# CS 6210: Matrix Computations

**Stationary iterations**

David Bindel

2025-11-10

## The need for model problems

Direct methods for solving linear systems and eigenvalue problems are (mostly) "black box." We design algorithms that work well for a broad category of problems with given structural properties; once we understand the structure, there is often a reasonably routine choice of solvers. Of course, even for direct methods, it is not entirely true that we get "black box" performance — for example, the fill in sparse direct factorization methods is highly dependent on the sparsity structure of the matrix at hand. Nonetheless, users of sparse solvers can largely leave the details to specialists once they understand the basic lay of the land.

For the remainder of the semester, we will focus on iterative solvers, which are a different beast altogether. Iterative solvers produce a sequence of approximate solutions that (ideally) converge to the true solution to a linear system or eigenvalue problem. However, the rate of convergence is highly dependent on both the iterative method and the details of the problem. Even when we are able to take advantage of a good library of iterative solvers, there are often a wide variety of methods to choose from and a large number of parameters that we need to understand and tune to get good performance.

Because iterative methods are more problem-dependent than direct methods, we will focus our presentation on a set of model problems that exhibit characteristics common in many problems drawn from physical models. We will also comment on other types of problem structures as we go along, but will mostly leave the details to select homework problems.

## The 1D model problem

It is difficult to say many useful things about the convergence of iterative methods without looking at a concrete problem. Therefore, we will set the stage with a very specific model problem: a discretization of the Poisson equation. We start with the one-dimensional case.

The continuous version of our model problem is a one-dimensional Poisson equation with homogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\frac{d^2u}{dx^2} &= f \text{ for } x \in (0,1) \\ u(0) &= 0 \\ u(1) &= 0 \end{aligned}$$

Let $x_j = j/(n+1)$ for $j = 0, 1, \dots, n+1$ be a set of mesh points. We can approximate the second derivative of $u$ at a point by a finite difference method:

$$-\frac{d^2u}{dx^2}(x_j) \approx \frac{-u(x_{j-1}) + 2u(x_j) - u(x_{j+1})}{h^2}$$

where $h = 1/(n+1)$ is the mesh spacing. If we replace the second derivative in the Poisson equation with this finite-difference approximation, we have a scheme for computing $u_j \approx u(x_j)$:

$$\begin{aligned} -u_{j-1} + 2u_j - u_{j-1} &= h^2 f_j \text{ for } 1 \leq j \leq n \\ u_0 &= 0 \\ u_{n+1} &= 0 \end{aligned}$$

We can write this approximation as a matrix equation $Tu = h^2 f$, where

$$T = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}$$

Part of what makes this simple Poisson discretization so appealing as a model problem is that we can compute the eigenvalues and eigenvectors directly. This is because solving the $(T - \lambda)\psi = 0$ is equivalent to considering the constant coefficient difference equation

$$\psi_{k+1} - (2 - \lambda)\psi_k + \psi_{k-1} = 0$$

subject to the boundary conditions $\psi_0 = \psi_{n+1} = 0$. Solutions to this difference equation must have the form

$$\psi_k = \alpha\xi^k + \beta\bar{\xi}^k,$$

where $\xi$ and $\bar{\xi}$ are the roots of the characteristic polynomial $p(z) = z^2 - (2 - \lambda)z + 1$. For $0 \leq \lambda \leq 4$, these roots form a complex conjugate pair, each with unit magnitude; that is, we can write $\xi = \exp(i\theta)$ for some $\theta$, and so

$$\xi^k = \exp(ik\theta) = \cos(k\theta) + i\sin(k\theta).$$

Thus, any solution to the difference equation must have the form

$$\psi_k = \gamma \cos(k\theta) + \mu \sin(k\theta).$$

Plugging in the boundary conditions, we find that $\gamma = 0$, and $\theta = l\pi/(n+1)$ for some $l$. Thus, the normalized eigenvectors of $T$ are $z_j$ with entries

$$\begin{aligned} z_j(k) &= \sqrt{\frac{2}{n+1}} \sin\left(\frac{jk\pi}{n+1}\right) \\ &= \sqrt{\frac{2}{n+1}} \sin((j\pi)x_k) \end{aligned}$$

and the corresponding eigenvalues are

$$\lambda_j = 2\left(1 - \cos\frac{\pi j}{n+1}\right).$$

For $j \ll n$, Taylor expansion gives that

$$\lambda_j = h^2(\pi j)^2 + O\left(h^4(\pi j)^4\right).$$

By way of comparison, the continuous Dirichlet eigenvalue problem

$$-\frac{d^2w}{dx^2} = \mu w, \quad w(0) = w(1) = 0$$

has eigenfunctions of the form

$$w_j = \sin(j\pi x), \quad \mu_j = (j\pi)^2.$$

Thus, the eigenvectors of $h^{-2}T$ are *exactly* the sampled eigenfunctions of $-d^2/dx^2$ on $[0,1]$ with Dirichlet boundary conditions, while the extremal eigenvalues of $h^{-2}T$ satisfy

$$h^{-2}\lambda_j = \mu_j + O(\mu_j^2 h^2).$$

## The 2D model problem

The problem with the 1D Poisson equation is that it doesn't make a terribly convincing challenge – since it is a symmetric positive definite tridiagonal, we can solve it in linear time with Gaussian elimination! So let us turn to a slightly more complicated example: the Poisson equation in 2D. Before discussing the 2D Poisson equation, though, let us digress to introduce two useful notations: the vec operator and the Kronecker product.

The vec operator simply lists the entries of a matrix (or an array with more than two indices) in column-major order; for example,

$$\operatorname{vec} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix}.$$

The Kronecker product $A \otimes B$ of two matrices is a block matrix where each block is a scalar multiple of $B$:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \ldots \\ a_{21}B & a_{22}B & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

The Kronecker product and the vec operation interact with each other as follows:

$$(B \otimes A) \operatorname{vec}(C) = \operatorname{vec}(ACB^T).$$

The Kronecker product also satisfies the identities

$$(A \otimes B)^T = A^T \otimes B^T$$
$$(A \otimes B)(C \times D) = (AB) \otimes (CD)$$

which implies, for example, that the Schur form of a Kronecker product is a Kronecker product of Schur forms:

$$(U_A \otimes U_B)^*(A \otimes B)(U_A \otimes U_B) = T_A \otimes T_B.$$

As one illustrative application of Kronecker products, consider the Sylvester operator $X \mapsto AX - XB$. Using Kronecker products, we can write this as

$$\operatorname{vec}(AX - XB) = (A \otimes I - I \otimes B) \operatorname{vec}(X).$$

Note that if $A = U_A T_A U_A^*$ and $B = U_B T_B U_B^*$ are Schur forms, then

$$A \otimes I - I \otimes B = (U_A \otimes U_B)(T_A \otimes I - I \otimes T_B)(U_A \otimes U_B)^*,$$

and $T_A \otimes I - T_B \otimes I$ is an upper triangular matrix. This transformation, followed by a triangular solve, is essentially what you did in problem 3 of your last homework.

Now let us return to the model 2D Poisson discretization. This is an approximation to the equation

$$-\nabla^2 u = -\left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = f$$

for $(x, y) \in (0, 1)^2$, with Dirichlet boundary conditions $u(x, y) = 0$ for $|x| = 1$ or $|y| = 1$. If we discretize on a regular mesh with interior points indexed by $1 \leq i \leq n$ and $1 \leq j \leq n$, we can write the solution as a matrix $U$. When we discretize, we have a partial derivative in $x$

corresponding to acting across columns of $U$, and a partial derivative in $y$ corresponding to acting across rows of $U$. We can write this operation as

$$TU + UT = h^2 F,$$

or as an ordinary matrix equation of dimension $N = n^2$

$$(T \otimes I + I \otimes T) \operatorname{vec}(U) = h^2 \operatorname{vec}(F).$$

What properties do we have for $T_{n \times n} = T \otimes I + I \otimes T$?

1. $T_{n\times n}$ is symmetric and positive definite.
2. $T_{n\times n}$ is (non-strictly) diagonally dominant.
3. If $(z_j, \lambda_j)$ are the eigenpairs for $T$, those for $T_{n\times n}$ are $(z_j \otimes z_l, \lambda_j + \lambda_l)$.
4. The condition number of $T_{n\times n}$ scales like $O(h^{-2})$.

## Methods for solving the 2D model problem

Suppose we wanted to solve the 2D model problem in practice. What methods do we have at our disposal so far? Of course, we have several direct methods

1. We could run Gaussian elimination on $T_{n\times n}$. This takes time $O(N^3)$, where $N = n^2$.
2. The matrix $T_{n\times n}$ is also a banded matrix with bandwidth $n$ so we could do band Gaussian elimination at a cost of $O(N^2 n) = O(N^{2.5})$.
3. A sparse direct solve using nested dissection ordering runs in $O(N^{1.5})$.
4. Treating the problem as a Sylvester equation and running Bartels-Stewart requires $O(n^3)$ time to find the eigensystem of $T$ and to transform $U$ and $F$ using the eigenvector matrix; and $O(n^2)$ time for the subsequent (diagonal) linear solve.
5. The eigenvector matrix for $T$ corresponds to a *discrete sine transform*, which is closely related to the FFT; and we know the eigenvalues in closed form. This allows us to reduce the time for Bartels-Stewart to $O(n^2 \log n) = O(N \log N)$.

In the coming lectures, we turn to a variety of *iterative methods*. These methods do not produce an exact answer, but rather produce a sequence of ever-better approximations to the truth. With appropriate parameter choices, the time to reduce the error by a constant factor scales like[1]

[1] See Table 6.1 of *Applied Numerical Linear Algebra* by J. Demmel.

| | |
|---|---|
| Jacobi | $N^2$ |
| Gauss-Seidel | $N^2$ |
| CG | $N^{3/2}$ |
| SOR | $N^{3/2}$ |
| SSOR with Chebyshev acceleration | $N^{5/4}$ |
| Multigrid | $N$ |

For both the direct and iterative methods, the more structure we use, the faster we can go.

## Iteration basics

An iterative solver for $Ax = b$ is produces a sequence of approximations $x^{(k)} \rightarrow x$. We always stop after finitely many steps, based on some convergence criterion, e.g.

- A residual estimate reached some threshold tolerance (relative to $b$ or to the initial residual).
- An error estimate reached some threshold tolerance (usually relative to the initial error estimate).
- We reach a maximum iteration count.

We say we have solved the problem when some error-related tolerance is satisfied. We can often reason about the cost per step in a simple way, but estimating the steps to a solution can be quite complicated. It depends on the nature of the iteration, the structure of the problem, the norms used to judge convergence, and the problem tolerances.

The oldest and simplest iterations for solving linear systems are *stationary iterations* (a.k.a. *fixed point iterations*) and more generally *relaxation iterations*. In many cases, these iterations have been supplanted by more sophisticated methods (such as Krylov subspace methods), but they remain a useful building block. Moreover, what is old has a way of becoming new again; many of the classic iterations from the 1950s and 1960s are seeing new life in applications to machine learning and large scale optimization problems.

## Stationary iterations

Stationary iterations are so named because the solution to a linear system is expressed as a stationary point (fixed point) of

$$x^{(k+1)} = F(x^{(k)}).$$

A sufficient (though not necessary) condition for convergence is that the mapping is a contraction, i.e. there is an $\alpha < 1$ such that for all $x, y$ in the vector space,

$$\|F(x) - F(y)\| \leq \alpha\|x - y\|.$$

The constant $\alpha$ is the rate of convergence.

If we are solving a linear equation $Ax = b$, it generally makes sense to write a fixed point iteration where the mapping $F$ is affine. We can write any such iteration via a *splitting* of the matrix $A$, i.e. by writing $A = M - N$ with $M$ nonsingular. Then we rewrite $Ax = b$ as

$$Mx = Nx + b,$$

and the fixed point iteration is

$$Mx^{(k+1)} = Nx^{(k)} + b,$$

which we may rewrite as

$$x^{(k+1)} = x^{(k)} + M^{-1}(b - Ax^{(k)}).$$

## Error iteration and convergence

We derive an error iteration by subtracting the fixed point equation from the iteration equation

$$\begin{array}{r}
Mx^{(k+1)} = Nx^{(k)} + b \\
-[Mx = Nx + b] \\
\hline
Me^{(k+1)} = Ne^{(k)}
\end{array}$$

or $e^{(k+1)} = Re^{(k)}$ where $R \equiv M^{-1}N$ is the *iteration matrix*. A sufficient condition for convergence is that $\|R\| < 1$ in some operator norm. The necessary and sufficient condition is that $\rho(R) < 1$, where the spectral radius $\rho(R)$ is defined as $\max|\lambda|$ over all eigenvalues $\lambda$ of $R$.

The choice of $M$ is key to the success of an iterative method. Ideally, we want it to be easy to solve linear systems with $M$ (low set-up time for any initial factorizations, and a low cost per iteration to solve), but we also want $R$ to have a small norm or spectral radius. Often, there is a direct tension between these two. For example, the “best” choice of $M$ from the perspective of iteration count is $M = A$. But this is a silly thing to do: the iteration converges after one step, but that step is to solve $Ax = b$!

## Complexity of stationary iterations

What is the cost to “solve” a system of linear equations using a stationary iteration? We never exactly solve the system, so we need a convergence criterion to address this problem. Let us instead ask the time to satisfy $\|e^{(k)}\| \leq \epsilon\|e^{(0)}\|$, where $\|e^{(0)}\|$ is the initial error. Supposing $\|R\| < 1$, we know

$$\|e^{(k)}\| \leq \|R\|^k\|e^{(0)}\|,$$

so the criterion should be met after $\lceil \log(\epsilon)/\log(\|R\|) \rceil$ steps. While norms on a finite-dimensional space are all equivalent, the constants involved may depend on the dimension of the space. Therefore, when we analyze the complexity of a stationary iteration, we must specify the family of norms (of either the error or the residual) that we are using to judge convergence.

The cost per step depends on the time to solve a linear system with $M$ and the time to form a residual. For many of the basic stationary iterations, the time per step is $O(\mathrm{nnz}(A))$, where $\mathrm{nnz}(A)$ is the number of nonzero elements in the matrix $A$. The number of steps, though, depends very strongly on not just the number of nonzeros, but more detailed properties of $A$. Therefore, we generally cannot describe the asymptotic complexity of an iterative method except in the context of a very specific family of matrices (such as the 2D Poisson model problem).

## The classical iterations

One of the simplest stationary iterations is *Richardson iteration*, in which $M$ is chosen to be proportional to the identity:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega(b - Ax^{(k)}) \\ &= (I - \omega A)x^{(k)} + \omega b. \end{aligned}$$

The iteration matrix in this case is simply $R = I - \omega A$. If $A$ is symmetric and positive definite, we can always make Richardson iteration converge with an appropriate $\omega$, though the convergence may be heart-breakingly slow.

Let $A = D - L - U$, where $D$ is diagonal, $L$ is strictly lower triangular, and $U$ is strictly upper triangular. Jacobi iteration takes $M = D$. When we discuss multigrid, we will also see *damped Jacobi*, for which $M = \omega^{-1}D$ with $\omega < 1$. Damped Jacobi is equivalent to moving in the Jacobi direction by a fraction $\omega$ of the usual step length. Like Richardson, we can always make (damped) Jacobi converge for SPD matrices; the method also converges for $A$ strictly diagonally dominant.

The *Gauss-Seidel* iteration incorporates a little more of $A$ into $M$, taking $M = D - L$. For $A$ symmetric and positive definite, this generally yields about twice the rate of convergence of Jacobi; and it is not necessary to damp the method to obtain convergence. However, Gauss-Seidel is less friendly to parallel computing because the triangular solve involves computing in a strict order.

### Splitting, sweeping, and updating

While we typically analyze stationary methods in terms of a splitting, that is not always how we implement them.

We can think of either Jacobi or Gauss-Seidel as a *sweep* over the variables, in which we update the value of variable $i$ using the $i$th equation and using a guess for all the other variables. In the Jacobi iteration, the guess for the other variables comes from the previous step; in Gauss-Seidel, the guess for the other variables involves whatever our most up-to-date information might be.

For more complicated $M$, we often use the update form of the stationary iteration. This just requires that we be able to compute the residual and solve linear systems with $M$.

## Over-relaxation

In the Jacobi iteration, we take $M = D$; in Gauss-Seidel, we take $M = D - L$. In general, Gauss-Seidel works better than Jacobi. So if we go even further in the "Gauss-Seidel" direction, perhaps we will do better still? This is the idea behind *successive over-relaxation*, which uses the splitting $M = D - \omega L$ for $\omega > 1$. The case $\omega < 1$ is called *under-relaxation*.

The iteration converges for positive definite $A$ for any $\omega \in (0, 2)$. The optimal choice is problem-dependent; but it is rarely of interest any more, since SOR is mostly used to accelerate more sophisticated iterative methods. Indeed, the most widely-used variant of SOR involves a forward sweep and a backward sweep; this SSOR iteration applied to an SPD $A$ matrix yields an SPD splitting matrix $M$, and can therefore be used to accelerate the conjugate gradient method (which depends on this structure).

## Red-black ordering

In Jacobi iteration, we can compute the updates for each equation independently of all other updates — order does not matter, and so the method is ripe for parallelism within one sweep[2] In general, though, Gauss-Seidel and over-relaxation methods depend on the order in which we update variables. The *red-black* ordering (or more general *multi-color ordering*) trick involves re-ordering the unknowns in our matrix by "color," where each unknown is assigned a color such that no neighbor in the graph of the matrix has the same color. In the 2D Poisson case, this can be achieved with two colors, usually dubbed "red" and "black," applied in a checkerboard pattern.

## Block iterations

So far, we have restricted our attention to *point relaxation* methods that update a single variable at a time. *Block* versions of Jacobi and Gauss-Seidel have exactly the same flavor as the regular versions, but they update a subset of variables simultaneously. These methods

[2] There is actually not enough work per sweep to make this worthwhile with ordinary Jacobi, usually, but it is worthwhile if we deal with the block variants.

correspond to a splitting with $M$ equal to the block diagonal or block lower triangular part of $A$.

The block Jacobi and Gauss-Seidel methods update disjoint subsets of variables. The *Schwarz* methods act on *overlapping* subsets It turns out that a little overlap can have a surprisingly large benefit. The book *Domain Decomposition* by Smith, Gropp, and Keyes provides a nice overview.

## Convergence of stationary iterations

For general non-symmetric (and nonsingular) matrices, none of the classical iterations is guaranteed to converge. But there are a few classes of problems for which we can say something about the convergence of the classical iterations, and we survey some of these now.

### Strictly row diagonally-dominant problems

Suppose $A$ is strictly diagonally dominant. Then by definition, the iteration matrix for Jacobi iteration ($R = D^{-1}(L+U)$) must satisfy $\|R\|_\infty < 1$, and therefore Jacobi iteration converges in this norm. A bound on the rate of convergence has to do with the strength of the diagonal dominance. Moreover, one can show (though we will not) that in this case

$$\|(D-L)^{-1}U\|_\infty \leq \|D^{-1}(L+U)\|_\infty < 1,$$

so Gauss-Seidel converges at least as quickly as Jacobi. The Richardson iteration is also guaranteed to converge, at least so long as $\omega < 1/(\max_i |a_i i|)$, since this is sufficient to guarantee that all the Gershgorin disks of $I - \omega A$ will remain within the unit circle.

### Symmetric and positive definite problems

#### Richardson iteration

If $A$ is SPD with eigenvalues $0 < \lambda_1 < ... < \lambda_n$, then Richardson iteration satisfies

$$\|R\|_2 = \max(|1-\omega\lambda_1|, |1-\omega\lambda_n|);$$

and the rate of convergence is optimal when $\omega = 2/(\lambda_1+\lambda_n)$, which yields \$ $\|R\|_2 = 1-\frac{2\lambda_1}{\lambda_1+\lambda_n} = 1 - \frac{2}{\kappa(A)+1}$\$ If $A$ is ill-conditioned, the iteration may be painfully slow.

### Jacobi iteration

The error iteration for Jacobi is

$$e^{(k+1)} = D^{-1}(L+U)e^{(k)} = D^{-1}(D-A)e^{(k)}.$$

If $A$ is SPD, then so is $D$, and therefore it induces a norm; scaling the error iteration by $D^{1/2}$ gives

$$\hat{e}^{(k+1)} = D^{-1/2}(D-A)D^{-1/2}\hat{e}^{(k)},$$

where $\hat{e}^{(k)} = D^{1/2}e^{(k)}$ and

$$\|\hat{e}^{(k)}\|_2 = \|e^{(k)}\|_D.$$

Therefore

$$\|e^{(k+1)}\|_D \leq \|D^{-1/2}(D-A)D^{-1/2}\|_2 \|e^{(k)}\|_D.$$

For $A$ is symmetric and positive definite, we then have

$$\|D^{-1/2}(D-A)D^{-1/2}\|_2 = \max(|1-\lambda_1|, |1-\lambda_n|),$$

where $0 < \lambda_1 < ... < \lambda_n$ are the eigenvalues of the pencil $(A, D)$. We have convergence when $\lambda_n < 2$, i.e. $2D - A$ is symmetric and positive definite. Damped Jacobi, on the other hand, can always be made to converge for a sufficiently large damping level $\omega$.

The same analysis holds for block Jacobi.

### Gauss-Seidel iteration

To understand the Gauss-Seidel convergence, it is useful to look at the linear system $Ax^{(*)} = b$ as the minimizer of the convex quadratic

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b.$$

Now consider a given $x$ and consider what happens if we update to $x + se_i$ for some $s$. This gives the value

$$\phi(x + se_i) = \phi(x) + \frac{a_{ii}}{2}s^2 + se_i^T Ax - sb_i.$$

Minimizing with respect to $s$ yields

$$a_{ii}s = b_i - e_i^T Ax$$

or

$$a_{ii}(x_i + s) = b_i - \sum_{j \neq i} a_{ij}x_j.$$

But this is precisely the Gauss-Seidel update! Hence, Gauss-Seidel for a positive definite system corresponds to optimization of $\phi$ by *cyclic coordinate descent*. The method decreases $\phi$ at each coordinate step, and each sweep is guaranteed to sufficiently reduce the objective so that we ultimately converge.

The same analysis holds for block Gauss-Seidel.

## Convergence on the 2D model problem

In the case of the 2D model problem, recall that the eigenvalues are

$$\lambda_{i,j} = 2\,(2 - \cos(\pi i h) - \cos(\pi j h))$$

The extreme eigenvalues are

$$\lambda_{1,1} = 2h^2\pi^2 + O(h^4)$$

and

$$\lambda_{n,n} = 4 - 2h^2\pi^2 + O(h^4).$$

The diagonal of $T_{n\times n}$ is simply $4I$, so the Jacobi iteration matrix looks like

$$R = \frac{1}{4}(4I - T_{n\times n}),$$

for which the eigenvalues are

$$\lambda_{i,j}(R) = -(\cos(\pi i h) + \cos(\pi j h))/2,$$

and the spectral radius is

$$\rho(R) = \cos(\pi h) = 1 - \frac{\pi^2 h^2}{2} + O(h^4)$$

Thus, the number of iterations to reduce the error by $1/e$ scales like

$$\frac{2}{\pi^2 h^2} = \frac{2}{\pi^2}(n+1)^2 = O(N);$$

and since each step takes $O(N)$ time, the total time to reduce the error by a constant factor scales like $O(N^2)$.

The successive overrelaxation iteration uses a splitting

$$M = \omega^{-1}(D - \omega\tilde{L}) = \omega^{-1}D^{-1}(I - \omega L),$$

which yields an iteration matrix

$$R_{SOR} = (I - \omega L)^{-1}((1-\omega)I + \omega U).$$

In general, this is rather awkward to deal with, since it is a nonsymmetric matrix. However, for the model problem with a particular ordering of unknowns (red-black ordering), one has that the eigenvalues $\mu$ of $R_J$ correspond to the eigenvalues $\lambda$ of $R_{SOR}$ via

$$(\lambda + \omega - 1)^2 = \lambda\omega^2\mu^2.$$

For the case $\omega = 1$ (Gauss-Seidel), this degenerates to

$$\lambda = \mu^2,$$

and so $\rho(R_{GS}) = \rho(R_J)^2$. Consequently, each Gauss-Seidel iteration reduces the error by the same amount as two Jacobi iterations, i.e. Gauss-Seidel converges twice as fast on the model problem. This tends to be true for other problems similar to the model problem, too. However, going from Jacobi to Gauss-Seidel only improves the convergence rate by a constant factor; it doesn't improve the asymptotic complexity at all. However optimal $\omega$ (about $2 - O(h)$) gives us a spectral radius of $1 - O(h)$ rather than $1 - O(h^2)$, allowing us to accelerate convergence to $O(N^{3/2})$.

The red-black ordering can be convenient for parallel implementation, because allowing the red nodes (or black nodes) to be processed in any orer gives more flexibility for different scheduling choices. But it is also a useful choice for analysis. For example, in the red-black ordering, the model problem looks like

$$A = \begin{bmatrix} 4I & B \\ B^T & 4I \end{bmatrix}$$

The preconditioner based on Jacobi iteration is

$$M_J = \begin{bmatrix} 4I & 0 \\ 0 & 4I \end{bmatrix},$$

which results in the iteration matrix

$$R_J = M_J^{-1}(M_J - A) = \frac{1}{4} \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}.$$

The eigenvalues of $R_J$ are thus plus or minus one quarter the singular values of $B$. Note that this much would have been the same for more general problems with the same structure!

I will not drag you in class through the rest of the analysis, and I would not expect you to repeat it on an exam. Nonetheless, it may be worth writing it out in order to satisfy the curious. The preconditioner for Gauss-Seidel is

$$M_{GS} = \begin{bmatrix} 4I & 0 \\ B^T & 4I \end{bmatrix};$$

and because of the relatively simple form of this matrix, we have

$$M_{GS}^{-1} = \frac{1}{4} \begin{bmatrix} I & 0 \\ B^T/4 & I \end{bmatrix}.$$

The iteration matrix for Gauss-Seidel is

$$R_{GS} = M_{GS}^{-1}(M_{GS} - A) = \begin{bmatrix} 0 & B/4 \\ 0 & -\frac{1}{16} B^T B \end{bmatrix},$$

which has several zero eigenvalues together with some eigenvalues that are minus 1/16 times the squared singular values of $B^T B$. Thus, as indicated earlier, the spectral radius of $R_{GS}$ is the square of the spectral radius of $R_J$ (for the model problem).

The analysis for the general SOR case is slightly messier, but I'll include it here for completeness. The preconditioner is

$$M_{SOR} = \frac{1}{\omega} \begin{bmatrix} 4I & 0 \\ \omega B^T & 4I \end{bmatrix},$$

and the inverse is

$$M_{SOR}^{-1} = \frac{\omega}{4} \begin{bmatrix} I & 0 \\ -\omega B^T/4 & I \end{bmatrix},$$

The iteration matrix is

$$\begin{aligned} R_{SOR} &= \frac{1}{4} \begin{bmatrix} I & 0 \\ -\omega B^T/4 & I \end{bmatrix} \begin{bmatrix} 4(1-\omega)I & -\omega B \\ 0 & 4(1-\omega)I \end{bmatrix} \\ &= \begin{bmatrix} (1-\omega)I & -\omega B/4 \\ -(1-\omega)\omega B^T/4 & \omega^2 B^T B/16 + (1-\omega)I \end{bmatrix}. \end{aligned}$$

If $\lambda$ is any eigenvalue of $R_{SOR}$ except $1-\omega$, we can do partial Gaussian elimination on the eigenvalue equation

$$(R_{SOR} - \mu I)v = 0;$$

after eliminating the first block of variables, we have the residual system

$$\left( \frac{\omega^2}{16} B^T B - (\lambda + \omega - 1)I - \frac{(1-\omega)\omega^2}{16} B^T ((1-\omega-\lambda)I)^{-1} B \right) v_2 = 0,$$

Refactoring, we have

$$\left[ \left( \frac{1-\omega}{\lambda+\omega-1} + 1 \right) \frac{\omega^2}{16} B^T B - (\lambda+\omega-1)I \right] v_2 = 0.$$

From our earlier arguments, letting $\mu$ be an eigenvalue of the Jacobi matrix, we know that $\mu^2$ is an eigenvalue of $B^T B/16$. The corresponding eigenvalues $\lambda$ of $R_{SOR}$ must therefore satisfy

$$\left( \frac{1-\omega}{\lambda+\omega-1} + 1 \right) \omega^2 \mu^2 - (\lambda - \omega - 1) = 0.$$

Multiplying through by $\lambda - \omega - 1$, we have

$$(1 - \omega + \lambda + \omega - 1)\omega^2 \mu^2 - (\lambda - \omega - 1)^2 = 0$$

or

$$\lambda \omega^2 \mu^2 = (\lambda - \omega - 1)^2,$$

which is the formula noted before.

## General relaxations

So far, we have mostly discussed stationary methods in we think of sweeping through all the variables in some fixed order and updating a variable or block of variables at a time. There is nothing that say that the order must be *fixed*, though, if we are willing to forgo the analytical framework of splittings. There are essentially two reasons that we might think to do this:

1. We decide which variable(s) to update next based on some adaptive policy, such as which equations have the largest residual. This leads to the *Gauss-Southwell* method. Various methods for fast (sublinear time) personalized PageRank use this strategy.
2. We update variable(s) on multiple processors, communicating the changes opportunistically. In this case, there may be no real rhyme or reason to the order in which we see updates. These methods are called *chaotic relaxation* or *asynchronous relaxation* approaches, and they have seen a great deal of renewed interest over the past several years for both classical scientific computing problems (e.g. PDE solvers) and for machine learning applications.

## Alternating Direction Implicit

The *alternating direction implicit* approach to the model problem began life as an operator-splitting approach to solving a time-domain diffusion problem. At each step of an ordinary implicit time stepper for the heat equation, one would solve a system of the form

$$(I + \Delta t T)x = b,$$

where $\Delta t$ is small and $T$ is the 2D Laplacian operator. But note that if $T = T_x + T_y$, then

$$(I + \Delta t/2 T_x)(I + \Delta t/2 T_y) = (I + \Delta t T + O(\Delta t)^2);$$

hence, we commit only a small amount of error if instead of solving one system with $T$ we solve two half-step systems involving $T_x$ and $T_y$, respectively, where $T_x$ and $T_y$ are the discretizations of the derivative operator in the $x$ and $y$ directions. This is known as the *alternating direction* method.

In practice, cycling between several different versions of the shift parameter (interpreted above as a time-step) can lead to very rapid convergence of the ADI iteration. This beautiful classical result, which has deep connections to the Zolotarev problem from approximation theory, has taken on renewed usefulness in modern control theory and model reduction, where recent work has connected ADI-type methods for Sylvester equations to various rational Krylov methods.

The ADI method and its relations have also garnered many citations over the past 5–10 years because of their role as prior art for various optimization methods, such as the ADMM method.