261: 31st January

The conjugate gradient algorithm

Recap

Last week, we saw that some matrix equations were too large to solve using normal techniques. In light of this, we studied an alternative approach, in which an iterative algorithm takes a crude, approximate answer and repeatedly refines it. With each iteration of the algorithm, the answer grows closer to the true solution.

It was apparent that the length of time needed to solve the equation is determined by how exact we require the answer to be; the more precision we demand, the greater the number of iterations that we need.

The advantage of this technique is that each iteration is quite simple and quick, so fairly accurate answers can be calculated quickly. However, we also saw that the true solution cannot be calculated in a finite length of time.

This week, we shall look at an enhanced version of this algorithm. It deserves our attention because it can calculate the exact answer in a finite number of iterations. This is the *Conjugate Gradient* method.

The starting point

We will use the same starting point as last week. $Ax = b$ is our matrix equation and we want to solve it for x. We use the equation to construct the quadratic function $f(x) = \frac{1}{2}x^T Ax - b^T x + c$, where c is a constant, whose value is irrelevant.

If A is symmetric and positive definite and x has two components, f(x) is a surface shaped like a bowl. If we substitute the solution to $Ax = b$ into $f(x)$, then $f(x) = c$. However, if $x_i \neq x$, then $f(x_i) > c$. So solving the equation becomes a question of trying to find the quickest way down to the bottom of the bowl.

Last week, we saw how to start at an arbitrary point on the surface and descend down it. But the algorithm that we used meant that we were zig-zagging back and forth across the surface. As we descended further and further, the steps became smaller and smaller and converged on the bottom of the bowl.

The failings in this technique is that, by zig-zagging, we are descending in more-or-less the same direction every time we zig and again every time we zag. The algorithm would be a bit more useful if we could modify it so that we only went in each direction once.

If we could get the distances right, we could reach the solution with just one zig and one zag.

This is what the conjugate gradient method does.

The algorithm

Firstly, we need to define a few quantities. e_i is the error vector and tells us how far we are from the true solution, after the ith step. It is equal to $x_i - x$, where x_i is our calculated solution after the ith step and x is the true solution to $Ax = b$.

The direction that takes us down the bowl most steeply is $-\underline{\nabla} f(x_i)$ and we label this r_i. Fortunately, for a symmetric matrix A, $\underline{\nabla} f(x_i) = Ax_i - b$, so $r_i = b - Ax_i$.

Finally the direction in which we modify x_i is d_i, so $x_{i+1} = x_i + \alpha_i d_i$, where α_i is a scalar.

So, we would like the algorithm to start from an arbitrary value of x and take the same number of steps as there are components of x. One way to do this would be to use the unit vectors as directions and modify the starting value of x once in each direction. However, it turns out to be not that simple. We shall see that we have to define a new direction at each step to get the algorithm to work.

So lets run through the algorithm and try to work it out as we go.

We start with a guess at the solution, x_0, and pick a direction in which to modify x_0. The direction that is going to take us down the slope the quickest is r_0, so we will give a try. We set our direction vector, d_0, equal to r_0 and go a distance α_0 in that direction.

$$x_1 = x_0 + \alpha_0 d_0$$

Now, if we're only to travel once in each direction, then it would make sense to pick α_0 so that x_1 is as close as possible to x. This is the point where $e_1 = x_1 - x$ is perpendicular to the direction d_i.

$$d_0^T e_1 = o$$

$$\begin{aligned}
d_0^T (x_1 - x) &= 0 \\
d_0^T (x_0 + \alpha_0 d_0 - x) &= 0 \\
d_0^T (\alpha_0 d_0 + x_0 - x) &= 0 \\
d_0^T (\alpha_0 d_0 + e_0) &= 0 \\
d_0^T \alpha_0 d_0 &= -d_0^T e_0 \\
\alpha_0 &= -\frac{d_0^T e_0}{d_0^T d_0}
\end{aligned}$$

This would be fine, if we knew e_0. However, we don't because we don't know x. x is what we are trying to work out.

However, we do know Ae_0, because $Ae_0 = A(x_0 - x) = Ax_0 - b = -r_0$. So we might fair better if we try to find α_0 that makes e_1 A-orthoginal to d_0.

$$d_0^T A e_1 = 0$$

$$\begin{aligned}
d_0^T A(x_1 - x) &= 0 \\
d_0^T A(x_0 + \alpha_0 d_0 - x) &= 0 \\
d_0^T A \alpha_0 d_0 &= -d_0^T A e_0 \\
\alpha_0 &= \frac{-d_0^T A e_0}{d_0^T A d_0} \\
\alpha_0 &= \frac{d_0^T r_0}{d_0^T A d_0}
\end{aligned}$$

We know everything in this definition of α_0, so we can use this in our algorithm.

This derivation is the same for a general i, so we can say that

$$\alpha_i = \frac{d_i^T r_i}{d_i^T A d_i}$$

Now seems a good place to mention a few things about A-orthogonality, which is quite similar to conventional orthogonality. We can define a set of A-othogonal vectors, d_i, that are not perpendicular to each other. However, they will fulfil $d_j^T A d_i = \delta_{ji}$. The number of vectors in the set is the same as the number of dimensions in the space of x and so the vectors d_i still span the space, in the way that conventionally orthogonal vectors would.

So far we have x_0, α_0 and d_0, from which we can work out x_1, so we know that, too. The next thing we need to work out is the direction in which to change x_1 to get x_2; we need to work out d_1.

Unfortunately, it's not as simple as taking d_1 to be equal to r_1. This is what we did in the method of steepest descent and it resulted in too much zig-zagging. Instead, d_1 will comprise r_1 plus a term containing d_0.

So,

$$d_1 = r_1 + \beta_{10} d_0$$

In general, when we are calculating the ith direction vector, d_i contains r_i plus a combination of the previous d vectors.

$$d_i = r_i + \sum_{k=0}^{i-1} \beta_{ik} d_k$$

We need d_1 to be A-orthogonal to d_0 and we can use this fact to work out that value of the β_{ij}. For a general i,

$$d_i^T A d_j = r_i^T A d_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T A d_j$$

If $i > j$, then,

$$r_i^T A d_j + \beta_{ij} d_j^T A d_j = 0$$

$$\beta_{ij} = \frac{-r_i^T A d_j}{d_j^T A d_j}$$

Now we want to simplify the right hand side, but before we can do that, we must look at how r_i changes with each step.

When we update the position, from x_i to x_{i+1}, we add a bit to it, $\alpha_i d_i$. But, r_i is defined as $b - Ax_i$, so when we update x_i to x_{i+1}, we can update r_i to r_{i+1}, by adding a bit to it.

$$r_{i+1} = b - Ax_{i+1}$$

$$r_{i+1} = b - A(x_i + \alpha_i d_i)$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

If we take the product of r_i^T with r_{j+1}, and use the last line, then we can contribute to a neater expression for β_{ij}

$$r_i^T r_{i+1} = r_i^T r_j - \alpha_j r_i^T A d_j$$

$$\alpha_j r_i^T A d_j = r_i^T r_j - r_i^T r_{j+1}$$

$$r_i^T A d_j = \frac{r_i^T r_j - r_i^T r_{j+1}}{\alpha_i}$$

Now we are faced with our final conceptual hurdle. Imagine the bowl shaped surface, f and that we are modifying x_0 to x_1 in the direction d_0. At the point x_1, the direction d_0 is tangential to a horizontal elipse running around the surface. Whatsmore, we know that the direction of steepest slope is always perpendicular to the tangent line, so r_0 and r_1 must be perpendicular.

Putting it another way, earlier, we chose α_0 so that $d_0^T A e_1 = 0$. Substituting for Ae_1, this is $-d_0^T r_1 = 0$. If, d_0 is r_0, then this means that r_0 and r_1 must be perpendicular.

This is true at all i, so we can say that, in the last line

$$r_i^T A d_j = \begin{cases} \frac{r_i^T r_i}{\alpha_i} & i = j \\ -\frac{r_i^T r_i}{\alpha_{i-1}} & i = j+1 \end{cases}$$

So this the expression that we will use in the numerator for β_{ij}.

$$\beta_{i,j} = \begin{cases} \frac{r_i^T r_i}{\alpha_{i-1} d_{i-1}^T A d_{i-1}} & i = j+1 \\ 0 & i \neq j+1 \end{cases}$$

This gives us everything that we need to work out the new direction vector, d_1. If we put the algorithm together, it looks like this

$$d_0 = r_0 = b - Ax_0$$

$$\alpha_0 = \frac{r_0^T r_0}{d_0^T A d_0}$$

$$x_1 = x_0 + \alpha_0 d_0$$

$$r_1 = r_0 - \alpha_0 A d_0$$

$$\beta_{1,0} = \frac{r_1^T r_1}{r_0^T r_0}$$

where, in the last line, we have substituted for α_0.

$$d_1 = r_1 + \beta_{1,0} d_0$$

For a general system, at the ith step, the algorithm looks like,

$$d_0 = r_0 = b - Ax_0$$

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}$$

$$x_{i+1} = x_i + \alpha_i d_i$$

$$r_{i+1} = r_i - \alpha_i A d_i$$

$$\beta_{i,i-1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$$

$$d_{i+1} = r_{i+1} + \beta_{i,i-1} d_i$$

Questions

Using the Conjugate gradient algorithm, calculate the solution to the following

$$1)A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} b = \begin{pmatrix} 5 \\ 11 \end{pmatrix} \quad 2)A = \begin{pmatrix} 5 & 2 \\ 9 & 1 \end{pmatrix} b = \begin{pmatrix} 11 \\ 12 \end{pmatrix}$$