Iterative Methods for Solving $A\vec{x} = \vec{b}$

A good (free) online source for iterative methods for solving $A\vec{x} = \vec{b}$ is given in the description of a set of iterative solvers called TEMPLATES found at netlib:

`http://www.netlib.org/linalg/html_templates/Templates.html`

A good paperback monograph on iterative methods is the SIAM book *Iterative Methods for Linear and Nonlinear Equations* by Tim Kelley.

Why do we want to look at solvers other than direct solvers?

- The main reason is storage! Oftentimes (especially in 3D calculations) we have a working code but we are unable to store our coefficient matrix when we attempt to do fine grid calculations even when we take advantage of the structure of our matrix.
- Sometimes iterative methods may take less work than a direct method if we have a good initial guess. For example, if we are solving a time dependent

problem for a sequence of time steps, $\Delta t, 2\Delta t, 3\Delta t, \ldots$ then we can use the solution at $k\Delta t$ as a starting guess for our iterative method at $(k+1)\Delta t$ and if Δt is sufficiently small, the method may converge in just a handful of iterations.

- If we have a large sparse matrix (the structure of zeros is random or there are many zeros inside the bandwidth) then one should use a method which only stores the nonzero entries in the matrix. There are direct methods for sparse matrices but they are much more complicated than iterative methods for sparse matrices.

Complications with using iterative methods

- When we use an iterative method, the first thing we realize is that we have to decide how to terminate our method; this was not an issue with direct methods.
- Another complication is that many methods are only guaranteed to converge for certain classes of matrices.
- We will also find in some methods that it is straightforward to find a search direction but determining how far to go in that direction is not known.

There are two basic types of iterative methods:

1. Stationary methods. These are methods where the data in the equation to find $\vec{x}^{k+1}$ remains fixed; they have the general form

$$\vec{x}^{k+1} = P\vec{x}^{k} + \vec{c} \qquad \text{for fixed matrix } P \text{ and a fixed vector } \vec{c}$$

   We call P the iteration matrix.

2. Nonstationary methods. These are methods where the data changes at each iteration; these are methods of the form

$$\vec{x}^{k+1} = \vec{x}^{k} + \alpha_k \vec{p}^{k}$$

   Note here that the data, α_k and $\vec{p}^{k}$ change for each iteration k. Here $\vec{p}^{k}$ is called the search direction and α_k the step length.

Stationary Iterative Methods

The basic idea is to split (not factor) A as the sum of two matrices

$$A = M - N$$

where M is easily invertible. Then we have

$$A\vec{x} = \vec{b} \implies (M-N)\vec{x} = \vec{b} \implies M\vec{x} = N\vec{x} + \vec{b} \implies \vec{x} = M^{-1}N\vec{x} + M^{-1}\vec{b}$$

This suggests the iteration

$$\text{Given } \vec{x}^0 \text{ then} \qquad \vec{x}^{k+1} = M^{-1}N\vec{x}^k + M^{-1}\vec{b}, \quad k = 0, 1, 2, \ldots$$

which is a stationary method with $P = M^{-1}N$ and $\vec{c} = M^{-1}\vec{b}$.

There are 3 basic methods which make different choices for M and N. We will look at all three.

Jacobi Method

The simplest choice for M is a diagonal matrix because it is the easiest to invert. This choice leads to the Jacobi Method.

We write

$$A = L + D + U$$

where here L is the lower portion of A, D is its diagonal and U is the upper part. For the Jacobi method we choose $M = D$ and $N = -(L+U)$ so $A = M - N$ is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix} - \begin{pmatrix} 0 & -a_{12} & -a_{13} & \cdots & -a_{1n} \\ -a_{21} & 0 & -a_{23} & \cdots & -a_{2n} \\ -a_{31} & -a_{32} & 0 & \cdots & -a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_{n1} & -a_{n2} & -a_{n3} & \cdots & 0 \end{pmatrix}$$

Then our iteration becomes

$\vec{x}^{k+1} = -D^{-1}(L+U)\vec{x}^{k} + D^{-1}\vec{b}$ with iteration matrix $P = -D^{-1}(L+U)$

However, to implement this method we don't actually form the matrices rather

we look at the equations for each component. The point form (i.e., the equation for each component of $\vec{x}^{k+1}$) is given by the following:

Jacobi Method Given $\vec{x}^0$, then for $k = 0, 1, 2, \ldots$ find

$$x_i^{k+1} = -\frac{1}{a_{ii}}\Big(\sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^k\Big) + \frac{1}{a_{ii}}b_i$$

This corresponds to the iteration matrix $P_{\mathrm{J}} = -D^{-1}(L+U)$

Example Apply the Jacobi Method to the system $A\vec{x} = \vec{b}$ where

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 2 \\ 36 \\ 25 \end{pmatrix}$$

with an exact solution $(1, 2, 3)^T$. Use $\vec{x}^0 = (0, 0, 0)^T$ as a starting guess.

We find each component by our formula above.

$$x_1^1 = -\frac{1}{1}(0) + \frac{2}{1} = 2$$

$$x_2^1 = -\frac{1}{20}(0) + \frac{36}{20} = 1.8$$

$$x_3^1 = -\frac{1}{10}(0) + \frac{25}{10} = 2.5$$

Continuing in this manner we get the following table

k	x_1^k	x_2^k	x_3^k
0	0	0	0
1	2	1.8	2.5
2	0.9	1.85	3.06
3	1.36	2.016	2.96
5	1.12	2.010	2.98
10	0.993	1.998	3.00

The Gauss-Seidel Method

The Gauss-Seidel method is based on the observation that when we calculate, e.g., x_1^{k+1}, it is supposedly a better approximation than x_1^k so why don't we use it as soon as we calculate it. The implementation is easy to see in the point form of the equations.

Gauss-Seidel Method Given $\vec{x}^0$, then for $k = 0, 1, 2, \ldots$ find

$$x_i^{k+1} = -\frac{1}{a_{ii}}\Big(\sum_{j=1}^{i-1} a_{ij}x_j^{k+1}\Big) - \frac{1}{a_{ii}}\Big(\sum_{j=i+1}^{n} a_{ij}x_j^k\Big) + \frac{1}{a_{ii}}b_i$$

This corresponds to the iteration matrix $P_{\mathrm{GS}} = -(D+L)^{-1}U$

We now want to determine the matrix form of the method so we can determine the iteration matrix P. We note that the point form becomes

$$\vec{x}^{k+1} = -D^{-1}\Big(L\vec{x}^{k+1}\Big) - D^{-1}\Big(U\vec{x}^k\Big) + D^{-1}\vec{b}$$

Now grouping the terms at the $(k+1)$st iteration on the left and multiplying through by D we have

$$D\vec{x}^{k+1} + \left(L\vec{x}^{k+1}\right) = -\left(U\vec{x}^{k}\right) + \vec{b}$$

or

$$(D+L)\vec{x}^{k+1} = -U\vec{x}^{k} + \vec{b} \implies P = -(D+L)^{-1}U$$

Example Apply the Gauss-Seidel method to the system in the previous example, i.e.,

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 2 \\ 36 \\ 25 \end{pmatrix}$$

with an exact solution $(1,2,3)^T$. Use $\vec{x}^0 = (0,0,0)^T$ as a starting guess.

We find each component by our formula above.

$$x_1^1 = -\frac{1}{1}(0) + \frac{2}{1} = 2$$

$$x_2^1 = -\frac{1}{20}(2*2) + \frac{36}{20} = -.2 + 1.8 = 1.6$$

$$x_3^1 = -\frac{1}{10}\big((-1)2 - 2(1.6)\big) + \frac{25}{10} = .52 + 2.5 = 3.02$$

Continuing in this manner we get the following table

k	x_1^k	x_2^k	x_3^k
0	0	0	0
1	2	1.6	3.02
2	1.82	1.92	3.066
3	1.226	1.984	3.0194
5	1.0109	1.99	3.001

So for this example, the Gauss-Seidel method is converging much faster because remember for the fifth iteration of Jacobi we had $(1.12, 2.01, 2.98)^T$ as our approximation.

The SOR Method

The Successive Over Relaxation (SOR) method takes a weighted average between the previous iteration and the result we would get if we took a Gauss-Seidel step. For a choice of the weight, it reduces to the Gauss-Seidel method.

SOR Method Given $\vec{x}^0$, then for $k = 0, 1, 2, \ldots$ find

$$x_i^{k+1} = (1-\omega)\vec{x}^k + \omega\Big[-\frac{1}{a_{ii}}\Big(\sum_{j=1}^{i-1} a_{ij}x_j^{k+1}\Big) - \frac{1}{a_{ii}}\Big(\sum_{j=i+1}^{n} a_{ij}x_j^k\Big) + \frac{1}{a_{ii}}b_i\Big]$$

where $0 < \omega < 2$. This corresponds to the iteration matrix $P_{\text{SOR}} = (D+\omega L)^{-1}\Big((1-\omega)D - \omega U\Big)$

We first note that if $\omega = 1$ we get the Gauss-Seidel method. If $\omega > 1$ then we say that we are over-relaxing and if $\omega < 1$ we say that we are under-relaxing. Of course there is a question as to how to choose ω which we will address shortly.

We need to determine the iteration matrix for SOR. From the point form we have

$$D\vec{x}^{k+1} + \omega L\vec{x}^{k+1} = (1-\omega)D\vec{x}^{k} - \omega U\vec{x}^{k} + \vec{b}$$

which implies

$$\vec{x}^{k+1} = (D+\omega L)^{-1}\Big((1-\omega)D - \omega U\Big)\vec{x}^{k} + (D+\omega L)^{-1}\vec{b}$$

so that $P = (D+\omega L)^{-1}\Big((1-\omega)D - \omega U\Big)$.

Example Let's return to our example and compute some iterations using different values of ω. Recall that

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 2 \\ 36 \\ 25 \end{pmatrix}$$

We find each component by our formula above. Using $\omega = 1.1$ and $\omega = 0.9$ we have the following results.

	$\omega = 1.1$				$\omega = 0.9$		
k	x_1^k	x_2^k	x_3^k		x_1^k	x_2^k	x_3^k
0	0	0	0		0	0	0
1	2.2	1.738	3.3744		1.8	1.458	2.6744
2	1.862	1.9719	3.0519		1.7626	1.8479	3.0087
3	1.0321	2.005	2.994		1.3579	1.9534	3.0247
5	0.9977	2.0000	2.9999		1.0528	1.9948	3.0051

Example As a last example consider the system $A\vec{x} = \vec{b}$ where

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -1 & 4 \\ 3 & 4 & 5 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 13 \\ 13 \\ 26 \end{pmatrix}$$

and apply Jacobi, Gauss-Seidel and SOR with $\omega = 1.1$ with an initial guess of $(1, 1, 1)$. The exact solution is $(1, 2, 3)^T$.

	Jacobi			Gauss-Seidel			SOR		
k	x_1^k	x_2^k	x_3^k	x_1^k	x_2^k	x_3^k	x_1^k	x_2^k	x_3^k
1	1	1	1	1	1	1	1	1	1
1	4.5	-8.0	3.8	4.5	-4.5	6.1	4.85	-4.67	6.52
2	4.8	6.7	8.9	-.4	11	-3.36	-1.53	13.18	-5.52
3	-10.2	27.4	-3.04	6.04	-20.4	17.796	9.16	-29.84	26.48

As you can see from these calculations, all methods fail to converge even though A was symmetric.

Stopping Criteria

One complication with iterative methods is that we have to decide when to terminate the iteration.

One criteria that is often used is to make sure that the residual $\vec{r}^k = \vec{b} - A\vec{x}^k$ is sufficiently small. Of course, the actual size of $\|\vec{r}^k\|$ is not as important as its relative size. If we have a tolerance τ, then one criteria for termination is

$$\frac{\|\vec{r}^k\|}{\|\vec{r}^0\|} < \tau$$

where $\|\vec{r}^0\|$ is the initial residual $\vec{b} - A\vec{x}^0$.

The problem with this criterion is that it depends on the initial iterate and may result in unnecessary work if the initial guess is too good and an unsatisfactory approximation $\vec{x}^k$ if the initial residual is large. For these reasons it is usually

better to use

$$\frac{\|\vec{r}^k\|}{\|\vec{b}\|} < \tau$$

Note that if $\vec{x}^0 = \vec{0}$, then the two are identical.

Another stopping criteria is to make sure that the difference in successive iterations is less than some tolerance; again the magnitude of the actual difference is not as important as the relative difference. Given a tolerance σ we could use

$$\frac{\|\vec{x}^{k+1} - \vec{x}^k\|}{\|\vec{x}^{k+1}\|} \leq \sigma$$

Often a combination of both criteria are used.

Analyzing the Convergence of Stationary Methods

From a previous example, we saw that the methods don't work for any matrix (even a symmetric one) so we want to determine when the methods are guaranteed to converge. Also we need to know if, like nonlinear equations, the methods are sensitive to the starting vector.

We start with a convergence theorem for the general stationary method $\vec{x}^{k+1} = P\vec{x}^k + \vec{c}$. We see that the eigenvalues of the iteration matrix play a critical role in the convergence.

Theorem Let P be an $n \times n$ matrix and assume there is a unique $\vec{x}^*$ such that $\vec{x}^* = P\vec{x}^* + \vec{c}$. Then the stationary iteration

$$\text{Given } \vec{x}^0 \text{ find} \quad \vec{x}^{k+1} = P\vec{x}^k + \vec{c} \quad \text{for } k = 0, 1, 2, \ldots$$

converges to $\vec{x}^*$ if and only if $\rho(P) < 1$.

This says that if there is a unique solution, then our iteration will converge to that solution if $\rho(P) < 1$; moreover, if $\rho(P) \geq 1$ it will not converge.

Example Let's compute the spectral radius of the iteration matrices in our two examples. For

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 20 & -2 \\ -1 & -2 & 10 \end{pmatrix}$$

We have that P_{J} for the Jacobi matrix is $-D^{-1}(L+U)$ which is just

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -\frac{1}{20} & 0 \\ 0 & 0 & -\frac{1}{10} \end{pmatrix} \begin{pmatrix} 0 & 2 & -1 \\ 2 & 0 & -2 \\ -1 & -2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -2 & 1 \\ -0.1 & 0 & 0.1 \\ 1 & 0.2 & 0 \end{pmatrix}$$

We compute its eigenvalues to be -0.62, 0.4879 and 0.1322 so $\rho(P_{\mathrm{J}}) = 0.62 < 1$.

For the second example

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & -1 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

so the iteration matrix for the Jacobi method is $-D^{-1}(L+U)$

$$\begin{pmatrix} 0 & -0.5 & -1.5 \\ 1 & 0 & 4 \\ -0.6 & -0.8 & 0 \end{pmatrix}$$

which has eigenvalues 0.72, -0.3612+1.786i and thus $\rho(P_{\mathrm{J}}) = 1.8 \geq 1$.

We can also calculate the iteration matrix for Gauss-Seidel. The Gauss-Seidel

iteration matrix is $-(D+L)^{-1}U$ so for the first matrix we have

$$P_{\mathrm{GS}} = \begin{pmatrix} 0 & -2 & 1 \\ 0 & 0.2 & 0 \\ 0 & -.16 & 0.1 \end{pmatrix} \quad \text{with eigenvalues 0, 0.1, 0.2}$$

so $\rho(P_{\mathrm{GS}}) = 0.2 < 1$. For the second matrix

$$P_{\mathrm{GS}} = \begin{pmatrix} 0 & -0.5 & -1.5 \\ 0 & -0.5 & 2.5 \\ 0 & 0.7 & -1.1 \end{pmatrix} \quad \text{with eigenvalues 0, 0.5565 and -2.1565}$$

so $\rho(P_{\mathrm{GS}}) = 2.1565 > 1$ and the iteration fails.

Note that the iteration matrix is often singular but we are not concerned about its smallest eigenvalue but rather its largest in magnitude. Similar results hold for P_{SOR}.

Now let's return to the theorem to see why the spectral radius of P governs the convergence.

We have

$$\vec{x}^{k+1} - \vec{x}^* = P\vec{x}^k + \vec{c} - (P\vec{x}^* + \vec{c}) = P\big(\vec{x}^k - \vec{x}^*\big)$$

because $\vec{x}^*$ satisfies the equation $\vec{x}^* = P\vec{x}^* + \vec{c}$. Now if we apply this equation recursively we have

$$\vec{x}^{k+1} - \vec{x}^* = P\big[(P\vec{x}^{k-1} + \vec{c}) - \big(P\vec{x}^* + \vec{c}\big)\big] = P^2\big(\vec{x}^{k-1} - \vec{x}^*\big) = \cdots = P^{k+1}(\vec{x}^0 - \vec{x}^*)$$

Taking the norm of both sides gives

$$\|\vec{x}^{k+1} - \vec{x}^*\| = \|P^{k+1}(\vec{x}^0 - \vec{x}^*)\| \leq \|P^{k+1}\|\|\vec{x}^0 - \vec{x}^*\|$$

for any induced matrix norm. This says that the norm of the error at the $(k+1)$st iteration is bounded by $\|P^{k+1}\|$ times the initial error. Now $\|\vec{x}^0 - \vec{x}^*\|$ is some fixed number so when we take the limit as $k \to \infty$ of both sides, we would like the right hand side to go to zero, i.e., $\lim_{k\to\infty} \|P^{k+1}\| = 0$. Recall that when we compute the limit as $k \to \infty$ of $|\sigma^k|$ (for scalar σ) then it goes to 0 if $|\sigma| < 1$ or to infinity if $|\sigma| > 1$. The following lemma tells us the analogous result for powers of matrices.

Lemma. The following statements are equivalent for an $n \times n$ matrix A.

(i) $\lim_{n\to\infty} A^n = 0$ (we say A is convergent; here 0 denotes the zero matrix)

(ii) $\lim_{n\to\infty} \|A^n\| = 0$ for any induced matrix norm

(iii) $\rho(A) < 1$

Thus in our result we have

$$\lim_{k\to\infty} \|\vec{x}^{k+1} - \vec{x}^*\| \leq \lim_{k\to\infty} \|P^{k+1}\| \|\vec{x}^0 - \vec{x}^*\|$$

and the right hand side goes to zero if and only if $\rho(P) < 1$. ■

Note that this result suggests why we got faster convergence in our example with Gauss-Seidel than Jacobi. For the Jacobi $\rho(P_{\mathrm{J}}) = 0.62$ and for Gauss-Seidel $\rho(P_{\mathrm{GS}}) = 0.2$; clearly the smaller $\rho(P)$ is, the faster we should expect convergence to be.

We present two results which give classes of matrices for which convergence is guaranteed. Remember that if we have a result for SOR, then it automatically holds for Gauss-Seidel.

Diagonal dominance theorem Assume that A is (strictly) diagonally dominant. Then both the Jacobi iterates and the Gauss-Seidel iterations converge to $A^{-1}\vec{b}$ for any $\vec{x}^0$.

Ostrowski-Reich Theorem Let A be a symmetric positive definite matrix and assume that $0 < \omega < 2$. Then the SOR iterates converge to $A^{-1}\vec{b}$ for any $\vec{x}^0$.

Note that these results tell us that, unlike iterative methods for nonlinear equations, iterative methods for linear equations are not sensitive to the initial guess.

These methods have limited use for solving linear systems today because they

only work for certain matrices. One advantage though is they are very simple to implement and if you have a matrix for which they are guaranteed to converge, they are simple to implement. However, they are still popular for use in preconditioning a system (more on this later).

In recent years, nonstationary methods have become the most popular iterative solvers.

Nonstationary Iterative Methods

Recall that nonstationary iterative methods have the general form

$$\vec{x}^{k+1} = \vec{x}^{k} + \alpha_k \vec{p}^{k}$$

Here $\vec{p}^{k}$ is called the search direction and α_k is the step length.

Unlike stationary methods, nonstationary methods do not have an iteration matrix.

The classic example of a nonstationary method is the Conjugate Gradient (CG) method which is an example of a Krylov space method. However, CG only works for symmetric positive definite matrices but there have been many variants of it (e.g., BICG, BICGSTAB) developed which handle even indefinite matrices.

What is the idea behind Krylov space methods?

We create a sequence of Krylov subspaces for $A\vec{x} = \vec{b}$ where

$$\mathcal{K}_n = \text{span}(\vec{b}, A\vec{b}, A^2\vec{b}, \ldots, A^{n-1}\vec{b})$$

and we look for our solution in the space spanned by these n vectors. We note that this space is formed by matrix-vector products.

Due to time limitations, we will only describe CG here. We begin by describing the minimization properties of the CG iterates. To understand CG we view it as a natural modification to the intuitive Steepest Descent Algorithm for minimizing a function. A nice (free) description by J. Shewchuk of the CG method is available on the web at http://www.cs.cmu.edu/~ quake-papers/painless-conjugate-gradient.pdf

Consider the quadratic functional

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

where A and $\vec{b}$ are given; i.e., ϕ is a map from $\mathbb{R}^n$ to $\mathbb{R}^1$. The following lemma tells us that for A symmetric positive definite, we have a choice of solving the linear system $A\vec{x} = \vec{b}$ or minimizing the given nonlinear functional $\phi(\vec{y})$.

Proposition. If A is an $n \times n$ symmetric positive definite matrix then the solution $\vec{x} = A^{-1}\vec{b}$ is equivalent to

$$\vec{x} = \min_{\vec{y} \in I\!R^n} \phi(\vec{y}) \quad \text{where} \quad \phi(\vec{y}) = \frac{1}{2}\vec{y}^T A\vec{y} - \vec{b}^T\vec{y}$$

Example Consider the linear system $A\vec{x} = \vec{b}$ where

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$$

The solution to this equation is (2,-2). Let's look at the quadratic function $\phi(\vec{x})$

$$\phi(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x} = \frac{1}{2}\left[3x_1^2 + 4x_1x_2 + 6x_2^2\right] - 2x_1 + 8x_2$$

and graph it. We will plot the surface and the level curves (i.e., where ϕ is a constant value) and see that the minimum is at $(2, -2)$. Next we compute the

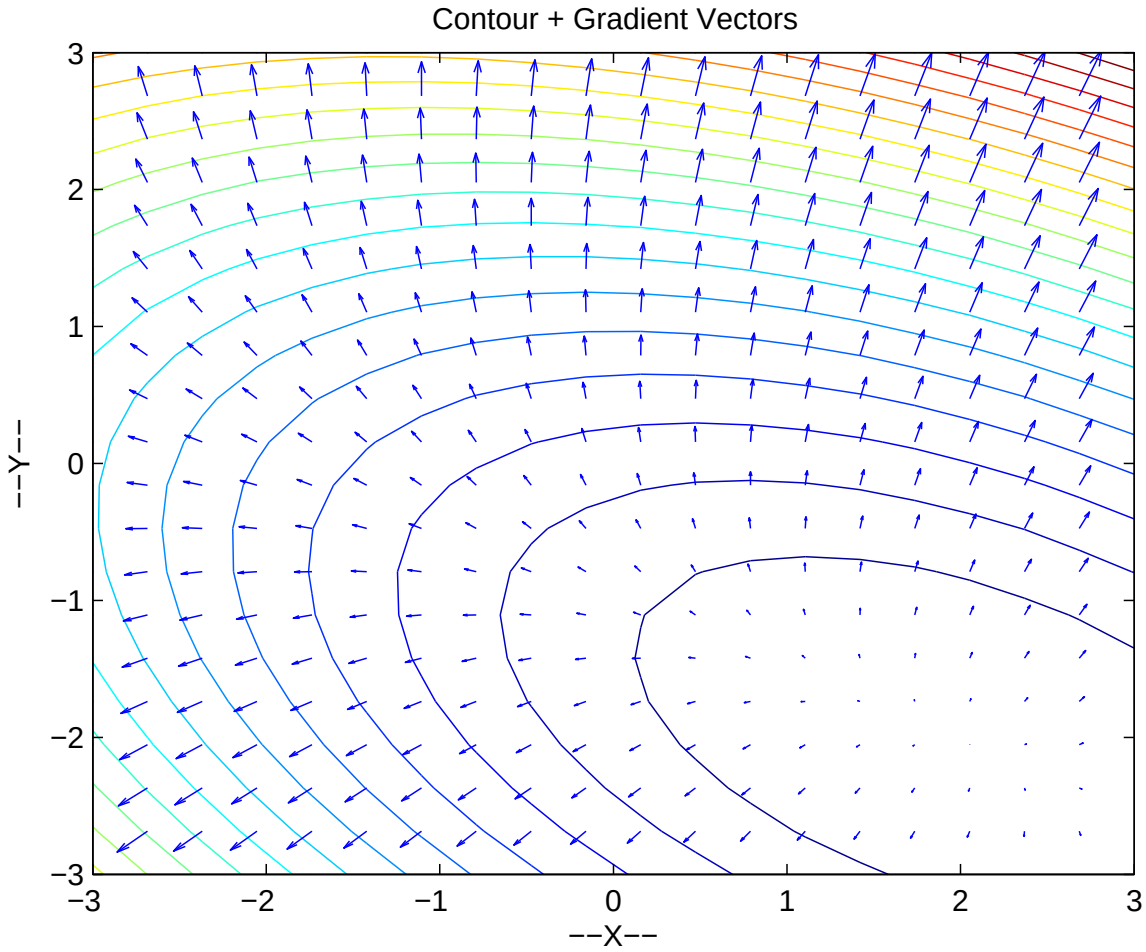
gradient of ϕ

$$\nabla\phi = \begin{pmatrix} \frac{\partial\phi}{\partial x_1} \\ \frac{\partial\phi}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 3x_1 + 2x_2 - 2 \\ 2x_1 + 6x_2 + 8 \end{pmatrix} = A\vec{x} - \vec{b}$$

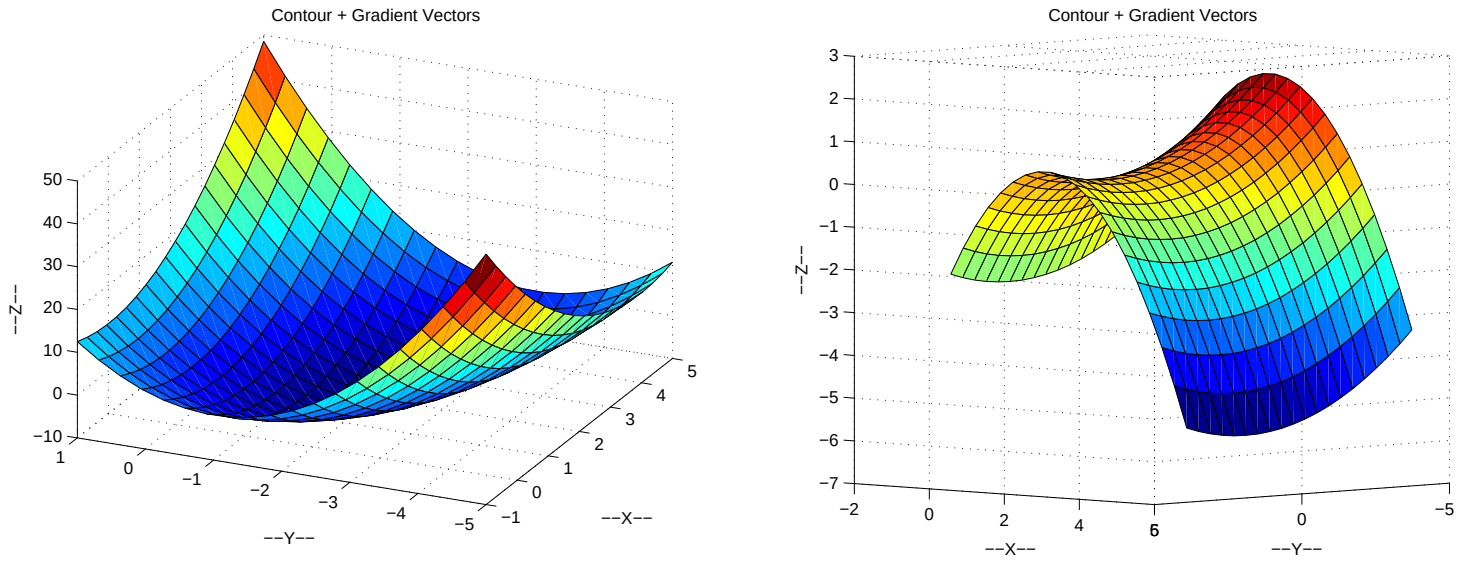
and plot it on the level curves. What do you notice?

For our example A is symmetric positive definite. In the figure below we illustrate the quadratic form for A as well as for an indefinite matrix; for a negative definite system we could just plot $-\phi$ for our choice of ϕ. Notice that we have a saddle point in the indefinite case. This helps us to understand why minimizing the quadratic form works for the case A is symmetric positive definite.

To understand CG it is helpful to see it as a modification to the intuitive method Steepest Descent which is based on a simple fact from calculus. If we want to minimize a function f then we know that $-\nabla f$ points in the direction of the maximum decrease in f. So a simple iterative method to minimize a function is to start at a point, compute the gradient of your function at the point and take a step in the direction of minus the gradient. Of course, we have to determine



how far to go in that direction. It turns out that if the function to minimize is quadratic (as in our case) then we can determine an explicit formula for α_k, the step length.



Because $-\nabla\phi(\vec{x}) = \vec{b} - A\vec{x}$ (i.e., the residual) then we choose the residual to be the search direction. We start by choosing an initial guess $\vec{x}^0$, compute the residual $\vec{r}^0 = \vec{b} - A\vec{x}^0$ and then use $\vec{x}^1 = \vec{x}^0 + \alpha_0\vec{r}^0$ to obtain the next iterate. Thus all we have to do is determine the step length α_0 and we will have the Steepest Descent algorithm for solving $A\vec{x} = \vec{b}$. For our quadratic functional ϕ we can determine the optimal step length but when applying Steepest Descent to minimize a general function, it is not possible to determine the optimal step length. In ACS II you will probably look at step length selectors when you study optimization.

So how do we choose α_0? We would like to move along our direction $\vec{r}^0$ until we find where $\phi(\vec{x}^1)$ is minimized; i.e., we want to do a line search. This means that the directional derivative (in the direction of $\vec{r}^0$) is zero

$$\nabla\phi(\vec{x}^1) \cdot \vec{r}^0 = 0$$

Note that this says that $\nabla\phi(\vec{x}^1)$ and $\vec{r}^0$ are orthogonal. But we know that $-\nabla\phi(\vec{x})$ is the residual so $\nabla\phi(\vec{x}^1) = \vec{r}^1$. Thus the residuals $\vec{r}^1$ and $\vec{r}^0$ are orthogonal. We can use this to determine α_0.

$$\vec{r}^{1^T}\vec{r}^0 = 0 \implies (\vec{b} - A\vec{x}^1)^T\vec{r}^0 = 0 \implies \left(\vec{b} - A(\vec{x}^0 + \alpha_0\vec{r}^0)\right)^T\vec{r}^0 = 0$$

Solving this expression for α_0 gives

$$(\vec{b} - A\vec{x}^0)^T\vec{r}^0 = \alpha_0\vec{r}^{0^T}A^T\vec{r}^0 \implies \alpha_0 = \frac{\vec{r}^{0^T}\vec{r}^0}{\vec{r}^{0^T}A\vec{r}^0}$$

where we have used the fact that A is symmetric.

Steepest Descent for Solving $A\vec{x} = \vec{b}$ Given $\vec{x}^0$, compute $\vec{r}^0 = \vec{b} - A\vec{x}^0$. Then for $k = 0, 1, 2, \ldots$

$$\alpha_k = \frac{\vec{r}^{k^T}\vec{r}^k}{\vec{r}^{k^T}A\vec{r}^k}$$
$$\vec{x}^{k+1} = \vec{x}^k + \alpha_k \vec{r}^k$$
$$\vec{r}^{k+1} = \vec{b} - A\vec{x}^{k+1}$$

If we draw our path starting at $\vec{x}^0$ then each segment will be orthogonal because $\vec{r}^{k^T}\vec{r}^{k-1} = 0$. So basically at any step k we are going along the same line as we did at step $k-2$; we are just using $\pm$ the two search direction over and over again. The first modification to this algorithm would be to see if we could choose a set of mutually orthogonal directions; unfortunately we are not able to determine a formula for the step length in this case so what we do is to choose the directions mutually A-orthogonal. To see how to do this, we first note an

interesting result about when we can get "instant" convergence with Steepest Descent.

Let $(\lambda_i, \vec{v}_i)$ be eigenpairs of A so that $A\vec{v}_i = \lambda_i \vec{v}_i$. First let's see what happens if the initial guess $\vec{x}^0 = \vec{x} - \vec{v}_i$, i.e., the initial error is an eigenvector. In this case Steepest Descent converges in 1 iteration! To see this first compute the initial residual

$$\vec{r}^0 = \vec{b} - A\vec{x}^0 = \vec{b} - A\big(\vec{x} - \vec{v}_i\big) = \vec{b} - \vec{b} - \lambda_i \vec{v}_i = \lambda_i \vec{v}_i$$

and note that $\alpha_0 = 1/\lambda_i$ from our formula and then

$$\vec{x}^1 = \vec{x}^0 + \frac{1}{\lambda_i}\vec{r}^0 = \vec{x}^0 + \frac{1}{\lambda_i}\lambda_i \vec{v}_i = \vec{x}^0 + \vec{v}_i \implies \vec{x}^1 - \vec{x}^0 = \vec{v}_i \implies \vec{x}^1 - (\vec{x} - \vec{v}_i) = \vec{v}_i$$

and thus $\vec{x}^1 = \vec{x}$, the solution to $A\vec{x} = \vec{b}$.

One can also show that if λ_1 has a.m. and g.m. n then once again we get convergence (where $\alpha = 1/\lambda_1$) in one step. When this is not the case, then α_i becomes a weighted average of the $1/\lambda_i$.

To modify Steepest Descent we would probably try to choose a set of n mutually *orthogonal* search directions $\vec{d^i}$; in each search direction, we take exactly one step, and we want that step to be just the right length. If our search directions were the coordinate axes, then after the first step we would have the x_1 component determined, after the second step we would know the second component so after n steps we would be done because we make the search direction $\vec{d^i}$ orthogonal to the error vector $\vec{e}_{i+1}$. This makes us think it is a direct method though! However, remember that we are not using exact arithmetic so the directions may not remain orthogonal.

Unfortunately, this doesn't quite work because we can't compute the step length unless we know the exact error which would mean we know the exact solution. What actually works is to make the vectors mutually A orthogonal or conjugate; that is, for search directions $\vec{d^k}$ we want

$$\vec{d^i}^T A\vec{d^j} = 0 \quad \text{for } i \neq j$$

We require that the error vector $\vec{e}_{i+1}$ be A-orthogonal to the search direction $\vec{d^i}$, i.e., $\vec{e}_{i+1}^T A\vec{d^i} = 0$ which the following demonstrates is equivalent to minimizing

$\phi(\vec{x}^{i+1})$ along the search direction $\vec{d^i}$. Setting the directional derivative to zero

$$\nabla\phi(\vec{x}^{i+1}) \cdot \vec{d^i} = 0 \implies \vec{r^{i+1}}^T \vec{d^i} = 0 \implies (\vec{b}^T - \vec{x^{i+1}}^T A)\vec{d^i} = 0$$

and using the fact that $A\vec{x} = \vec{b}$ and A is symmetric we have

$$(\vec{b}^T - \vec{x^{i+1}}^T A - \vec{b}^T + \vec{x}^T A)\vec{d^i} = 0 \implies \vec{e}_{i+1} A\vec{d^i} = 0$$

Now to calculate the step length we use this condition to get

$$\vec{d^i}^T A\vec{e}_{i+1} = 0 \implies \vec{d^i}^T A(\vec{x} - \vec{x}^{i+1}) = \vec{d^i}^T A(\vec{x} - \vec{x}^i - \alpha_i\vec{d^i}) = 0$$

and using the fact that $A\vec{e}_i = A(\vec{x} - \vec{x}^i) = \vec{b} - A\vec{x}^i = \vec{r}^i$ we have

$$\vec{d^i}^T \vec{r^i} - \alpha_i \vec{d^i}^T A\vec{d^i} = 0 \implies \alpha_i = \frac{\vec{d^i}^T \vec{r^i}}{\vec{d^i}^T A\vec{d^i}}$$

So our algorithm will be complete if we can determine a way to calculate a set of A-orthogonal search directions $\vec{d^i}$. Recall that if we have a set of n linearly independent vectors then we can use the Gram Schmidt method to create a set of orthogonal vectors. We can use a modification of this method to produce a set of A-orthogonal vectors. We start with a set of linearly independent vectors $\vec{u}^i$ and set $\vec{d^1} = \vec{u}^1$. Then for $\vec{d^2}$ we subtract off the portion of $\vec{u}^2$ which is

not A-orthogonal to $\vec{d}^1$; for $\vec{d}^3$ we subtract off the portion of $\vec{u}^3$ which is not A-orthogonal to $\vec{d}^1$ and the portion not A-orthogonal to $\vec{d}^2$. Proceeding in this manner we can determine the search directions. This approach is called the method of Conjugate Directions and was never popular because of the cost (time and storage) in determining these search directions. Developers of the CG method found a way to efficiently calculate these search directions to make the method computationally feasible.

The idea in CG is to use the residual vectors $\vec{r}^i$ for the $\vec{u}^i$ in the Gram Schmidt method. Recall that the residuals have the property that each residual is orthogonal to the previous residuals

$$\vec{r}^{iT}\vec{r}^j = 0 \quad \text{for } i \neq j$$

Moreover, $\vec{r}^i$ is a linear combination of $\vec{r}^{i-1}$ and $A\vec{d}^{i-1}$ because

$$\vec{r}^i = b - A\vec{x}^i = \vec{b} - A(\vec{x}^{i-1} + \alpha_{i-1}\vec{d}^{i-1} = \vec{r}^{i-1} - \alpha_{i-1}A\vec{d}^{i-1}$$

Our search space is just

$$\mathcal{D}_i = \{\vec{d}^0, A\vec{d}^0, A^2\vec{d}^0 \ldots A^{i-1}\vec{d}^0\} = \{\vec{r}^0, A\vec{r}^0, A^2\vec{r}^0 \ldots A^{i-1}\vec{r}^0\}$$

which you may recall is our definition of a Krylov space. This means that

$A\mathcal{D}_i \subset \mathcal{D}_{i+1}$ and because the next residual $\vec{r}^{i+1^T}$ is orthogonal to $\mathcal{D}_{i+1}$ we have $\vec{r}^{i+1}A\mathcal{D}_{i+1} = 0$ and thus the residual vectors are already A-orthogonal!

Recall that we are assuming that A is symmetric and positive definite. Therefore, we know that A has a complete set of orthogonal (and thus linearly independent) eigenvectors.

Conjugate Gradient Method $\vec{x}^0 = 0$; compute $\vec{r}^0 = \vec{b} - A\vec{x}^0$

for $k = 1, 2, 3, \ldots$

$\rho_{k-1} = \vec{r}^{k-1^T}\ \vec{r}^{k-1}$

if $k = 1$ then $\vec{d}^1 = \vec{r}^0$

else $\beta_{k-1} = \frac{\rho_{i-1}}{\rho_{i-2}}$; $\quad \vec{d}^k = \vec{r}^k + \beta_{k-1}\vec{d}^{k-1}$

$\vec{q}^k = A\vec{d}^k$

$$\alpha_k = \frac{\rho^{k-1}}{\vec{d}^{k^T}\vec{q}^k}$$

$\vec{x}^k = \vec{x}^{k-1} + \alpha_k \vec{d}^k$

$\vec{r}^k = \vec{r}^{k-1} - \alpha_k \vec{q}^k$

check for convergence

Summary of properties of CG:

1. The residuals are orthogonal

$$\vec{r}^{k^T}\vec{r}^{j} = 0 \quad \text{for } j < k$$

2. The search directions are A-orthogonal

$$\vec{d}^{k^T} A\vec{d}^{j} = 0 \quad \text{for } j < k$$

3. If exact arithmetic is used, the method will get the exact solution in n steps

Example Let's return to our example where

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$$

Applying CG with an initial guess $\vec{x}^0 = (1, 1)^T$ we get

$\vec{r}^0$	$\vec{r}^1$	$\vec{r}^2$	$\vec{d}^0$	$\vec{d}^1$	$\vec{x}^0$	$\vec{x}^1$	$\vec{x}^2$
-3	3.1909	10^{-15}	-3	3.0716	1	0.547	2
-16	-0.5983	10^{-15}	-16	-1.2346	1	-1.416	-2

When we compute $\vec{r}^{0^T}\vec{r}^{1}$ we get 10^{-15} and similarly for $\vec{r}^{1}$.

In this simple example we only have 2 steps but if n is large then our vectors can lose A-orthogonality so that it doesn't converge to the exact answer in n steps.

Storing a sparse matrix

If we have a large sparse $n \times n$ system and either there is no structure to the zero entries or it is banded with a lot of zeros inside the bandwidth then we can just store the nonzero entries in the matrix in a single vector, say $a(1 : n\_nonzero)$

How can we determine where an entry in this array belongs in the original matrix?

We have two pointer (integer) arrays. The first is an n-vector which gives the index in the array a where each row starts. The second pointer array is dimensioned by $n\_nonzero$ and indicates the column location in the original matrix.

As an example, consider the matrix and its sparse storage

$$A = \begin{pmatrix} 1 & 0 & 3 & 0 & 0 & 2 \\ 0 & 4 & 7 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 0 \end{pmatrix} \qquad \vec{a} = (1, 3, 2, 4, 7, 2, 1)^T$$

We set up two pointer arrays; the array `i_row` and `i_col` is

$$\texttt{i\_row} = (1, 4, 6, 8)^T \qquad \texttt{i\_col} = (1, 3, 6, 2, 3, 3, 5)^T$$

Then if we access, e.g., $a(5) = 7$, then from the pointer `i_row` we know its in the second row and from the column pointer we know $\texttt{i\_col}(5) = 3$ so it corresponds to $A(2, 3)$.

Preconditioning

If we have a linear system $A\vec{x} = \vec{b}$ we can always multiply both sides by a matrix M^{-1}. If we can find a matrix M^{-1} such that our iterative method converges faster then we are preconditioning our problem.

The basic idea is to find a matrix M^{-1} which improves the spectral properties of the coefficient matrix.

One possibility for a preconditioner is to find a matrix M^{-1} which approximates A^{-1}. Another is to devise a matrix M that approximates A; oftentimes something like an incomplete LU factorization is used.

There is a huge body of literature on this topic and the interested reader is referred to either taking Dr. Gallivan's linear algebra course in the Mathematics department or Chapter 3 of the Templates reference given at the first of these notes.