition');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-grid.y_vector,cB);
shading('interp');
colorbar;
title('Concentration of cB with Dirichlet Condition');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-grid.y_vector,cAB);
shading('interp');
colorbar;
title('Concentration of cAB with Dirichlet Condition');
ylabel('y (m)');
xlabel('z (m)');

figure;
semilogy(z,cA(1,:),'s');
hold on;
semilogy(z,cB(1,:),'-');
semilogy(z,cAB(1,:),'-');
title('Concentration at surface Dirichlet Condition');
ylabel('y (m)');
xlabel('z (m)');
legend('cA','cB','cAB');

```

```

cAB = cAB(1:20,:);
cA = cA(1:20,:);
cB = cB(1:20,:);

y = grid.y_vector(1:20);

figure;
pcolor(param.L-z,param.b-y,cAB);
shading('interp');
colorbar;
title('Concentration of cAB with Dirichlet Condition (Blown Up)');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-y,cA);
shading('interp');
colorbar;
title('Concentration of cA with Dirichlet Condition (Blown Up)');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-y,cB);
shading('interp');
colorbar;
title('Concentration of cB with Dirichlet Condition (Blown Up)');
ylabel('y (m)');
xlabel('z (m)');

flux = zeros(length(z),1);
cA_int = zeros(length(z),1);

for i = 1:length(z)
    flux(i) = (-3*param.HA*param.pA+4*cA(1,i)-cA(2,i))/(2*(y(2)-y(1)))+...
        (0 + 4*cAB(1,i)-cAB(2,i))/(2*(y(2)-y(1)));
    cA_int(i) = trapz(y,cA(:,i))+trapz(y,cAB(:,i));
end

flux_A_surface = -trapz(z,flux)*param.D/param.L
integral_A_AB = trapz(z,cA_int)/param.L/param.b/100

return;

%===== subroutine calc_volflow.m
function f = calc_vel(y,param)
f = zeros(1, length(y));
constant = param.rho*param.g*cos(param.theta)/param.mu;
f = -constant*y.^2+1e-6*constant;
return
%===== subroutine calc_coeff_P6C5.m
function [A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);

```

```

y = grid.y_vector;
z = grid.z_vector;
v = param.vel;
D = param.D;

if m==1
    dy_mid = y(2)-y(1);
    dy_lo = dy_mid;
    dy_hi = dy_mid;
elseif m == grid.y
    dy_mid = y(end) - y(end-1);
    dy_lo = dy_mid;
    dy_hi = dy_mid;
else
    dy_mid = (y(m+1)-y(m-1))/2;
    dy_lo = y(m)-y(m-1);
    dy_hi = y(m+1)-y(m);
end

if n==1
    dz_mid = (z(2)-z(1));
    dz_hi = dz_mid;
    dz_lo = dz_mid;
elseif n == grid.z
    dz_mid = z(end)-z(end-1);
    dz_lo = dz_mid;
    dz_hi = dz_mid;
else
    dz_mid = (z(n+1)-z(n-1))/2;
    dz_lo = z(n)-z(n-1);
    dz_hi = z(n+1)-z(n);
end

A1 = -v(m)/dz_lo-D/dz_mid*(dz_hi^-1+dz_lo^-1)-D/dy_mid*(dy_hi^-1+dy_lo^-1);
A2 = D/dy_mid/dy_hi;
A3 = D/dy_mid/dy_lo;
A4 = D/dz_mid/dz_hi;
A5 = v(m)/dz_lo + D/dz_mid/dz_lo;

if k==3
    rR = param.k;
else
    rR = -param.k;
end

return
%===== subroutine calc_bc_coeff.m
function [BC1, BC2, BC3, BC4, BC5] = calc_bc_coeff(m,n,k,param,grid);

[A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);

y = grid.y_vector;
z = grid.z_vector;
v = param.vel;

```

```
BC1 = 0;
BC2 = 0;
BC3 = 0;
BC4 = 0;
BC5 = 0;
```

```
y_BC = A2;
z_BC = A4;
```

```
% right boundary condition (Dirichlet)
```

```
if n == 1
    if k == 1
        BC5 = 0;
    elseif k == 2
        BC5 = param.CB0*A5;
    elseif k == 3
        BC5 = 0;
    end
end
```

```
% left boundary condition (Neumann)
```

```
if n == grid.z
    BC1 = BC1 + 4/3*z_BC;
    BC4 = 0;
    BC5 = BC5 - 1/3*z_BC;
end
```

```
% upper boundary condition (Dirichlet)
```

```
if m == 1
    if k == 1
        BC3 = param.HA*param.pA*y_BC;
    elseif k == 2
        BC3 = 0;
    elseif k == 3
        BC3 = 0;
    end
end
```

```
% lower boundary condition (Neumann)
```

```
if m == grid.y
    BC1 = BC1 + 4/3*y_BC;
    BC2 = 0;
    BC3 = BC3 - 1/3*y_BC;
end
```

```
return
```

```
%===== subroutine get_Jac_indices.m
```

```

function [p,p2,p3,p4,p5,p6,p7] = get_Jac_indices(m,n,k,grid);

index = 3*((m-1)*grid.z + (n-1)) + k;
p = index;
if k == 1
    p2 = index+1;
    p3 = index+2;
elseif k==2
    p2 = index-1;
    p3 = index+1;
else
    p2 = index-2;
    p3 = index-1;
end

p4 = index+3*grid.z;
p5 = index-3*grid.z;
p6 = index+3;
p7 = index-3;

return

%===== subroutine calc_func_P6C5.m
function [f, Jac] = calc_func_P6C5(x0, param, grid)
%function [f] = calc_func_P6C5(x0, param, grid)
f = zeros(length(x0),1);
v = param.vel;
% extract vector into more physical format

c = zeros(grid.y,grid.z,3);

Jac = spalloc(length(x0),length(x0),7*length(x0));

for m = 1:grid.y
    for n = 1:grid.z
        for k = 1:3
            index_c = 3*((m-1)*grid.z + (n-1)) + k;
            c(m,n,k) = x0(index_c);
        end
    end
end

for m = 1:grid.y
    for n = 1:grid.z
        for k = 1:3

            [A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);
            [BC1,BC2,BC3,BC4,BC5] = calc_bc_coeff(m,n,k,param,grid);

            % get function index
            index_c = 3*((m-1)*grid.z + (n-1)) + k;
            % top condition
            if m == 1
                % upper right condition (2 Dirichlet)

```

```

if n == 1
    f(index_c) = A1*c(m,n,k) + A2*c(m+1,n,k) + BC3 + A4*c(m,n+1,k) + ...
                BC5 + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
    % upper left condition (1 Dirichlet and 1 Neumann)
elseif n == grid.z
    f(index_c) = (A1+BC1)*c(m,n,k) + A2*c(m+1,n,k) + BC3 + ...
                (A5+BC5)*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
else
    % purely top condition (Dirichlet)
    f(index_c) = A1*c(m,n,k) + A2*c(m+1,n,k) + BC3 + A4*c(m,n+1,k) + ...
                A5*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
end
% bottom boundary condition
elseif m == grid.y
    if n == 1
        % lower right condition (1 Dirichlet and 1 Neumann)
        f(index_c) = (A1+BC1)*c(m,n,k) + (A3+BC3)*c(m-1,n,k) + A4*c(m,n+1,k) + ...
                    BC5 + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
    elseif n == grid.z
        % lower left condition (2 Neumann)
        f(index_c) = (A1+BC1)*c(m,n,k) + (A3+BC3)*c(m-1,n,k) + (A5+BC5)*c(m,n-
1,k) + ...
                    rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
    else
        % bottom condition (1 Neumann)
        f(index_c) = (A1+BC1)*c(m,n,k) + (A3+BC3)*c(m-1,n,k) + A4*c(m,n+1,k) + ...
                    A5*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
    end
else
    % right boundary (Dirichlet)
    if n == 1
        f(index_c) = A1*c(m,n,k) + A2*c(m+1,n,k) + A3*c(m-1,n,k) + ...
                    A4*c(m,n+1,k) + BC5 + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        %left boundary (Neumann)
    elseif n == grid.z
        f(index_c) = (A1+BC1)*c(m,n,k) + A2*c(m+1,n,k) + A3*c(m-1,n,k) + ...
                    (A5+BC5)*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
    else
        %interior grid points
        f(index_c) = A1*c(m,n,k) + A2*c(m+1,n,k) + A3*c(m-1,n,k) + ...
                    A4*c(m,n+1,k) + A5*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
    end
end
end
end
end
end

for m = 1:grid.y
    for n = 1:grid.z
        for k = 1:3
            [p,p2,p3,p4,p5,p6,p7] = get_Jac_indices(m,n,k,grid);

            index_c = 3*((m-1)*grid.z + (n-1)) + k;
            [c1,c2,c3,c4,c5,c6,c7] = get_jac_const(m,n,k,param,grid,c);
            if (index_c <= 3*grid.z) | (index_c >= length(x0)-3*grid.z)

```

```

if (index_c <= 3*grid.z)
  if (index_c <= 3)
    Jac(p,p) = c1;
    Jac(p,p2) = c2;
    Jac(p,p3) = c3;
    Jac(p,p4) = c4;
    Jac(p,p6) = c6;
  else
    Jac(p,p) = c1;
    Jac(p,p2) = c2;
    Jac(p,p3) = c3;
    Jac(p,p4) = c4;
    Jac(p,p6) = c6;
    Jac(p,p7) = c7;
  end
elseif (index_c >= length(x0)-3*grid.z)
  if (index_c >= length(x0) - 3)
    Jac(p,p) = c1;
    Jac(p,p2) = c2;
    Jac(p,p3) = c3;
    Jac(p,p5) = c5;
    Jac(p,p7) = c7;
  else
    Jac(p,p) = c1;
    Jac(p,p2) = c2;
    Jac(p,p3) = c3;
    Jac(p,p5) = c5;
    Jac(p,p6) = c6;
    Jac(p,p7) = c7;
  end
end
end
else
  Jac(p,p) = c1;
  Jac(p,p2) = c2;
  Jac(p,p3) = c3;
  Jac(p,p4) = c4;
  Jac(p,p5) = c5;
  Jac(p,p6) = c6;
  Jac(p,p7) = c7;
end
end
end
end

spy(Jac)
return

% ===== subroutine get_jac_const.m

function [c1,c2,c3,c4,c5,c6,c7] = get_jac_const(m,n,k,param,grid,c)

[A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);
[BC1,BC2,BC3,BC4,BC5] = calc_bc_coeff(m,n,k,param,grid);

if k == 1

```

```

c1 = A1 + rR*c(m,n,2);
c2 = rR*c(m,n,1);
c3 = -rR/param.K;
elseif k == 2
c1 = A1 + rR*c(m,n,1);
c2 = rR*c(m,n,2);
c3 = -rR/param.K;
elseif k == 3
c1 = A1 - rR/param.K;
c2 = rR*c(m,n,2);
c3 = rR*c(m,n,1);
end

```

```

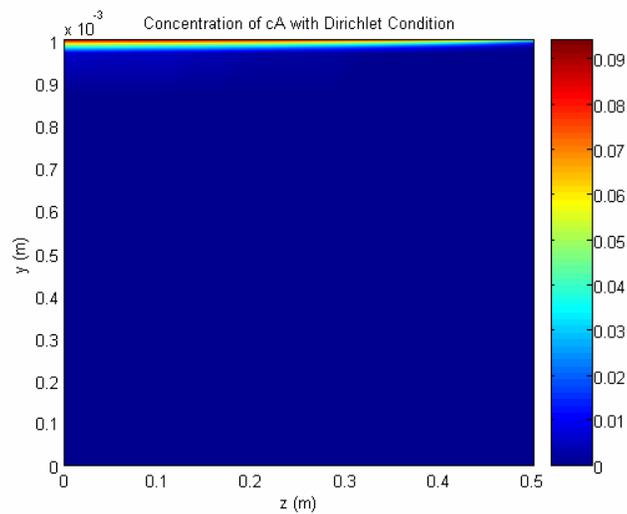
if m == 1
if n == 1
c4 = A2;
c5 = 0;
c6 = A4;
c7 = 0;
elseif n == grid.z
c1 = c1 + BC1;
c4 = A2;
c5 = 0;
c6 = 0;
c7 = A5 + BC5;
else
c4 = A2;
c5 = 0;
c6 = A4;
c7 = A5;
end
elseif m == grid.y
if n == 1
c1 = c1 + BC1;
c4 = 0;
c5 = A3 + BC3;
c6 = A4;
c7 = 0;
elseif n == grid.z
c1 = c1 + BC1;
c4 = 0;
c5 = A3 + BC3;
c6 = 0;
c7 = A5 + BC5;
else
c1 = c1 + BC1;
c4 = 0;
c5 = A3 + BC3;
c6 = A4;
c7 = A5;
end
else
if n == 1
c4 = A2;
c5 = A3;
c6 = A4;

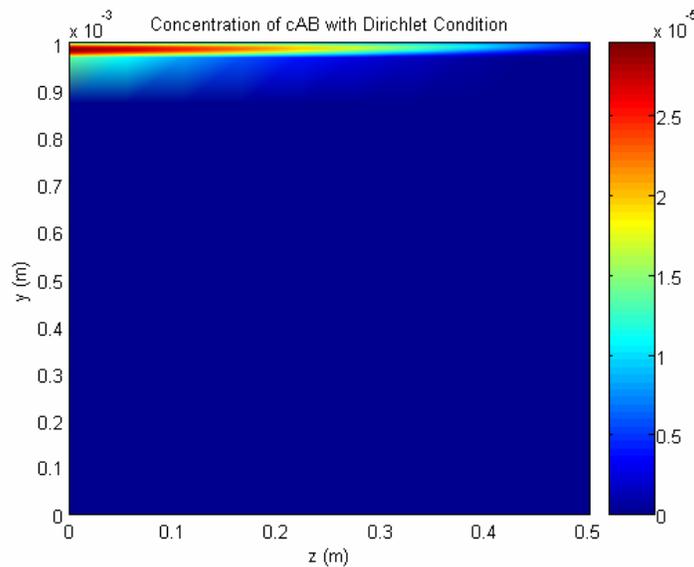
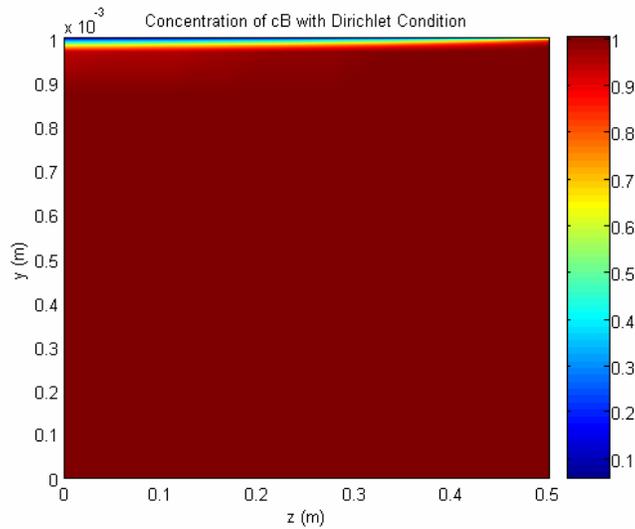
```

```
c7 = 0;
elseif n == grid.z
    c1 = c1 + BC1;
    c4 = A2;
    c5 = A3;
    c6 = 0;
    c7 = A5 + BC5;
else
    c4 = A2;
    c5 = A3;
    c6 = A4;
    c7 = A5;
end
end

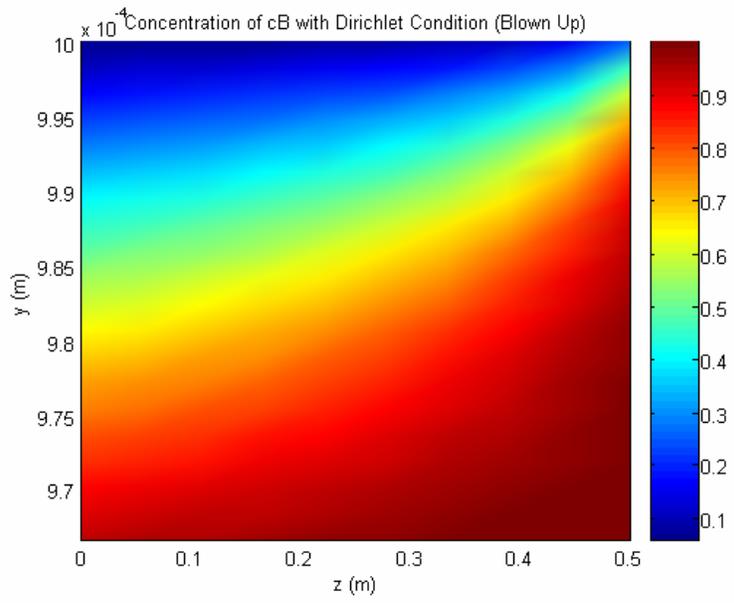
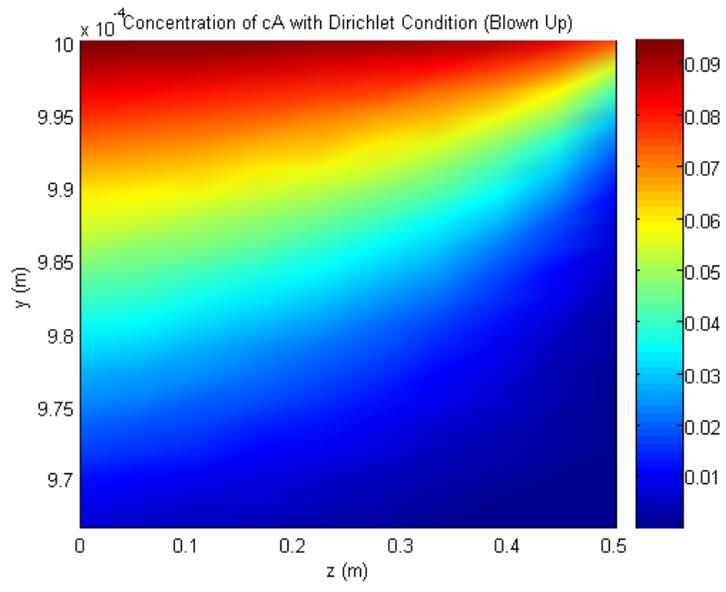
return
```

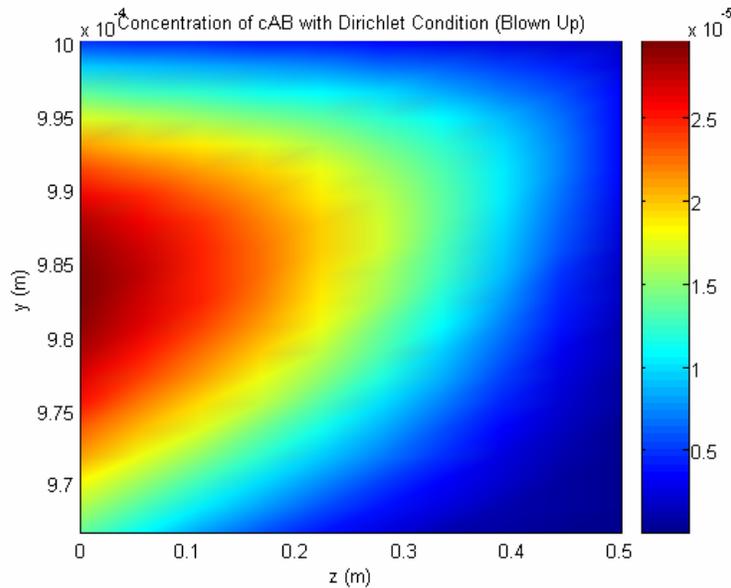
The steady state concentration profiles for each species might look like (note despite the numerical values, the origin of these charts are in the upper right with flow traveling from right to left):





We notice immediately, that most of the action occurs very close to the surface, which makes sense, since we have a pretty fast velocity with relatively slow diffusion. Because of this, we take advantage of uneven grid spacing to run our simulation, with a finer mesh close to the surface. We note that if the values at the surface continue to change as we increase the mesh then the grid spacing is not tight enough. If you pass FSOLVE either the sparsity pattern of the Jacobian, or the Jacobian, itself you could run a pretty accurate simulation in a reasonable time. If you would be able to do this and look at the simulation only close to the surface, you might get graphs such as these (blown up near the region of interest)





Remember that the origin is on the upper right corner.

### Absorption Rate

We are also asked for the average absorption rate per unit area. Since the goal of the problem was to increase the amount of absorption of A from the gas stream, by the addition of a reactive falling film, we'd like to determine how much extra A gets into the stream. We can do this in one of two ways. We can calculate the 'flux' of A at the interface of liquid and gas. We can also integrate the total amount of A in the film, and normalize it to some unit length.

Let's look at the flux of A at the interface. We again use the equation from page 376. We now plug in certain values and integrate over the length of the film:

$$\frac{dC_{A(1,n)}}{dy} = \frac{-3C_{A(0,n)} + 4C_{A(1,n)} - C_{A(2,n)}}{2(y_2 - y_1)} \quad (41)$$

$$mean\_flux = \frac{-1}{L} \int_0^L D \frac{dC_{A(1,n)}}{dy} dz \quad (42)$$

If we do a full integration of all the A in the film, we just do a 2-D integration across y and z of all cA and cAB.

$$mean\_A = \frac{1}{bL} \int_0^L \int_0^b (C_A + C_{AB}) dy dz \quad (43)$$

We accomplish these integrations using trapz. We are asked to calculate these mean absorption rates and we note that we get the following values:

flux\_A\_surface =

5.7474e-006

integral\_A\_AB =

1.1339e-005

We compare this with the rate constant  $k$ , set to 0.

flux\_A\_surface =

5.7474e-006

integral\_A\_AB =

1.1339e-005

When look at the two, we see no improvement. This makes sense because we have effectively locked in all the material by specifying no flux at the right and bottom of A, as well as a specified concentration at top. Therefore, the reaction term serves to only change species  $A \rightarrow AB$ .

By changing the top boundary condition to a no-flux condition, we can get an improvement in absorption. To get to this boundary condition, it is just to apply our standard  $4/3 - 1/3$  treatment as per before (the code is very similar, but is included at the end for your perusal if you wish). For further consideration of the Neumann condition, you might get numbers on the order of these (note that the number here were calculated for different grid spacings, which result in different absolute numbers).

**Neumann boundary conditions (no flux at surface for AB, B):**

flux\_A\_surface =

5.7495e-006

integral\_A\_AB =

1.1349e-005

We compare this with the rate constant  $k$ , set to 0.

```
flux_A_surface =
```

```
5.7474e-006
```

```
integral_A_AB =
```

```
1.1339e-005
```

So it is acceptable for the Dirichlet boundary condition to see no relative increase in extraction... this is because much of that AB (and B, for that matter) has evaporated off into the air. We'd need to model that as well in order to get a fuller picture of what has happened. As Dr. Beers pointed out in class, a way to inelegantly force things to look like we'd like is to apply the Neumann condition for AB and B... in this case, we see an increase in extractive potential. We note that it is small, though this may be expected because the reaction is relatively slow (less than 0.1 M/s) and the fluid is flowing fast (0.8 m/s) over a short span (0.5 m). If you were to crank up the value of the reaction constant  $k$ , you'd see a significant improvement of extraction for the reactive extraction. And here are plots and code... first for Dirichlet boundary conditions, then for von Neumann boundary conditions.

Note that different rate constants and top boundary conditions do not actually change the graphs all that much, at least visually. So the graphs should be about the same if you chose either boundary condition, so the others are not included here. You can compare against part 2 of this solution, also attached provided on MIT server.

```
% benwang_P6C5.m
% Ben Wang
% HW#7 Problem #3
% due 11/7/05 9 am

% Holy crap. Solving a couple of fluid and transport problem

% ===== main routine benwang_P6C5.m
function iflag_main = benwang_P6C5();

iflag_main = 0;

%PDL> clear graphs, screen etc. general initialization
clear all; close all; clc;

%PDL> specify parameters
param.pA = 1e-4;           % [atm]
param.HA = 1e3;           % [M/atm]
param.k = 1e-2;           % [L/mol-s]
```

```

param.K = 1e3;
param.D = 1e-5/(1e2)^2;           % [m^2/s]
param.b = 1e-3;                   % [m]
param.L = 50e-2;                  % [m]
param.mu = 1e-3;                  % [Pa]
param.rho = 1e3;                  % [kg/m^3]
param.theta = 80/180*pi;         % [rad]
param.g = 9.8;                   % [m/s^2]
param.CB0 = 1;                   % [M]

```

**%PDL> specify grid parameters**

```

grid.z = 10;                      % number of grid points along the flow direction
grid.y1 = 20;                     % number of grid points in the orthogonal to flow
grid.y2 = 11 ;
grid.y = grid.y1 + grid.y2 -1;

```

```

grid.dz = param.L/(grid.z+1);

```

```

y1 = linspace(0,param.b/30,grid.y1);
y2 = linspace(param.b/30, param.b, grid.y2);

```

**%PDL> initialize 3D matrix, where out of the plane we have the different  
%speciesd efine index for 2D grid, here we move from right to left, top to bottom,  
% to keep the convention of the diagram**

```

c = zeros(grid.y,grid.z,3);
cA = c(:, :, 1);
cB = c(:, :, 2);
cAB = c(:, :, 3);

```

**%PDL> calculate flow velocity**

```

z = linspace(0, param.L, grid.z);
grid.y_vector = [y1 y2(2:end)];
grid.z_vector = linspace(0,param.L,grid.z);
y = grid.y_vector;
param.vel = calc_vel(grid.y_vector,param);

```

**%PDL> collapse matrix into a single vector**

```

guess = zeros(3*grid.y*grid.z,1);

```

```

for i = 1:grid.y
    for j = 1:grid.z
        for k = 1:3
            index = 3*((i-1)*grid.z + (j-1)) + k;
            if k == 2
                guess(index) = param.CB0;
            else
            end
        end
    end
end
end

```

```

%PDL> Call Fsolve
options = optimset('Jacobian','on');
[x, f] = fsolve(@calc_func_P6C5, guess, options, param, grid);

%PDL> extract results

for i = 1:grid.y
    for j = 1:grid.z

        index_cA = 3*(i-1)*grid.z + 3*(j-1) + 1;
        index_cB = 3*(i-1)*grid.z + 3*(j-1) + 2;
        index_cAB = 3*(i-1)*grid.z + 3*(j-1) + 3;

        cA(i,j) = x(index_cA);
        cB(i,j) = x(index_cB);
        cAB(i,j) = x(index_cAB);
    end
end

figure;
pcolor(param.L-z,param.b-grid.y_vector,cA);
shading('interp');
colorbar;
title('Concentration of cA with Neumann Condition');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-grid.y_vector,cB);
shading('interp');
colorbar;
title('Concentration of cB with Neumann Condition');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-grid.y_vector,cAB);
shading('interp');
colorbar;
title('Concentration of cAB with Neumann Condition');
ylabel('y (m)');
xlabel('z (m)');

figure;
semilogy(z,cA(1,:),'s');
hold on;
semilogy(z,cB(1,:),'-');
semilogy(z,cAB(1,:),'-');
title('Concentration at surface Neumann Condition');
ylabel('y (m)');
xlabel('z (m)');
legend('cA','cB','cAB');

```

```

cAB = cAB(1:20,:);
cA = cA(1:20,:);
cB = cB(1:20,:);

y = grid.y_vector(1:20);

figure;
pcolor(param.L-z,param.b-y,cAB);
shading('interp');
colorbar;
title('Concentration of cAB with Neumann Condition (Blown Up)');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-y,cA);
shading('interp');
colorbar;
title('Concentration of cA with Neumann Condition (Blown Up)');
ylabel('y (m)');
xlabel('z (m)');

figure;
pcolor(param.L-z,param.b-y,cB);
shading('interp');
colorbar;
title('Concentration of cB with Neumann Condition (Blown Up)');
ylabel('y (m)');
xlabel('z (m)');

flux = zeros(length(z),1);
cA_int = zeros(length(z),1);

for i = 1:length(z)
    flux(i) = (-3*param.HA*param.pA+4*cA(1,i)-cA(2,i))/(2*(y(2)-y(1)))+...
        (4*cAB(1,i)-cA*2,i)/(2*(y(2)-y(1)));
    cA_int(i) = trapz(y,cA(:,i))+trapz(y,cAB(:,i));
end

flux_A_surface = -trapz(z,flux)*param.D/param.L
integral_A_AB = trapz(z,cA_int)/param.L/param.b/100

return;

%===== subroutine calc_volfow.m
function f = calc_vel(y,param)
f = zeros(1, length(y));
constant = param.rho*param.g*cos(param.theta)/param.mu;
f = -constant*y.^2+1e-6*constant;
return
%===== subroutine calc_coeff_P6C5.m
function [A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);

y = grid.y_vector;

```

```

z = grid.z_vector;
v = param.vel;
D = param.D;

if m==1
    dy_mid = y(2)-y(1);
    dy_lo = dy_mid;
    dy_hi = dy_mid;
elseif m == grid.y
    dy_mid = y(end) - y(end-1);
    dy_lo = dy_mid;
    dy_hi = dy_mid;
else
    dy_mid = (y(m+1)-y(m-1))/2;
    dy_lo = y(m)-y(m-1);
    dy_hi = y(m+1)-y(m);
end

if n==1
    dz_mid = (z(2)-z(1));
    dz_hi = dz_mid;
    dz_lo = dz_mid;
elseif n == grid.z
    dz_mid = z(end)-z(end-1);
    dz_lo = dz_mid;
    dz_hi = dz_mid;
else
    dz_mid = (z(n+1)-z(n-1))/2;
    dz_lo = z(n)-z(n-1);
    dz_hi = z(n+1)-z(n);
end

A1 = -v(m)/dz_lo-D/dz_mid*(dz_hi^-1+dz_lo^-1)-D/dy_mid*(dy_hi^-1+dy_lo^-1);
A2 = D/dy_mid/dy_hi;
A3 = D/dy_mid/dy_lo;
A4 = D/dz_mid/dz_hi;
A5 = v(m)/dz_lo + D/dz_mid/dz_lo;

if k==3
    rR = param.k;
else
    rR = -param.k;
end

return
%===== subroutine calc_bc_coeff.m
function [BC1, BC2, BC3, BC4, BC5] = calc_bc_coeff(m,n,k,param,grid);

[A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);

y = grid.y_vector;
z = grid.z_vector;
v = param.vel;

```

```
BC1 = 0;
BC2 = 0;
BC3 = 0;
BC4 = 0;
BC5 = 0;
```

```
y_BC = A2;
z_BC = A4;
```

```
% right boundary condtion (Dirichlet)
```

```
if n == 1
    if k == 1
        BC5 = 0;
    elseif k == 2
        BC5 = param.CB0*A5;
    elseif k == 3
        BC5 = 0;
    end
end
```

```
% left boundary condition (Neumann)
```

```
if n == grid.z
    BC1 = BC1 + 4/3*z_BC;
    BC4 = 0;
    BC5 = BC5 - 1/3*z_BC;
end
```

```
% upper boundary condition (Neumann)
```

```
if m == 1
    if k == 1
        BC3 = param.HA*param.pA*y_BC;
    elseif k == 2
        BC1 = BC1 + 4/3*z_BC;
        BC2 = BC2 - 1/3*z_BC;
    elseif k == 3
        BC1 = BC1 + 4/3*z_BC;
        BC2 = BC2 - 1/3*z_BC;
    end
end
```

```
% lower boundary condition (Neumann)
```

```
if m == grid.y
    BC1 = BC1 + 4/3*y_BC;
    BC2 = 0;
    BC3 = BC3 - 1/3*y_BC;
end
```

```
return
```

```
%===== subroutine get_Jac_indices.m
```

```

function [p,p2,p3,p4,p5,p6,p7] = get_Jac_indices(m,n,k,grid);

index = 3*((m-1)*grid.z + (n-1)) + k;
p = index;
if k == 1
    p2 = index+1;
    p3 = index+2;
elseif k==2
    p2 = index-1;
    p3 = index+1;
else
    p2 = index-2;
    p3 = index-1;
end

p4 = index+3*grid.z;
p5 = index-3*grid.z;
p6 = index+3;
p7 = index-3;

return

%===== subroutine calc_func_P6C5.m
function [f, Jac] = calc_func_P6C5(x0, param, grid)
%function [f] = calc_func_P6C5(x0, param, grid)
f = zeros(length(x0),1);
v = param.vel;
% extract vector into more physical format

c = zeros(grid.y,grid.z,3);

Jac = spalloc(length(x0),length(x0),7*length(x0));

for m = 1:grid.y
    for n = 1:grid.z
        for k = 1:3
            index_c = 3*((m-1)*grid.z + (n-1)) + k;
            c(m,n,k) = x0(index_c);
        end
    end
end

for m = 1:grid.y
    for n = 1:grid.z
        for k = 1:3

            [A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);
            [BC1,BC2,BC3,BC4,BC5] = calc_bc_coeff(m,n,k,param,grid);

            % get function index
            index_c = 3*((m-1)*grid.z + (n-1)) + k;
            % top condition
            if m == 1
                % upper right condition (1 Dirichlet 1 Neumann)

```

```

        if n == 1
            f(index_c) = (A1+BC1)*c(m,n,k) + (A2+BC2)*c(m+1,n,k) + BC3 +
A4*c(m,n+1,k) + ...
                BC5 + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
            % upper left condition (2 Neumann)
        elseif n == grid.z
            f(index_c) = (A1+BC1)*c(m,n,k) + (A2+BC2)*c(m+1,n,k) + BC3 + ...
                (A5+BC5)*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        else
            % purely top condition (Neumann)
            f(index_c) = (A1+BC1)*c(m,n,k) + (A2+BC2)*c(m+1,n,k) + BC3 +
A4*c(m,n+1,k) + ...
                A5*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        end
        % bottom boundary condition
    elseif m == grid.y
        if n == 1
            % lower right condition (1 Dirichlet and 1 Neumann)
            f(index_c) = (A1+BC1)*c(m,n,k) + (A3+BC3)*c(m-1,n,k) + A4*c(m,n+1,k) + ...
                BC5 + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        elseif n == grid.z
            % lower left condition (2 Neumann)
            f(index_c) = (A1+BC1)*c(m,n,k) + (A3+BC3)*c(m-1,n,k) + (A5+BC5)*c(m,n-
1,k) + ...
                rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        else
            % bottom condition (1 Neumann)
            f(index_c) = (A1+BC1)*c(m,n,k) + (A3+BC3)*c(m-1,n,k) + A4*c(m,n+1,k) + ...
                A5*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        end
    else
        % right boundary (Dirichlet)
        if n == 1
            f(index_c) = A1*c(m,n,k) + A2*c(m+1,n,k) + A3*c(m-1,n,k) + ...
                A4*c(m,n+1,k) + BC5 + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
            %left boundary (Neumann)
        elseif n == grid.z
            f(index_c) = (A1+BC1)*c(m,n,k) + A2*c(m+1,n,k) + A3*c(m-1,n,k) + ...
                (A5+BC5)*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        else
            %interior grid points
            f(index_c) = A1*c(m,n,k) + A2*c(m+1,n,k) + A3*c(m-1,n,k) +...
                A4*c(m,n+1,k) + A5*c(m,n-1,k) + rR*(c(m,n,1)*c(m,n,2)-c(m,n,3)/param.K);
        end
    end
end
end
end
end

for m = 1:grid.y
    for n = 1:grid.z
        for k = 1:3
            [p,p2,p3,p4,p5,p6,p7] = get_Jac_indices(m,n,k,grid);

            index_c = 3*((m-1)*grid.z + (n-1)) + k;

```

```

[c1,c2,c3,c4,c5,c6,c7] = get_jac_const(m,n,k,param,grid,c);
if (index_c <= 3*grid.z) | (index_c >= length(x0)-3*grid.z)
    if (index_c <= 3*grid.z)
        if (index_c <= 3)
            Jac(p,p) = c1;
            Jac(p,p2) = c2;
            Jac(p,p3) = c3;
            Jac(p,p4) = c4;
            Jac(p,p6) = c6;
        else
            Jac(p,p) = c1;
            Jac(p,p2) = c2;
            Jac(p,p3) = c3;
            Jac(p,p4) = c4;
            Jac(p,p6) = c6;
            Jac(p,p7) = c7;
        end
    elseif (index_c >= length(x0)-3*grid.z)
        if (index_c >= length(x0) - 3)
            Jac(p,p) = c1;
            Jac(p,p2) = c2;
            Jac(p,p3) = c3;
            Jac(p,p5) = c5;
            Jac(p,p7) = c7;
        else
            Jac(p,p) = c1;
            Jac(p,p2) = c2;
            Jac(p,p3) = c3;
            Jac(p,p5) = c5;
            Jac(p,p6) = c6;
            Jac(p,p7) = c7;
        end
    end
end
else
    Jac(p,p) = c1;
    Jac(p,p2) = c2;
    Jac(p,p3) = c3;
    Jac(p,p4) = c4;
    Jac(p,p5) = c5;
    Jac(p,p6) = c6;
    Jac(p,p7) = c7;
end
end
end
end
end

```

```

spy(Jac)
return

```

```

% ===== subroutine get_jac_const.m

```

```

function [c1,c2,c3,c4,c5,c6,c7] = get_jac_const(m,n,k,param,grid,c)

```

```

[A1,A2,A3,A4,A5,rR] = calc_coeff_P6C5(m,n,k,param,grid);
[BC1,BC2,BC3,BC4,BC5] = calc_bc_coeff(m,n,k,param,grid);

```

```

if k == 1
    c1 = A1 + rR*c(m,n,2);
    c2 = rR*c(m,n,1);
    c3 = -rR/param.K;
elseif k == 2
    c1 = A1 + rR*c(m,n,1);
    c2 = rR*c(m,n,2);
    c3 = -rR/param.K;
elseif k == 3
    c1 = A1 - rR/param.K;
    c2 = rR*c(m,n,2);
    c3 = rR*c(m,n,1);
end

```

```

if m == 1
    if n == 1
        c1 = c1 + BC1;
        c4 = A2 + BC2;
        c5 = 0;
        c6 = A4;
        c7 = 0;
    elseif n == grid.z
        c1 = c1 + BC1;
        c4 = A2 + BC2;
        c5 = 0;
        c6 = 0;
        c7 = A5 + BC5;
    else
        c4 = A2 + BC2;
        c5 = 0;
        c6 = A4;
        c7 = A5;
    end
elseif m == grid.y
    if n == 1
        c1 = c1 + BC1;
        c4 = 0;
        c5 = A3 + BC3;
        c6 = A4;
        c7 = 0;
    elseif n == grid.z
        c1 = c1 + BC1;
        c4 = 0;
        c5 = A3 + BC3;
        c6 = 0;
        c7 = A5 + BC5;
    else
        c1 = c1 + BC1;
        c4 = 0;
        c5 = A3 + BC3;
        c6 = A4;
        c7 = A5;
    end
end
else

```

```
if n == 1
  c4 = A2;
  c5 = A3;
  c6 = A4;
  c7 = 0;
elseif n == grid.z
  c1 = c1 + BC1;
  c4 = A2;
  c5 = A3;
  c6 = 0;
  c7 = A5 + BC5;
else
  c4 = A2;
  c5 = A3;
  c6 = A4;
  c7 = A5;
end
end

return
```