D-46-2
Numerical Solution of Partial Differential Equations
Winter 1997
Hermann Riecke

Project 2 Dynamics of a Chemical Model

Deadline: March 20, 1997.

1. The Code and Preliminary Tests
Write a code which solves two coupled reaction-diffusion models of the form

$$\partial_t u = D_1 \partial_x^2 u + f(u,v), \tag{1}$$
$$\partial_t v = D_2 \partial_x^2 v + g(u,v), \tag{2}$$

on a domain $[0, L]$[1] with fixed-value boundary conditions

$$u(0,t) = \gamma_{10},\ \ u(L,t) = \gamma_{1L},\ \ v(0,t) = \gamma_{20},\ \ v(L,t) = \gamma_{2L}, \tag{3}$$

where the γ are constants. Use the Chebyshev pseudospectral method and calculate the derivatives using the matrix-multiply approach. Do time-stepping in a semi-implicit way, i.e. Crank-Nicholson on the diffusion term (introduce α to be able to run backward Euler as well (cf. below)) and 2^{nd}-order Adams-Bashforth on the remaining terms.
To calculate the relevant matrices write a subroutine which performs the following steps:

1. calculates D using *setd* (to be called with the total number of points involved including the (known) boundary points)
2. calculates D^2
3. calculates the matrix C_r appearing on the r.h.s of the equation (involving the present values of $\partial_x^2 u$). This uses the modified matrix $D_2^{0,N}$.
4. calculates the matrix C_l appearing on the l.h.s of the equation (involving the new (unknown) values of $\partial_x^2 u$). This uses again the modified matrix $D_2^{0,N}$.
5. calculates the inverse of C_l using Gaussian elimination (sample program with calling test routine by e-mail). Note, that the dimension of C_l is $(nx-2)\times(nx-2)$ if there are a total of nx grid points.
6. calculates the vectors D_{i0}^2 and D_{iN}^2 which multiply the boundary conditions on the r.h.s.

Since the matrix to be inverted depends on N, dt, α, D_i etc.[2], it has to be called whenever one of these parameters is changed. Therefore it is most efficient to include these parameters also in the definitions of C_r as well as of $D_{i0,N}^2$.
For starters, set $f = 0$ and $g = 0$ and test the diffusion part in the following way (again no detailed analysis required, just an order of magnitude check):

[1] Don't forget to rescale the diffusion coefficients to reflect the change of coordinates
[2] α is the parameter arising in Crank-Nicholson ($\alpha = 0.5$ for CN and $\alpha = 1$ for BE)

1. Choose $L = 10$, $N = 8$ or $N = 16$ and use $sin(kx\pi/L)$ with $k = 1$ for u and $k = 2$ for v as initial conditions. Evolve the solution in time and check whether it converges to a solution which satisfies the boundary conditions $u(0,t) = 1$, $u(L,t) = 0$, $v(0,t) = 0$, $v(L,t) = 1$.

2. Choose the same initial conditions now with boundary conditions $u(0,t) = 0$, $u(L,t) = 0$, $v(0,t) = 0$, $v(L,t) = 0$. Making an exponential ansatz check that the time for the maximal amplitude to decay to half its value is given by $t_{1/2} = L^2 \ln 2/(\pi^2 k^2 D_i)$ for each of the concentrations. Let the solution evolve for that time and compare with this analytical result.

To be flexible introduce functions $f(u,v)$ and $g(u,v)$ which will be called in the Adams-Bashforth part with old and present values of u and v.
Structure your program in a way that allows you to run it up to some time and then change some parameters and *resume* execution with the *previous* state as initial condition.

2. The Brusselator
A very well-studied model for a tri-molecular reaction is the so-called 'Brusselator', which was introduced by Prigogine and Lefever in 1986. Its reactions lead to the following functions f and g (cf. 'Self-Organization in Nonequilibrium Systems' by G. Nicolis and I. Prigogine; appendix in 'Chemical Oscillations, Waves and Turbulence' by Y. Kuramoto),

$$
\begin{aligned}
f(u,v) &= A - (B+1)u + u^2 v, &\quad (4)\\
g(u,v) &= Bu - u^2 v. &\quad (5)
\end{aligned}
$$

In this project you will investigate this model in two regimes.

3. Steady Spatial Structures
Choose $D_1 = 0.5$, $D_2 = 1$ and $A = 3$ with boundary conditions $\gamma_{1,2;0,L} = 3$.

1. Choose $L = 30$, $N = 64$ and $dt = 0.1$. Take as initial conditions $u = 3$ and $v = 3$, and evolve it to a steady state for $B = 8$. It should be flat in the center.
   Increase B in steps up to $B = 10$. For each value of B evolve the solution up to $\Delta T = 30$([3]). What happens when $B > 9$? You'll find that $N = 32$ is not sufficient to resolve the arising structure.

2. Choose $L = 10$ and $B = 9.5$. Use as initial conditions $u = 3$ and $v = 3$, and evolve it until (u,v) reach a steady state ($t \geq 30$). Increase N and decrease dt (starting from $dt = 0.2$) until you reach graphical resolution of the structure (plot u). Note: to compare solutions with different N graphically, you have to take into account that the grid is not equidistant, i.e. you have to calculate x_i for the output file. Also, to correctly assess the resolution you should compare the solutions only at calculated grid points. The values between grid points which you get when plotting lines are less accurate since they usually are obtained by *linear* interpolation between the grid points.

These steady structures are called Turing structures after A. Turing, who suggested this mechanism as the origin of biological structures (e.g. the segmented tail of a yellow jacket or the

[3] Note again, that you take the previous result as initial condition so the total time actually accumulates by ΔT for each step in B

stripes on a zebra ...) (Phil. Trans. Roy. Soc. Lond. B237 (1952) 5). The concentration u would be expected to determine whether the fur is black or white, say, at a given location.

4. Temporal Structures

Now choose $L = 30$, $N = 32$, $D_1 = 5$, $D_2 = 0.1$, and $A = 3$ with boundary conditions $\gamma_{1,2;0} = 2$ and $\gamma_{1,2;L} = 1.2$. Add a line of code in your program that prints the value of u at $i = 18$ along with the time to a file[4]. Start again with initial conditions $u = 3$ and $v = 3$. As above, increase B, starting with $B = 7$ up to $B = 10$, in small steps keeping B fixed for a time of 50. For each increase in B use the final state obtained for the previous value for B as new initial condition, i.e. just keep going without resetting u and v after you have changed B. What happens around $B = 8$, around $B = 9$ and around $B = 10$? For $B = 10$ let the code run for a longer time in order to get an impression of the dynamics. Plot the time sequences $u(18, t)$ for each value of B into a single graph. Compare and discuss the results for the different values of B. Show the longer time series for $B = 10$ in a separate plot. Can you characterize the temporal behavior in a simple way?

Note: for larger values of B you might have to choose a small $dt \approx 0.01$ to keep the code stable.

If you want to see interesting effects arising from the interaction of the spatial and temporal regime: J.-J. Perraud et al., Phys. Rev. Lett. 71 (1993) 1272.

5. Convergence of the Time-Dependent Solution

In **4.** you saw that the propagation of a front is an important part of the dynamics in this regime. Investigate the convergence behavior of your code for this solution. Note that there is no exact solution with which one could compare the numerical result!

Choose a shorter system $L = 15$ and $D_1 = 5$, $D_2 = 1$, $A = 3$, and $B = 10$ with boundary conditions $\gamma_{1,2;0} = 2$ and $\gamma_{1,2;L} = 1.2$. Run the code up to $t = 3.1$. Use as initial condition a linear function connecting the boundary conditions. Print $u(x = L/2, t = 3.1)$ to a file and use it as a measure for the solution. Make sure that the final time is really $t = 3.1$ and *does not vary* depending on dt! Similarly, make sure that the measured value corresponds to the same physical location independent of N!

For $N = 32$ increase the number of time steps nt from 250 in factors of 2 up to 8000 (i.e. $dt = 3.1/nt$). To discuss the convergence assume the error scales like dt^β, i.e.

$$|u_{num}(L/2, t = 3.1) - u_{dt=0}(L/2, t = 3.1)| = \alpha * dt^\beta. \tag{6}$$

The difference $u_{dt=0\infty}(N/2, t = 3.1) - u_{num}(N/2, t = 3.1)$ should lead to a straight line in a double-logarithmical plot. Find $u_{dt=0}(L/2, t = 3.1)$ by trial and error using *xmgr*, i.e. try to find a value which when subtracted from u_{num} gives the best straight line in a double-log plot for the remainder (in $edit/transformations/evaluate - expressions$ you can transform the data easily by typing, for instance, y=y+2, to invoke a shift by 2). Extrapolate your result to $dt = 0$. Show the log-log plot of the shifted data.

Then investigate the convergence in N for which you expect a dependence of the error of the form

$$|u_{num}(L/2, t = 3.1) - u_{N\to\infty}| = \gamma * e^{-\delta N}. \tag{7}$$

[4] Add a line *call flush*(11) after your write statement *write*(11, ... (assuming you are writing to unit 11) which writes the number right away instead of waiting until the buffer is full. This allows you to look at the time sequence with *xmgr* without quitting the code. The HP-fortran seems not to provide *flush*. You can use the routine below which was written by David Raitt.

Since the location of the grid points changes with changing N make sure that you trace the same physical location for $u(L/2, t)$. Use $nt = 2000$ and increase N from 64 up to 256 or 384 in steps of 64. Try to fit the error of your results to an exponential using a lin-log plot of the error *vs.* N. Again, by trial and error subtract a constant from $u_{num}(L/2, t = 3.1)$ such that the remainder comes closest to a straight line in the lin-log plot. Show the resulting lin-log plot. Use it to extrapolate your result to $N \to \infty$.
Is the temporal or the spatial error dominant for the parameters you used in **4.**?

6. Comparison between Crank-Nicholson and Backward Euler
Choose $A = 3$, $B = 9.5$, $D_1 = 0.5$, $D_2 = 1$, $L = 1$, $N = 32$, $dt = 0.01$, with initial conditions $u = 0 = v$ and boundary conditions $\gamma_{1,2;0,L} = 3$. Note that the initial conditions do not satisfy the boundary conditions. Evolve this solution up to $t = 5$ with Crank-Nicholson ($\alpha = 0.5$) and with backward Euler ($\alpha = 1$). Comment on the difference between the behavior of the two methods.

Subroutine flush

```
      subroutine flush(iunit)

c A subroutine to write all pending output to a file attached
c to logical unit iunit (integer)

c David Raitt, February 21, 1994

      character*65 file
      integer iunit

      inquire(iunit,err=10,NAME=file)
      close(iunit)
      open(iunit,file=file,status='old',access='APPEND',err=10)

      return

   10 write(*,*) 'File handling error in subroutine flush.'

      return

      end
```