Days 23: A digression on the numerical solution of boundary value problems

Earlier in the course, we spent substantial time learning how to solve initial value problems. As we have seen, both perfectly mixed batch reactors and completely unmixed plug flow reactors can be described mathematically in terms of an initial value problem. The other prototypical reactor type is the perfectly mixed stirred-tank reactor. It can be modeled by a set of algebraic, rather than differential, equations. We briefly discussed the numerical solution of a set of nonlinear algebraic equations by Newton's method in the context of implicit methods for initial value problems. We will discuss this further below in the context of steady-state boundary value problems.

A reactor with partial mixing generally cannot be described by a set of algebraic equations or by a set of differential equations that take the form of an initial value problem. Instead, a partially mixed reactor will usually lead to some sort of boundary value problem. The equations and boundary conditions that you will derive in the first problem of homework 9 for cases where the reactor is neither perfectly mixed nor completely unmixed are boundary value problems. These are sets of differential equations that include derivatives or at least second order and that have boundary conditions specified at more than one location. A boundary value problem that also includes time dependence is an initial boundary-value problem. An example from the previous lecture is the set of general reactor balance equations for constant physical properties

$$\nabla\cdot\underline{v} = 0$$

$$\frac{dC_k}{dt} = -\underline{v}\cdot\nabla C_k + D_k\nabla^2 C_k + \sum_{i=1}^{M}\alpha_{ik} r_i$$

$$\rho\hat{C}_p\left(\frac{dT}{dt} + \underline{v}\cdot\nabla T\right) = \sum_{i=1}^{M}(-\Delta H_i) r_i + \lambda\nabla^2 T + \sum_{i=1}^{N}\left(C_{pk}\left(T - T_{ref}\right)D_k\nabla^2 C_k + C_{pk} D_k \nabla T\cdot\nabla C_k\right)$$

With boundary conditions specified over the reactor walls, inlet, and outlet, and an initial condition specified over the whole reactor volume.

This is an initial-boundary-value problem. It is a set of partial differential equations in up to three spatial dimensions plus the time dimension. We will not learn how to solve such equations in this course. Such equations are probably best solved by starting from a commercial computational fluid dynamics code.

A slightly simpler problem is given by the same set of reactor balance equations at steady state

$$\nabla\cdot\underline{v} = 0$$

$$-\underline{v}\cdot\nabla C_k + D_k\nabla^2 C_k + \sum_{i=1}^{M}\alpha_{ik} r_i = 0$$

$$\rho\hat{C}_p\left(\underline{v}\cdot\nabla T\right) = \sum_{i=1}^{M}(-\Delta H_i) r_i + \lambda\nabla^2 T + \sum_{i=1}^{N}\left(C_{pk}\left(T - T_{ref}\right)D_k\nabla^2 C_k + C_{pk} D_k \nabla T\cdot\nabla C_k\right)$$

With boundary conditions specified over the reactor walls, inlet, and outlet.

This is still a multi-dimensional problem, and we will not learn how to solve these either – although it can be done by straightforward extensions of what we will learn. If you are interested

in learning how to solve such equations, I would recommend the short and simple book by Prof. Suhas Patankar entitled 'Numerical Heat Transfer and Fluid Flow'.

What we will learn how to solve numerically is a one-dimensional boundary value problem. The prototypical example of this is the steady-state plug flow tubular reactor with axial mixing. The governing equations for that situation (assuming constant physical properties) are

$$-v_x \frac{dC_k}{dx} + D_k \frac{d^2C_k}{dx^2} + \sum_{i=1}^{M} \alpha_{ik} r_i = 0 \text{ , } k = 1,N \text{ } (N = \text{number of chemical species})$$

$$-\rho \hat{C}_p v_x \frac{dT}{dx} + \sum_{i=1}^{M} (-\Delta H_i) r_i + \lambda \frac{d^2T}{dx^2} + \sum_{i=1}^{N} \left(C_{pk} (T - T_{ref}) D_k \frac{d^2C_k}{dx^2} + C_{pk} D_k \frac{dT}{dx} \frac{dC_k}{dx} \right)$$

with boundary conditions

$C_k(x = 0) = C_{ko}$, and $T(x = 0) = T_o$

and $$\left.\frac{dC_k}{dx}\right|_{x=L} = \left.\frac{dT}{dx}\right|_{x=L} = 0$$

To obtain an approximate numerical solution to this one-dimensional boundary value problem, we will replace the continuous dependent variables (C_k, and T) with their values at a set of P discrete points that span the length of the reactor ($x = 0$ to $x = L$). Then we will express the derivatives in the equations and boundary conditions with finite difference approximations involving the values at the P discrete points. This procedure is called discretizing the equations. It converts the set of differential equations into a larger set of algebraic equations. We know how to solve the algebraic equations using Newton's method. As described above, we are applying a 'finite difference' method to do the discretization. We could equally well apply a 'finite element' or 'finite volume' method. For a one-dimensional problem they would all come out about the same. For a two or three dimensional problem, the different discretization methods would lead to different sets of algebraic equations.

To illustrate this method, we will consider the simple set of reactions

(1) $A_1 + A_2 \leftrightarrow A_3$ with $r_1 = k_1 C_1 C_2 - k_{-1} C_3$

(2) $A_1 + A_3 \leftrightarrow A_4$ with $r_2 = k_2 C_1 C_3 - k_{-2} C_4$

Taking place in an isothermal plug-flow reactor with axial mixing. The reactor feed will be equal amounts of A_1 and A_2. ($C_1(x = 0) = C_2(x = 0) = C_o$, $C_3(x = 0) = C_4(x = 0) = 0$). Since we are neglecting density changes in the reactor, the axial velocity is constant and we will just call it v. For an isothermal reactor, we do not need to solve the enthalpy balance. Therefore the equations describing this situation are

$$-v \frac{dC_1}{dx} + D_1 \frac{d^2C_1}{dx^2} - k_1 C_1 C_2 + k_{-1} C_3 - k_2 C_1 C_3 + k_{-2} C_4 = 0$$

$$-v \frac{dC_2}{dx} + D_2 \frac{d^2C_2}{dx^2} - k_1 C_1 C_2 + k_{-1} C_3 = 0$$

$$-v \frac{dC_3}{dx} + D_3 \frac{d^2C_3}{dx^2} + k_1 C_1 C_2 - k_{-1} C_3 - k_2 C_1 C_3 + k_{-2} C_4 = 0$$

$$-v\frac{dC_4}{dx} + D_4\frac{d^2C_4}{dx^2} + k_2C_1C_3 - k_{-2}C_4 = 0$$

with boundary conditions

$C_1(x = 0) = C_2(x = 0) = C_o$, $C_3(x = 0) = C_4(x = 0) = 0$

$$\left.\frac{dC_1}{dx}\right|_{x=L} = \left.\frac{dC_2}{dx}\right|_{x=L} = \left.\frac{dC_3}{dx}\right|_{x=L} = \left.\frac{dC_4}{dx}\right|_{x=L} = 0$$

These may or may not be the best (most physically reasonable) boundary conditions. It depends on the exact geometry of the reactor. But, for purposes of illustration, these are what we will use.

To discretize the equations, we specify P points in the reactor at which we will (approximately) solve for the concentrations. The more points we use, the more accurate our solution will be, but the fewer points we use, the faster we will be able to calculate it. Consider the l^{th} point, which has position in the reactor x_l and at which the concentrations are $C_1 = C_{1,l}$, $C_2 = C_{2,l}$, $C_3 = C_{3,l}$, $C_4 = C_{4,l}$. The derivatives appearing in the equations can then be expressed in terms of finite difference approximations involving the concentrations and position at points $l - 1$, l, and $l + 1$. There are two simple approximations that we will consider for approximating the first derivative of the concentrations at point l.

These are a *central difference approximation*, given by

$$\left.\frac{dC_i}{dx}\right|_{x=x_l} \cong \left.\frac{\Delta C}{\Delta x}\right|_{x=x_l} = \frac{C_{i,l+1} - C_{i,l-1}}{x_{l+1} - x_{l-1}}$$

and an *upwind difference approximation*, given by

$$\left.\frac{dC_i}{dx}\right|_{x=x_l} \cong \left.\frac{\Delta C}{\Delta x}\right|_{x=x_l} = \frac{C_{i,l} - C_{i,l-1}}{x_l - x_{l-1}} \text{ when } v > 0$$

and

$$\left.\frac{dC_i}{dx}\right|_{x=x_l} \cong \left.\frac{\Delta C}{\Delta x}\right|_{x=x_l} = \frac{C_{i,l+1} - C_{i,l}}{x_{l+1} - x_l} \text{ when } v < 0$$

The central difference approximation (which involves the values at neighboring points, but not at point l itself) is the more accurate approximation to the derivative when the convective term in the equation is comparable to or smaller than the diffusive term. For situations where the convective term is larger than the diffusive term, the transport is primarily from the upstream to the downstream portion of the reactor. In those cases, the upwind difference approximation, which is biased to include information from the upstream direction of the flow, performs better. Use of the central difference approximation in such situations can cause severe numerical difficulties in obtaining the solution.

The simplest approximation to the 2nd derivative of one of the concentrations at point l is given by applying the central difference approximation to get first derivatives at points halfway between point l and its neighboring points (at $x_{l+1/2} = (x_l + x_{l+1})/2$ and $x_{l-1/2} = (x_l + x_{l-1})/2$), and then applying the central difference approximation again to those derivatives to get the 2nd derivative

$$\left.\frac{dC_i}{dx}\right|_{x=x_{l+1/2}} \cong \frac{C_{i,l+1}-C_{i,l}}{x_{l+1}-x_l}$$

$$\left.\frac{dC_i}{dx}\right|_{x=x_{l-1/2}} \cong \frac{C_{i,l}-C_{i,l-1}}{x_l-x_{l-1}}$$

and

$$\left.\frac{d^2C_i}{dx^2}\right|_{x=x_l} \cong \frac{\left.\frac{dC_i}{dx}\right|_{x=x_{l+1/2}} - \left.\frac{dC_i}{dx}\right|_{x=x_{l-1/2}}}{x_{l+1/2}-x_{l-1/2}} \cong \frac{\frac{C_{i,l+1}-C_{i,l}}{x_{l+1}-x_l} - \frac{C_{i,l}-C_{i,l-1}}{x_l-x_{l-1}}}{\frac{x_{l+1}+x_l}{2} - \frac{x_l+x_{l-1}}{2}}$$

$$\left.\frac{d^2C_i}{dx^2}\right|_{x=x_l} \cong \frac{2\left((x_l-x_{l-1})C_{i,l+1} - (x_{l+1}-x_{l-1})C_{i,l} + (x_{l+1}-x_l)C_{i,l-1}\right)}{(x_l-x_{l-1})(x_{l+1}-x_{l-1})(x_{l+1}-x_l)}$$

If the points are equally spaced so that $x_{l+1} - x_l = x_l - x_{l-1} = h$, then this simplifies to

$$\left.\frac{d^2C_i}{dx^2}\right|_{x=x_l} \cong \frac{C_{i,l+1} - 2C_{i,l} + C_{i,l-1}}{h^2}$$

It is often advantageous to use unequally spaced points – putting more points in locations where there are steep concentration gradients and fewer points in locations where the concentration is not changing much. This is best done via grid refinement – solving the problem repeatedly with larger and larger sets of points. After each solution, more points are added in the regions where they are most needed. The process is repeated until some criterion for the maximum change in concentration between adjacent points is satisfied. An example of a program where this is done is the PREMIX program that is part of the CHEMKIN suite of programs that were originally developed at Sandia National Laboratories and are now sold by Reaction Design, Inc. (**http://www.reactiondesign.com/premixsum.html)**. All that being said, we will complete this example using a uniformly spaced grid with grid spacing h, and will use central differences for the first derivatives, so that we can approximate the derivatives by

$$\left.\frac{dC_i}{dx}\right|_{x=x_l} \cong \frac{C_{i,l+1}-C_{i,l-1}}{2h}, \text{ and } \left.\frac{d^2C_i}{dx^2}\right|_{x=x_l} \cong \frac{C_{i,l+1}-2C_{i,l}+C_{i,l-1}}{h^2}$$

With this discretization scheme, the equations for the concentrations at point l become

$$-v\frac{C_{1,l+1}-C_{1,l-1}}{2h} + D_1\frac{C_{1,l+1}-2C_{1,l}+C_{1,l-1}}{h^2} - k_1C_{1,l}C_{2,l} + k_{-1}C_{3,l} - k_2C_{1,l}C_{3,l} + k_{-2}C_{4,l} = 0$$

$$-v\frac{C_{2,l+1}-C_{2,l-1}}{2h} + D_2\frac{C_{2,l+1}-2C_{2,l}+C_{2,l-1}}{h^2} - k_1C_{1,l}C_{2,l} + k_{-1}C_{3,l} = 0$$

$$-v\frac{C_{3,l+1}-C_{3,l-1}}{2h} + D_3\frac{C_{3,l+1}-2C_{3,l}+C_{3,l-1}}{h^2} + k_1C_{1,l}C_{2,l} - k_{-1}C_{3,l} - k_2C_{1,l}C_{3,l} + k_{-2}C_{4,l} = 0$$

$$-v\frac{C_{4,l+1}-C_{4,l-1}}{2h} + D_4\frac{C_{4,l+1}-2C_{4,l}+C_{4,l-1}}{h^2} + k_2C_{1,l}C_{3,l} - k_{-2}C_{4,l} = 0$$

And we have a total of $P - 2$ of these sets of equations, for $l = 2$ to $P - 1$. At the end points, we apply the boundary conditions. At the first point ($l = 1$, $x_l = 0$) the boundary conditions are

$$C_{1,1} = C_{2,1} = C_o,\ C_{3,1} = C_{4,1} = 0$$

At the other boundary, we can approximate the first derivatives (which are set equal to zero in the boundary condition) by a finite difference using points $P-1$, and P. That gives

$$\frac{C_{1,P}-C_{1,P-1}}{h}=\frac{C_{2,P}-C_{2,P-1}}{h}=\frac{C_{3,P}-C_{3,P-1}}{h}=\frac{C_{4,P}-C_{4,P-1}}{h}=0$$

or more simply

$$C_{1,P}=C_{1,P-1},\quad C_{2,P}=C_{2,P-1},\quad C_{3,P}=C_{3,P-1},\quad C_{4,P}=C_{4,P-1}$$

Including these equations at the boundaries, we now have a total of $4P$ algebraic equations in $4P$ unknowns (4 concentrations at each of P points). The equations are non-linear due to non-linear due to the second-order terms in the reaction rate terms. As we discussed earlier in the quarter, we can solve a set of non-linear algebraic equations using Newton's method. In preparation for doing so, we will arrange the unknowns in a vector (which we will call $\underline{y}$) as follows

$$\underline{y}=\left[C_{1,1}\quad C_{2,1}\quad C_{3,1}\quad C_{4,1}\quad C_{1,2}\quad C_{2,2}\quad \ldots\quad \ldots\quad C_{P,1}\quad C_{P,2}\quad C_{P,3}\quad C_{P,4}\right]^T$$

where the superscript T indicates the transpose of the vector (so it is a column vector, which would take a lot of space to write). The corresponding vector of algebraic equations can be written as $\underline{f}(\underline{y})=\underline{0}$, where $\underline{f}(\underline{y})$ is as shown on the following page.

$$\underline{f} = \begin{bmatrix} C_{1,1} - C_o \\ C_{2,1} - C_o \\ C_{3,1} \\ C_{4,1} \\ -v\frac{C_{1,3} - C_{1,1}}{2h} + D_1\frac{C_{1,3} - 2C_{1,2} + C_{1,1}}{h^2} - k_1 C_{1,2} C_{2,2} + k_{-1} C_{3,2} - k_2 C_{1,2} C_{3,2} + k_{-2} C_{4,2} \\ -v\frac{C_{2,3} - C_{2,1}}{2h} + D_2\frac{C_{2,3} - 2C_{2,2} + C_{2,1}}{h^2} - k_1 C_{1,2} C_{2,2} + k_{-1} C_{3,2} \\ -v\frac{C_{3,3} - C_{3,1}}{2h} + D_3\frac{C_{3,3} - 2C_{3,2} + C_{3,1}}{h^2} + k_1 C_{1,2} C_{2,2} - k_{-1} C_{3,2} - k_2 C_{1,2} C_{3,2} + k_{-2} C_{4,2} \\ -v\frac{C_{4,3} - C_{4,1}}{2h} + D_4\frac{C_{4,3} - 2C_{4,2} + C_{4,1}}{h^2} + k_2 C_{1,2} C_{3,2} - k_{-2} C_{4,2} \\ \cdots \\ \cdots \\ -v\frac{C_{1,P} - C_{1,P-2}}{2h} + D_1\frac{C_{1,P} - 2C_{1,P-1} + C_{1,P-2}}{h^2} - k_1 C_{1,P-1} C_{2,P-1} + k_{-1} C_{3,P-1} - k_2 C_{1,P-1} C_{3,P-1} + k_{-2} C_{4,P-1} \\ -v\frac{C_{2,P} - C_{2,P-2}}{2h} + D_2\frac{C_{2,P} - 2C_{2,P-1} + C_{2,P-2}}{h^2} - k_1 C_{1,P-1} C_{2,P-1} + k_{-1} C_{3,P-1} \\ -v\frac{C_{3,P} - C_{3,P-2}}{2h} + D_3\frac{C_{3,P} - 2C_{3,P-1} + C_{3,P-2}}{h^2} + k_1 C_{1,P-1} C_{2,P-1} - k_{-1} C_{3,P-1} - k_2 C_{1,P-1} C_{3,P-1} + k_{-2} C_{4,P-1} \\ -v\frac{C_{4,P} - C_{4,P-2}}{2h} + D_4\frac{C_{4,P} - 2C_{4,P-1} + C_{4,1}}{h^2} + k_2 C_{1,P-1} C_{3,P-1} - k_{-2} C_{4,P-1} \\ C_{1,P} - C_{1,P-1} \\ C_{2,P} - C_{2,P-1} \\ C_{3,P} - C_{3,P-1} \\ C_{4,P} - C_{4,P-1} \end{bmatrix}$$

For purposes of solving the equations, it is useful to re-write this in terms of the components of $\underline{y}$, which are what we are solving for. Doing so gives

$$\underline{f}(\underline{y}) = \begin{bmatrix} y_1 - C_o \\ y_2 - C_o \\ y_3 \\ y_4 \\ -v\frac{y_9 - y_1}{2h} + D_1\frac{y_9 - 2y_5 + y_1}{h^2} - k_1 y_5 y_6 + k_{-1} y_7 - k_2 y_5 y_7 + k_{-2} y_8 \\ -v\frac{y_{10} - y_2}{2h} + D_2\frac{y_{10} - 2y_6 + y_2}{h^2} - k_1 y_5 y_6 + k_{-1} y_7 \\ -v\frac{y_{11} - y_3}{2h} + D_3\frac{y_{11} - 2y_7 + y_3}{h^2} + k_1 y_5 y_6 - k_{-1} y_7 - k_2 y_5 y_7 + k_{-2} y_8 \\ -v\frac{y_{12} - y_4}{2h} + D_4\frac{y_{12} - 2y_8 + y_4}{h^2} + k_2 y_5 y_7 - k_{-2} y_8 \\ .... \\ .... \\ -v\frac{y_{4P-3} - y_{4P-11}}{2h} + D_1\frac{y_{4P-3} - 2y_{4P-7} + y_{4P-11}}{h^2} - k_1 y_{4P-7} y_{4P-6} + k_{-1} y_{4P-5} - k_2 y_{4P-7} y_{4P-5} + k_{-2} y_{4P-4} \\ -v\frac{y_{4P-2} - y_{4P-10}}{2h} + D_2\frac{y_{4P-2} - 2y_{4P-6} + y_{4P-10}}{h^2} - k_1 y_{4P-7} y_{4P-6} + k_{-1} y_{4P-5} \\ -v\frac{y_{4P-1} - y_{4P-9}}{2h} + D_3\frac{y_{4P-1} - 2y_{4P-5} + y_{4P-9}}{h^2} + k_1 y_{4P-7} y_{4P-6} - k_{-1} y_{4P-5} - k_2 y_{4P-7} y_{4P-5} + k_{-2} y_{4P-4} \\ -v\frac{y_{4P} - y_{4P-8}}{2h} + D_4\frac{y_{4P} - 2y_{4P-4} + y_{4P-8}}{h^2} + k_2 y_{4P-7} y_{4P-5} - k_{-2} y_{4P-4} \\ y_{4P-3} - y_{4P-7} \\ y_{4P-2} - y_{4P-6} \\ y_{4P-1} - y_{4P-5} \\ y_{4P} - y_{4P-4} \end{bmatrix}$$

Now the problem is formulated so that it can be solved efficiently using Newton's method. (Also known as the Newton-Raphson method, for all of you Raphson fans out there). As I am sure you remember, the algorithm is basically as follows

Set $\underline{y}$ = some initial guess
Do for j from 1 to some maximum allowable number of iterations
    Solve the linear matrix equation $\underline{\underline{J}}(\Delta\underline{y}) = -\underline{f}(\underline{y})$ for $\Delta\underline{y}$
    Set $\underline{y} = \underline{y} + \Delta\underline{y}$
    If some measure of convergence (like the norm of $\Delta\underline{y}$) is small enough, stop
End do loop

In this algorithm, the matrix $\underline{\underline{J}}$ is the Jacobian of the system of equations. The elements of $\underline{\underline{J}}$ are defined by $J_{i,j} = \frac{\partial f_i}{\partial y_j}$. For this example problem, the Jacobian matrix has the structure

$$\underline{\underline{J}} = \begin{bmatrix}
\underline{\underline{M}}_{1,1} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} \\
\underline{\underline{M}}_{2,1} & \underline{\underline{M}}_{2,2} & \underline{\underline{M}}_{1,3} & \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} \\
\underline{\underline{0}} & \underline{\underline{M}}_{3,2} & \underline{\underline{M}}_{3,3} & \underline{\underline{M}}_{3,4} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} \\
\underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{M}}_{4,3} & \underline{\underline{M}}_{4,4} & \underline{\underline{M}}_{4,5} & \cdots & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \underline{\underline{0}} & \underline{\underline{0}} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{M}}_{4P-3,4P-4} & \underline{\underline{M}}_{4P-3,4P-3} & \underline{\underline{M}}_{4P-3,4P-2} & \underline{\underline{0}} & \underline{\underline{0}} \\
\underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{M}}_{4P-2,4P-3} & \underline{\underline{M}}_{4P-2,4P-2} & \underline{\underline{M}}_{4P-2,4P-1} & \underline{\underline{0}} \\
\underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{M}}_{4P-1,4P-2} & \underline{\underline{M}}_{4P-1,4P-1} & \underline{\underline{M}}_{4P-1,4P} \\
\underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{0}} & \underline{\underline{M}}_{4P,4P-1} & \underline{\underline{M}}_{4P,4P}
\end{bmatrix}$$

where each of the sub-matrices $\underline{\underline{M}}_{i,j}$ is a 4x4 matrix of the appropriate derivatives. Some of these sub-matrices are

$$\underline{\underline{M}}_{1,1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \underline{\underline{M}}_{2,1} = \begin{bmatrix} \frac{v}{2h} + \frac{D_1}{h^2} & 0 & 0 & 0 \\ 0 & \frac{-v}{2h} + \frac{D_2}{h^2} & 0 & 0 \\ 0 & 0 & \frac{-v}{2h} + \frac{D_3}{h^2} & 0 \\ 0 & 0 & 0 & \frac{-v}{2h} + \frac{D_4}{h^2} \end{bmatrix}$$

$$\underline{\underline{M}}_{2,2} = \begin{bmatrix} \frac{-2D_1}{h^2} - k_1 y_6 - k_2 y_7 & -k_1 y_5 & -k_2 y_5 + k_{-1} & k_{-2} \\ -k_1 y_6 & \frac{-2D_2}{h^2} - k_1 y_5 & k_{-1} & 0 \\ k_1 y_6 - k_2 y_7 & k_1 y_5 & \frac{-2D_3}{h^2} - k_2 y_5 - k_{-1} & k_{-2} \\ k_2 y_7 & 0 & k_2 y_5 & \frac{-2D_4}{h^2} - k_{-2} \end{bmatrix}$$

$$\underline{\underline{M}}_{2,3} = \begin{bmatrix} \frac{-v}{2h} + \frac{D_1}{h^2} & 0 & 0 & 0 \\ 0 & \frac{-v}{2h} + \frac{D_2}{h^2} & 0 & 0 \\ 0 & 0 & \frac{-v}{2h} + \frac{D_3}{h^2} & 0 \\ 0 & 0 & 0 & \frac{-v}{2h} + \frac{D_4}{h^2} \end{bmatrix}$$

We see a pattern for the three sub-matrices in a given row, so we can say that for l=2 to l=P-1 the submatrices are

$$\underline{\underline{M}}_{l,l-1} = \begin{bmatrix} \frac{v}{2h}+\frac{D_1}{h^2} & 0 & 0 & 0 \\ 0 & \frac{v}{2h}+\frac{D_2}{h^2} & 0 & 0 \\ 0 & 0 & \frac{v}{2h}+\frac{D_3}{h^2} & 0 \\ 0 & 0 & 0 & \frac{v}{2h}+\frac{D_4}{h^2} \end{bmatrix}$$

$$\underline{\underline{M}}_{l,l} = \begin{bmatrix} \frac{-2D_1}{h^2}-k_1 y_{4l-2}-k_2 y_{4l-1} & -k_1 y_{4l-3} & -k_2 y_{4l-3}+k_{-1} & k_{-2} \\ -k_1 y_{4l-2} & \frac{-2D_2}{h^2}-k_1 y_{4l-3} & k_{-1} & 0 \\ k_1 y_{4l-2}-k_2 y_{4l-1} & k_1 y_{4l-3} & \frac{-2D_3}{h^2}-k_2 y_{4l-3}-k_{-1} & k_{-2} \\ k_2 y_{4l-1} & 0 & k_2 y_{4l-3} & \frac{-2D_4}{h^2}-k_{-2} \end{bmatrix}$$

$$\underline{\underline{M}}_{2,3} = \begin{bmatrix} \frac{-v}{2h}+\frac{D_1}{h^2} & 0 & 0 & 0 \\ 0 & \frac{-v}{2h}+\frac{D_2}{h^2} & 0 & 0 \\ 0 & 0 & \frac{-v}{2h}+\frac{D_3}{h^2} & 0 \\ 0 & 0 & 0 & \frac{-v}{2h}+\frac{D_4}{h^2} \end{bmatrix}$$

Finally, the last submatrices (for l=P) are

$$\underline{\underline{M}}_{P,P-1} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad \underline{\underline{M}}_{P,P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we have all of the information that we need to implement Newton's method to solve the equations. We notice that the Jacobian matrix is mostly zeros and has a *banded* structure. The *bandwidth* of the Jacobian is 9. That means that all matrix elements except for the 9 elements on each row that are closest to the diagonal are zero. The only non-zero elements are those on the diagonal and four on each side of the diagonal. This means that in a computer program, we can store just 9 elements of each row instead of the whole thing. If we had 100 grid points, we would have 400 equations, so $\underline{\underline{J}}$ would be a 400 by 400 matrix. Storing all of the elements of $\underline{\underline{J}}$ would require $(400)^2$ = 160,000 words of memory. Storing just 9 elements from each row requires only 9*400 = 3600 words of memory (a factor of 44 less memory). There number of operations required to solve the matrix equation $\underline{\underline{J}}\left(\Delta \underline{y}\right) = -\underline{f}\left(\underline{y}\right)$ is reduced by the square of this factor. The number of operations required to solve a matrix equation with a full matrix (all non-

zero elements) is proportional to n^3, where n is the number of rows (or columns) in the matrix. For a banded matrix, the number of operations is proportional to $n*(bw)^2$, where bw is the bandwidth. Many matrix equation solvers are structured so that they can take advantage of the banded structure of the matrix. Therefore, it is important that we arrange the matrix elements as we have done so that the bandwidth is minimized and it is important that we provide information about the bandwidth to the solver.

To illustrate the implementation of the method, I have attached below a Matlab program written to solve the particular problem that we have been using as an example here. It looks long, but that is mostly because I have tried to put in a lot of comments. With 100 grid points, it runs in less than 1 minute. The sparse matrix capabilities of matlab are used to store only the non-zero matrix elements.

```
function sol=day23(Co,k1,km1,k2,km2,v,D1,D2,D3,D4,L,np)
% function day23(Co,k1,km1,k2,km2,v,D1,D2,D3,D4,L,np)
% Co = initial concentration of species 1 and 2 (mol/cm3)
% k1, km1, k2, km2 = rate constants (mol, cm, s units)
% v = velocity (cm/s)
% D1, D2, D3, D4 = diffusion coefficients (cm2/s)
% L = reactor length (cm)
% np = total number of grid points
%
% the program solves the example set of equations from the
% 'day 23' lecture on finite difference methods
%
% Set a couple parameters
itmax=10;      % maximum number of iterations
errcrit=1.e-6;  % error criterion
%
% put dummy values in y, f, and deltay so that they are
% interpreted as column vectors throughout the program.
%
deltay=ones(4*np,1);
f=ones(4*np,1);
y=ones(4*np,1);
%
% Fill in an initial guess for the dependent variables
% y is the vector of dependent variables (concentrations),
% with interlaced variables as in the course notes, so that y1
% is C1 at node 1, y2 is C2 at node 1, etc.
% For the initial guess, we'll use the inlet composition
%
for i=1:np
  y(4*i-3)=Co;
  y(4*i-2)=Co;
  y(4*i-1)=0;
  y(4*i)=0;
end
%
% compute the grid spacing
h=L/(np-1);
%
% compute the Jacobian elements that don't change from iteration
% to iteration
```

```
e1=v/2/h+D1/h/h;
e2=v/2/h+D2/h/h;
e3=v/2/h+D3/h/h;
e4=v/2/h+D4/h/h;
e5=-2*D1/h/h;
e6=-2*D2/h/h;
e7=-2*D3/h/h;
e8=-2*D4/h/h;
e9=-v/2/h+D1/h/h;
e10=-v/2/h+D2/h/h;
e11=-v/2/h+D3/h/h;
e12=-v/2/h+D4/h/h;
%
% Now we will begin the loop over the iterations of Newton's method.
%
% fix maximum number of iterations at 100
for i=1:itmax
% evaluate the function of which we're finding a root
  f(1)=y(1)-Co;
  f(2)=y(2)-Co;
  f(3)=y(3);
  f(4)=y(4);
  for l=2:np-1;
    f(4*l-3)=-v/2/h*(y(4*(l+1)-3)-y(4*(l-1)-3));
    f(4*l-3)=f(4*l-3)+D1/h/h*(y(4*(l+1)-3)-2*y(4*l-3)+y(4*(l-1)-3));
    f(4*l-3)=f(4*l-3)-k1*y(4*l-3)*y(4*l-2)+km1*y(4*l-1);
    f(4*l-3)=f(4*l-3)-k2*y(4*l-3)*y(4*l-1)+km2*y(4*l);
    f(4*l-2)=-v/2/h*(y(4*(l+1)-2)-y(4*(l-1)-2));
    f(4*l-2)=f(4*l-2)+D2/h/h*(y(4*(l+1)-2)-2*y(4*l-2)+y(4*(l-1)-2));
    f(4*l-2)=f(4*l-2)-k1*y(4*l-3)*y(4*l-2)+km1*y(4*l-1);
    f(4*l-1)=-v/2/h*(y(4*(l+1)-1)-y(4*(l-1)-1));
    f(4*l-1)=f(4*l-1)+D3/h/h*(y(4*(l+1)-1)-2*y(4*l-1)+y(4*(l-1)-1));
    f(4*l-1)=f(4*l-1)+k1*y(4*l-3)*y(4*l-2)-km1*y(4*l-1);
    f(4*l-1)=f(4*l-1)-k2*y(4*l-3)*y(4*l-1)+km2*y(4*l);
    f(4*l)=-v/2/h*(y(4*(l+1))-y(4*(l-1)));
    f(4*l)=f(4*l)+D4/h/h*(y(4*(l+1))-2*y(4*l)+y(4*(l-1)));
    f(4*l)=f(4*l)+k2*y(4*l-3)*y(4*l-1)-km2*y(4*l);
  end
  f(4*np-3)=y(4*np-3)-y(4*np-7);
  f(4*np-2)=y(4*np-2)-y(4*np-6);
  f(4*np-1)=y(4*np-1)-y(4*np-5);
  f(4*np)  =y(4*np)  -y(4*np-4);
%
% Now we check for convergence of Newton's method.
% The convergence criterion assures that both the function and the change
% in the solution from iteration to iteration are small.  They are weighted
% to have the same units (and therefore be of the same order of magnitude).
% Since f has units of reaction rate, dividing f by the largest
% first-order rate constant gives it units of concentrations (the same units
% as delta y).  To make the convergence measure dimensionless, we divide by
the
% norm of y (the concentration vector).
%
  disp(['Norm(f) = ',num2str(norm(f))])
  disp(['Norm(deltay) = ',num2str(norm(deltay))])
  errmeas=(norm(f)/max([k1 k2]')+norm(deltay))/norm(y);
  disp(['Error Measure = ',num2str(errmeas)])
```

```
  if (errmeas<errcrit)
    disp(['Converged!'])
    x=0:h:L;
    for j=1:np;
      sol(j,1)=y(j*4-3);
      sol(j,2)=y(j*4-2);
      sol(j,3)=y(j*4-1);
      sol(j,4)=y(j*4);
    end
    plot(x,sol)
    return
  end
%
% Now we must build the Jacobian matrix.  The elements that don't change can
% just be filled in using the values calculated above (e1,e2,etc.).  The
others
% will be calculated.
%
% I'll put the elements into a form that will eventually let me make them
% into a sparse matrix.  Jsparse is a three column matrix in which the
% three columns are a row index, a column index, and a value to
% be stored in the matrix.
%
% I'll re-build the whole thing each time through this loop, so I don't have
% to keep track of the rows.  I start with an empty matrix, and just keep
adding
% to the bottom of it.
%
% first the M(1,1) submatrix
  Jsparse=[];
  Jsparse(1,:)=[1 1 1];
  Jsparse=[Jsparse;2 2 1];
  Jsparse=[Jsparse;3,3,1];
  Jsparse=[Jsparse;4,4,1];
%
% Now, we loop over the internal grid points
  for l=2:np-1
%
%   First the submatrix M(l,l-1), which is diagonal, and for which we
%   have already calculated the elements
    Jsparse=[Jsparse;4*l-3 4*l-7 e1];
    Jsparse=[Jsparse;4*l-2 4*l-6 e2];
    Jsparse=[Jsparse;4*l-1 4*l-5 e3];
    Jsparse=[Jsparse;4*l   4*l-4 e4];
%
%   Now the submatrix M(l,l), which is nominally full, and whose elements
%   depend on the reaction rates and therefore on the concentrations.  Refer
%   to the course notes for the formulae
    Jsparse=[Jsparse;4*l-3 4*l-3 e5-k1*y(4*l-2)-k2*y(4*l-1)];
    Jsparse=[Jsparse;4*l-3 4*l-2 -k1*y(4*l-3)];
    Jsparse=[Jsparse;4*l-3 4*l-1 -k2*y(4*l-3)+km1];
    Jsparse=[Jsparse;4*l-3 4*l   km2];
    Jsparse=[Jsparse;4*l-2 4*l-3 -k1*y(4*l-2)];
    Jsparse=[Jsparse;4*l-2 4*l-2 e6-k1*y(4*l-3)];
    Jsparse=[Jsparse;4*l-2 4*l-1 km1];
    Jsparse=[Jsparse;4*l-1 4*l-3 k1*y(4*l-2)-k2*y(4*l-1)];
    Jsparse=[Jsparse;4*l-1 4*l-2 k1*y(4*l-3)];
```

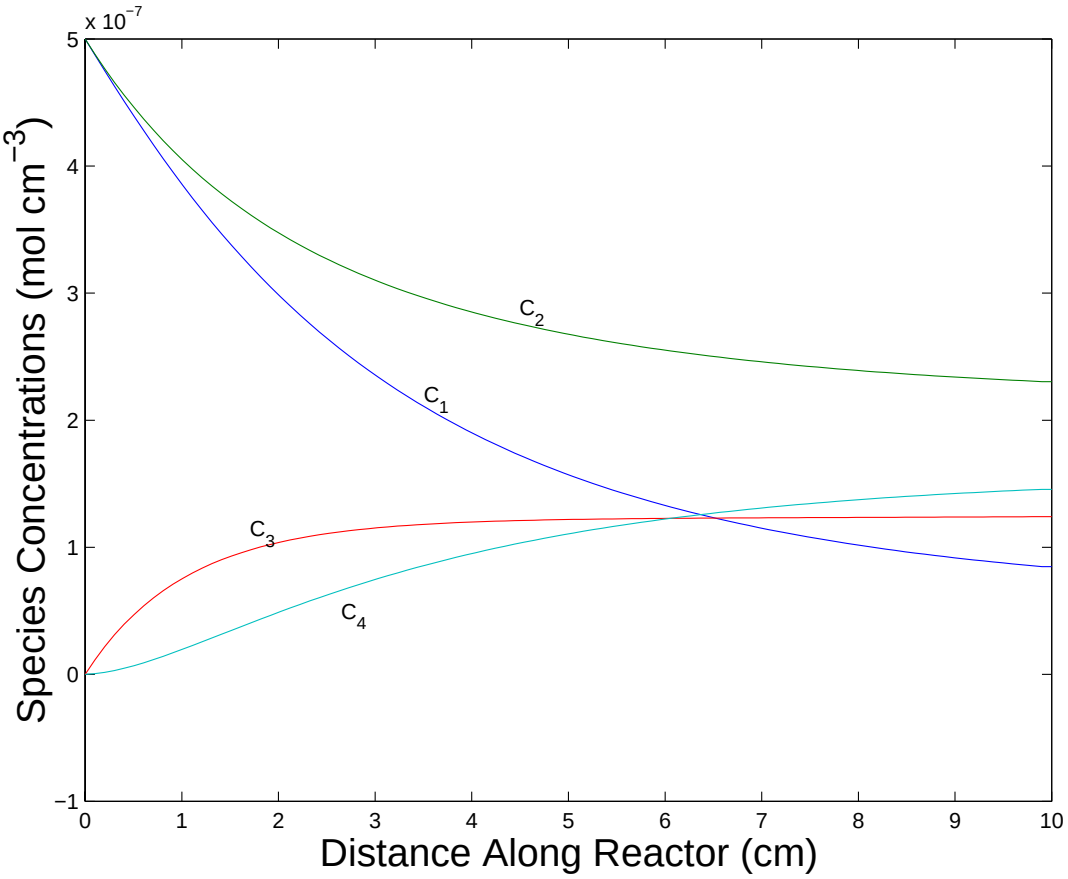
```
    Jsparse=[Jsparse;4*l-1 4*l-1 e7-k2*y(4*l-3)-km1];
    Jsparse=[Jsparse;4*l-1 4*l   km2];
    Jsparse=[Jsparse;4*l   4*l-3 k2*y(4*l-1)];
    Jsparse=[Jsparse;4*l   4*l-1 k2*y(4*l-3)];
    Jsparse=[Jsparse;4*l   4*l   e8-km2];
%
%   Now the submatrix M(l,l+1), which is diagonal, and for which we
%   have already calculated the elements
    Jsparse=[Jsparse;4*l-3 4*l+1 e9];
    Jsparse=[Jsparse;4*l-2 4*l+2 e10];
    Jsparse=[Jsparse;4*l-1 4*l+3 e11];
    Jsparse=[Jsparse;4*l   4*l+4 e12];
  end
%
% Finally, we fill in the last two submatrices, M(np,np-1) and M(np,np)
%
  Jsparse=[Jsparse;4*np-3 4*np-7 -1];
  Jsparse=[Jsparse;4*np-2 4*np-6 -1];
  Jsparse=[Jsparse;4*np-1 4*np-5 -1];
  Jsparse=[Jsparse;4*np   4*np-4 -1];
  Jsparse=[Jsparse;4*np-3 4*np-3  1];
  Jsparse=[Jsparse;4*np-2 4*np-2  1];
  Jsparse=[Jsparse;4*np-1 4*np-1  1];
  Jsparse=[Jsparse;4*np   4*np    1];
%
% Now, we make the jacobian into a matlab sparse matrix
  Jac=sparse(Jsparse(:,1),Jsparse(:,2),Jsparse(:,3),4*np,4*np);
%
% Now we solve J*(delta y) = -f
  deltay=-Jac\f;
% and add this change to y
  y=y+deltay;
% Now we return to the top of the loop.
end
%
% If we get to this point, the max. number of iterations was completed
% without converging.
disp(['Maximum number of iterations exceeded'])
disp(['Convergence criterion was not satisfied'])
sol=[];
return
```

Running this program with the input parameters (just arbitrarily made up)

$C_o = 5\times10^{-7}$ mol cm^{-3}; $k_1 = 5\times10^{6}$ mol cm^{-3} s^{-1}; $k_{-1} = 0.5$ s^{-1}; $k_2 = 1\times10^{7}$ mol cm^{-3} s^{-1}; $k_{-2} = = 0.5$ s^{-1}; $D_1 = D_2 = D_3 = D_4 = 0.5$ cm^2 s^{-1}; $L = 10$ cm; and $v = 1$ cm/s

Gave convergence after 5 Newton iterations for 100 grid points, and resulted in the computed concentration profiles shown below.

You may want to copy this program and use it as a starting point for the 2nd homework problem, which is obviously closely related.


x 10^-7
5
4
3
2
1
0
−1
0
1
2
3
4
5
6
7
8
9
10
Species Concentrations (mol cm^-3)
Distance Along Reactor (cm)
C1
C2
C3
C4