ADI for Reaction-Diffusion Systems

Stephanie Christ

August 4, 2015

1 Gray-Scott Reaction-Diffusion System

Reaction-Diffusion systems describe how the concentration of chemical substances changes in space. The interaction of the two chemical species U and V is through chemical reactions, the distribution in space due to diffusion. The reaction between the two species can be described by the following chemical terms:

$$U + 2V \to 3V$$
$$V \to P$$

By combining the reaction and the diffusion, we obtain differential equations for the concentrations u and v.

$$\frac{\partial u}{\partial t} = D_u \Delta u - uv^2 + F(1-u) \tag{1}$$
$$\frac{\partial v}{\partial t} = D_v \Delta v + uv^2 - (F+k)v \tag{2}$$

D_u and D_v denote the diffusion coefficients for u and v respectively. F and k are constant coefficient which describe at which rate the chemical reactions occur.

The model is in 2 dimensions.

2 Numerical Method

We split our two-dimensional domain in square cells.

To solve the equations for u and v (equations 1 and 2), we split the reaction and the diffusion parts to solve them separately. First, we solve the diffusion and then we add the reaction to the obtained result. u and v can both be computed separately in each time step, as their contributions to the other equation is only explicit.

2.1 ADI (Alternating Direction Implicit)

To solve the diffusion equation $\frac{\partial \rho}{\partial t} = D_\rho \Delta \rho$, we use the ADI method. This method has first been proposed by Craig and Sneyd in 1988 [1].

First half-step:

$$\rho_{i,j}^{n+\frac{1}{2}} = \rho_{i,j}^{n} + \frac{D_\rho \delta t}{2}\left[\frac{\partial^2 \rho_{i,j}^{n+\frac{1}{2}}}{\partial x^2} + \frac{\partial^2 \rho_{i,j}^{n}}{\partial y^2}\right] \tag{3}$$

Second half-step:

$$\rho_{i,j}^{n+1} = \rho_{i,j}^{n+\frac{1}{2}} + \frac{D_\rho \delta t}{2}\left[\frac{\partial^2 \rho_{i,j}^{n+\frac{1}{2}}}{\partial x^2} + \frac{\partial^2 \rho_{i,j}^{n+1}}{\partial y^2}\right] \tag{4}$$

We approximate the second derivative using a second order central differences scheme:

$$\frac{\partial \rho_{i,j}}{\partial x^2} = \frac{\rho_{i-1,j} - 2\rho_{i,j} + \rho_{i+1,j}}{\Delta x^2} \tag{5}$$

$$\frac{\partial \rho_{i,j}}{\partial y^2} = \frac{\rho_{i,j-1} - 2\rho_{i,j} + \rho_{i,j+1}}{\Delta y^2} \tag{6}$$

Inserting this into equation 3 for the first half-step and setting $\Delta x = \Delta y$, we obtain the following equation:

$$\rho_{i,j}^{n+\frac{1}{2}} = \rho_{i,j}^{n} + \frac{D_\rho \delta t}{2}\left[\frac{\rho_{i-1,j}^{n+\frac{1}{2}} - 2\rho_{i,j}^{n+\frac{1}{2}} + \rho_{i+1,j}^{n+\frac{1}{2}}}{\Delta x^2} + \frac{\rho_{i-1,j}^{n} - 2\rho_{i,j}^{n} + \rho_{i+1,j}^{n}}{\Delta x^2}\right] \tag{7}$$

We sort the implicit and explicit parts and get the following system:

$$\begin{aligned} -\frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i-1,j}^{n+\frac{1}{2}} + \left(1 + \frac{D_\rho \Delta t}{\Delta x^2}\right)\rho_{i,j}^{n+\frac{1}{2}} - \frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i+1,j}^{n+\frac{1}{2}} \\ = \frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i,j-1}^{n} + \left(1 - \frac{D_\rho \Delta t}{\Delta x^2}\right)\rho_{i,j}^{n} - \frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i,j+1}^{n} \end{aligned} \tag{8}$$

This gives us a tridiagonal system with a constant on each diagonal for each row of the domain. On the lower and upper diagonals, we have the coefficient $-\frac{D_\rho \Delta t}{2\Delta x^2}$, on the middle diagonal the coefficient $\left(1 + \frac{D_\rho \Delta t}{\Delta x^2}\right)$.

Performing analogous calculations for the second half-step 4 yields

$$\begin{aligned} -\frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i,j-1}^{n+1} + \left(1 + \frac{D_\rho \Delta t}{\Delta x^2}\right)\rho_{i,j}^{n+1} - \frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i,j+1}^{n+1} \\ = \frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i-1,j}^{n+\frac{1}{2}} + \left(1 - \frac{D_\rho \Delta t}{\Delta x^2}\right)\rho_{i,j}^{n+\frac{1}{2}} - \frac{D_\rho \Delta t}{2\Delta x^2}\rho_{i+1,j}^{n+\frac{1}{2}} \end{aligned} \tag{9}$$

The second half-step gives a tridiagonal system to solve for each column of the domain.

Thomas Algorithm Tridiagonal systems $Ax = d$ can be solved with the Thomas Algorithm in $\mathcal{O}(n)$ time.

$$Ax = d \tag{10}$$

$$\begin{bmatrix} b_0 & c_0 & & & 0 \\ a_1 & b_1 & c_1 & & \\ & a_2 & b_2 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \tag{11}$$

The method first modifies the coefficients in the following way:

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & ; \quad i = 0 \\ \dfrac{c_i}{b_i - a_i c'_{i-1}} & ; \quad i = 1, 2, \ldots, n-1 \end{cases} \tag{12}$$

$$d'_i = \begin{cases} \dfrac{d_i}{b_i} & ; \quad i = 0 \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} & ; \quad i = 1, 2 \ldots, n. \end{cases} \tag{13}$$

Then the solution is obtained by backwards substitution.

$$x_n = d'_n \tag{14}$$

$$x_i = d'_i - c'_i x_{i+1} \qquad ; \; i = n-1, n-2, \ldots, 0. \tag{15}$$

2.2 Reaction Term

To add the reaction term to the solution of the diffusion equation, we use a forward Euler scheme. $\tilde{u}^{n+1}$ and $\tilde{v}^{n+1}$ denote the solution of the diffusion equations before adding the contribution of the reaction term.

$$u_{i,j}^{n+1} = \tilde{u}_{i,j}^{n+1} + \Delta t \left(-\tilde{u}_{i,j}^{n+1} (\tilde{v}_{i,j}^{n+1})^2 + F(1 - \tilde{u}_{i,j}^{n+1}) \right) \tag{16}$$

$$v_{i,j}^{n+1} = \tilde{v}_{i,j}^{n+1} + \Delta t \left(\tilde{u}_{i,j}^{n+1} (\tilde{v}_{i,j}^{n+1})^2 - (F + k) \tilde{v}_{i,j}^{n+1} \right) \tag{17}$$

2.3 Boundary conditions

We use reflecting boundary conditions. This modifies the tridiagonal matrix system, so that the values on each diagonal aren't constant for the whole diagonal anymore. The first value on the upper diagonal, as well as the last value on the lower diagonal, are doubled.

3 Implementation

We save the grid for u and v each in a `std::vector<double>` in row-wise storage. For the tridiagonal matrix, we wrote our own class to simplify the access to the diagonals.

To visualize the result, we used OpenGL for displaying the concentration of u during the computation, and a library called `lodepng` – written by Lode Vandevenne – to save the image after the simulation.

3.1 Serial version

We loop over all time steps and apply the numerical scheme described above.

In each step, we first apply the ADI method to u and v to solve the diffusion part of the problem. Then we solve the reaction.

For the ADI-step, we have to treat the boundary rows/columns differently, as the dependencies on the neighbouring cells change (see 1. To facilitate the access to a grid cell, we have defined macros that let us use the two-dimensional coordinates of the cells. The solver for the tridiagonal matrix is an implementation of the Thomas algorithm, see section 3.2.

```
// perform the first half-step
// loop over all rows

// j=0
for (int i=0; i<N_; ++i) {
    uRhs[i] = U(i,0) + uCoeff * (U(i,1) - U(i,0));
    vRhs[i] = V(i,0) + vCoeff * (V(i,1) - V(i,0));
}
TriDiagMatrixSolver::solve(N_, matU1_, uRhs, &UHALF(0,0), 1);
TriDiagMatrixSolver::solve(N_, matV1_, vRhs, &VHALF(0,0), 1);

// inner grid points
for (int j=1; j<N_-1; ++j) {
    // create right-hand side of the systems
    for (int i=0; i<N_; ++i) {
        uRhs[i] = U(i,j) + uCoeff * (U(i,j+1) - 2.*U(i,j) + U(i, ↩
            j-1));
        vRhs[i] = V(i,j) + vCoeff * (V(i,j+1) - 2.*V(i,j) + V(i, ↩
            j-1));
    }

    TriDiagMatrixSolver::solve(N_, matU1_, uRhs, &UHALF(0,j), 1);
    TriDiagMatrixSolver::solve(N_, matV1_, vRhs, &VHALF(0,j), 1);
}

// j=N_-1
for (int i=0; i<N_; ++i) {
    uRhs[i] = U(i,N_-1) + uCoeff * (- U(i,N_-1) + U(i,N_-2));
    vRhs[i] = V(i,N_-1) + vCoeff * (- V(i,N_-1) + V(i,N_-2));
}
TriDiagMatrixSolver::solve(N_, matU1_, uRhs, &UHALF(0,N_-1), 1);
TriDiagMatrixSolver::solve(N_, matV1_, vRhs, &VHALF(0,N_-1), 1);
```

Listing 1: Serial implementation of the first half-step

The second half-step is implemented analogously to the first half-step, but with the loop over the columns `i` instead of the rows `j`.

For the reaction, we used a simple loop over all grid cells and add the reaction term to u and v.

3.2 Thomas Algorithm

For the implementation of the Thomas Algorithm, we used the version of wikipedia from April 2014 [2]. We modified it so that we can pass the whole grid as an argument. Since the access to the grid is row-/column-wise argument `result` is a pointer to the first element we want to access. The increment `inc` is for the distinction between row-wise and column-wise access and is either 1 or N (the size of the grid in one dimension).

```
1  namespace TriDiagMatrixSolver
2  {
3      /**
4       * Solve a tridiagonal matrix system.
5       *
6       * @param n          number of elements in the result
7       * @param mat        tridiagonal matrix
8       * @param rhs        right-hand side of the system
9       * @param result     vector for the result, pointer to the ↩
            first element
10      * @param inc        increment for the elements of the result
11      */
12     void solve(unsigned int n, const TriDiagMatrix& mat, const ↩
           std::vector<double>& rhs, double *result, const unsigned ↩
           int inc);
13 };
```

Listing 2: Declaration of our implementation of the Thomas algorithm

References

[1] I.J.D. Craig and A.D. Sneyd. An alternating-direction implicit scheme for parabolic equations with mixed derivatives. *Computers & Mathematics with Applications*, 16(4):341350, 1988.

[2] Wikipedia. Tridiagonal matrix algorithm — Wikipedia, the free encyclopedia, 2014. [Online; Version from 9 April 2014].