Iterative methods for contamination model (edited 6/12/18)

The model, as presented in the project description is:

$$\begin{aligned}
\frac{\partial^2 C}{\partial r^2} + \frac{1}{r}\frac{\partial C}{\partial r} + \frac{1}{r^2}\frac{\partial^2 C}{\partial \theta^2} - gC = -S, \\
S = exp\left[-20\left((r-r_0)^2 + (\theta-\theta_0)^2\right)\right], \\
1 \le r \le \pi + 1, \\
0 \le \theta \le \pi, \\
C(r = \pi + 1) = C(\theta = 0) = C(\theta = \pi) = 0, \\
C(r = 1) = sin^2(k\theta).
\end{aligned}$$

The equations will be solved numerically on an *n+2 x n+2* grid with $r_i = 1 + i\Delta$, $\theta_j = j\Delta$, $i, j \in 0, 1, \ldots, n+1$ and the grid spacing is, $\Delta = \pi/(n+1)$. The derivatives in the model are approximated with 2nd-order centered finite differences, *e.g.*

$$\begin{aligned}
\frac{\partial C}{\partial r}|_{r_i,\theta_j} \approx \frac{C_{i+1,j} - C_{i-1,j}}{2\Delta}, \\
\frac{\partial^2 C}{\partial \theta^2}|_{r_i,\theta_j} \approx \frac{C_{i,j+1} - 2C_{i,j} + C_{i,j-1}}{\Delta^2},
\end{aligned}$$

with analogous expressions for the other derivatives. The model PDE then is rewritten as the following system of equations:

$$\begin{aligned}
\alpha_i^m C_{i-1,j} + \beta_i C_{i,j-1} - \gamma_i C_{i,j} + \beta_i C_{i,j+1} + \alpha_i^p C_{i+1,j} + \Delta^2 S_{i,j} = 0, \\
\alpha_i^m = 1 - \Delta/(2r_i), \\
\alpha_i^p = 1 + \Delta/(2r_i), \\
\beta_i = 1/r_i^2, \\
\gamma_i = g\Delta^2 + 2 + 2/r_i^2.
\end{aligned}$$

Note that we recover the Poisson equation example from lecture by setting $\alpha_i^m = \alpha_i^p = 0$, $\beta_i = 1$, and $\gamma_i = 4$.

Descriptions of three iterative methods for solving this system of equations are provided below.

Jacobi iteration

This is a straightforward modification of examples from class. The approximate solution for $C_{i,j}$ at iteration *k+1* is found by solving:

$$\gamma_i C_{i,j}^{(k+1)} = \alpha_i^m C_{i-1,j}^{(k)} + \beta_i C_{i,j-1}^{(k)} + \beta_i C_{i,j+1}^{(k)} + \alpha_i^p C_{i+1,j}^{(k)} + \Delta^2 S_{i,j}.$$

Python and Fortran codes solving this system have been provided.

Over-step iteration (OSI)

The OSI method *pushes* the iterations forward by 1) adding a portion of $C_{i,j}^{(k)}$ to the update equation and 2) using $C_{i-1,j}^{(k+1)}$ and $C_{i,j-1}^{(k+1)}$ instead of $C_{i-1,j}^{(k)}$ and $C_{i,j-1}^{(k)}$, respectively, when computing $C_{i,j}^{(k+1)}$. This assumes, of course, that $C_{i-1,j}^{(k+1)}$ and $C_{i,j-1}^{(k+1)}$ have been computed prior to the calculation of $C_{i,j}^{(k)}$. This naturally occurs if calculations begin at *(i,j)=(1,1)*, and then progress through the matrix element-by-element one row at a time. The OSI update equation then is:

$$\gamma_i C_{i,j}^{(k+1)} = \frac{1}{2}\left[-\gamma_i C_{i,j}^{(k)} + 3\left(\alpha_i^m C_{i-1,j}^{(k+1)} + \beta_i C_{i,j-1}^{(k+1)} + \beta_i C_{i,j+1}^{(k)} + \alpha_i^p C_{i+1,j}^{(k)} + \Delta^2 S_{i,j}\right)\right].$$

You will have to develop Python and Fortran implementations of this method in part 2 of the project.

Alternating Over-step iteration (AOS)

Consider labeling the $(n+2)^2$ grid points as black or white following the pattern on a chess board. Then, *(i,j) = (1,1), (1,3), ..., (2,2) (2,4), ...* would be white and *(1,2), (1,4), ..., (2,1), (2,3), ...* would be black. The AOS method first updates the 'white' grid points using values from the previous iteration. Then the 'black' points are updated using the updated values at the white points and values from the previous iterations at the black points. The AOS update equations are:

white update

$$\gamma_i C_{i,j}^{(k+1)} = \frac{1}{2}\left[-\gamma_i C_{i,j}^{(k)} + 3\left(\alpha_i^m C_{i-1,j}^{(k)} + \beta_i C_{i,j-1}^{(k)} + \beta_i C_{i,j+1}^{(k)} + \alpha_i^p C_{i+1,j}^{(k)} + \Delta^2 S_{i,j}\right)\right].$$

black update

$$\gamma_i C_{i,j}^{(k+1)} = \frac{1}{2}\left[-\gamma_i C_{i,j}^{(k)} + 3\left(\alpha_i^m C_{i-1,j}^{(k+1)} + \beta_i C_{i,j-1}^{(k+1)} + \beta_i C_{i,j+1}^{(k+1)} + \alpha_i^p C_{i+1,j}^{(k+1)} + \Delta^2 S_{i,j}\right)\right].$$

Of course, there is no need to update C for points on the boundaries if the initial C satisfied the boundary conditions. You will develop a Fortran + MPI implementation of this method in part 3 of the project.

In all of the methods above, iterations terminate when the magnitude of the maximum difference between $C^{(k+1)}$ and $C^{(k)}$ falls below a specified tolerance.