

This is basically a summary of [1]. He explains all the insights there, but I have trouble following the flow of logic, so here's it again in my own words. There is nothing new here, those are just notes. Tony Bruguier.

Contents

1 Problem setup	1
2 The iterations that compute the x_i	1
2.1 In which direction to walk?	1
2.2 The iteration	2
2.3 Some properties of the errors and residuals	2
2.3.1 Each direction only once	2
2.3.2 The orthogonality of the directions and residuals	3
3 Creating the basis of $\{d_i\}$	4
3.1 Gram-Schmidt 1	4
3.2 Orthogonality of r_i and d_j	4
3.3 Gram-Schmidt 2	4
4 The algorithm	5
4.1 Equations	5
4.2 MATLAB	5

1 Problem setup

The goal is to solve the system $Ax = b$ where A is a known (n, n) invertible matrix, b is a known $(n, 1)$ vector. In addition, it is required that A be symmetric and positive-definite.

The algorithm will compute iterative approximations of x , denoted x_i and we define the *errors* as:

$$e_i = x_i - x \tag{1}$$

We define the *residuals* as:

$$r_i = Ae_i = Ax_i - b \tag{2}$$

2 The iterations that compute the x_i

2.1 In which direction to walk?

The conjugate gradient is better than the steepest descent because it goes along each directions *only once*. To achieve this trick, we should use an orthogonal basis to choose the directions. We can use, for example, the usual basis vectors. In this case, we recover the usual elimination algorithm. Choosing a good orthogonal basis is difficult, so instead, we choose a basis that is *A-orthogonal*.

A basis $\{d_i\}$ is *A-orthogonal* iff

$$d_i^T A d_j \begin{cases} \neq 0 & \text{if } i = j \\ = 0 & \text{otherwise} \end{cases} \quad (3)$$

We will assume in section 2 that the basis $\{d_i\}$ is chosen for us. In section 3, I will show how they create this basis.

2.2 The iteration

The equation that updates the x_i is:

$$\boxed{x_{i+1} = x_i + \alpha_i d_i} \quad (4)$$

How to chose the values for α_i ? We want the errors to be A -orthogonal to the search directions. In other words:

$$d_i^T A e_{i+1} = 0 \quad (5)$$

With a regular orthogonal basis, the errors would be orthogonal to directions instead. By plugging the recurrence relation (4) into the equation (5) above, we get:

$$d_i^T A (x_{i+1} - x) = d_i^T A (x_i + \alpha_i d_i - x) = d_i^T A (e_i + \alpha_i d_i) = d_i^T r_i + \alpha_i A d_i = 0 \quad (6)$$

$$\boxed{\alpha_i = -\frac{d_i^T r_i}{d_i^T A d_i}} \quad (7)$$

This is *exactly* the same things as we would do for an orthogonal basis except that we use A in the dot products. Using an orthogonal basis, we would have certain properties. The same properties can be proven here too.

As a quick side note, using the definition of the error (1) and the recurrence equation (4), we get a recurrence equation on the errors:

$$e_{i+1} = e_i + \alpha_i d_i \quad (8)$$

We can also right multiply (4) by A and get:

$$\boxed{r_{i+1} = r_i + \alpha_i A d_i} \quad (9)$$

2.3 Some properties of the errors and residuals

2.3.1 Each direction only once

The vectors of the basis $\{d_i\}$ cannot be linearly dependent, for if they were, the basis would not be A -orthogonal (A is invertible). Thus, the initial error can be expressed as:

$$e_0 = \sum_{j=0}^{n-1} \delta_j d_j \quad (10)$$

Let's right-multiply (10) by $d_k^T A$ and use the property of A -orthogonality (3):

$$d_k^T A e_0 = \sum_{j=0}^{n-1} \delta_j d_k^T A d_j = \delta_k d_k^T A d_k \quad (11)$$

$$\delta_k = \frac{d_k^T A e_0}{d_k^T A d_k} \quad (12)$$

Let's use the recurrence equation (8) on e_i , and the A -orthogonality (3) again:

$$\delta_k = \frac{d_k^T A \left(e_k + \sum_{j=0}^{k-1} \alpha_j d_j \right)}{d_k^T A d_k} \quad (13)$$

$$\delta_k = \frac{d_k^T A e_k}{d_k^T A d_k} = \frac{d_k^T r_k}{d_k^T A d_k} \quad (14)$$

We see that this is the definition of α_i (7):

$$\delta_k = -\alpha_k \quad (15)$$

So the error is:

$$e_0 = - \sum_{j=0}^{n-1} \alpha_j d_j \quad (16)$$

and by using the recurrence equation(8) on the errors again:

$$e_i = - \sum_{j=i}^{n-1} \alpha_j d_j \quad (17)$$

This is nothing new; just generalizing well-know results from orthogonal to A -orthogonal. We see here that each direction is travelled on only *once* (very good).

2.3.2 The orthogonality of the directions and residuals

Again, an obvious property, generalized to A -orthogonal bases:

By using (17), and right-multiplying by $d_k^T A$, we get:

$$d_k^T A e_i = - \sum_{j=i}^{n-1} \alpha_j d_k^T A d_j \quad (18)$$

$$d_k^T r_i = - \sum_{j=i}^{n-1} \alpha_j d_k^T A d_j \quad (19)$$

The basis $\{d_i\}$ is A -orthogonal (3) and we for $k < i$ have:

$$d_k^T r_i = 0 \quad (20)$$

This is again nothing new, do a drawing with an orthogonal (instead of A -orthogonal basis) and you will see.

3 Creating the basis of $\{d_i\}$

3.1 Gram-Schmidt 1

Section 2 assumed the basis $\{d_i\}$ was already given. Here I show how they create this basis. The idea is to start from the r_i and use a Gram-Schmidt process:

$$d_i = r_i + \sum_{j=0}^{i-1} \beta_{i,j} d_j \quad (21)$$

The Gram-Schmidt gives us:

$$\beta_{i,j} = -\frac{r_i^T A d_j}{d_j^T A d_j} \quad (22)$$

This is quite painful because we need to keep all the vectors and do dot products. There is a way out where some magic happens. I think Gilbert Strang has a great insight [2] but I will write the full solution from [1] here.

3.2 Orthogonality of r_i and d_j

Claim: r_i is orthogonal to $\text{Span}\{d_1, d_2, \dots, d_{i-1}\}$. In other words, $(\forall j < i)(d_j^T r_i = 0)$.

Proof: true for $i = 0$ (The basis is empty)

Induction: Let's write again the recurrence equation on r_i (9): $r_{i+1} = r_i + \alpha_i A d_i$. We want to prove that $(\forall j < i + 1)(d_j^T r_{i+1} = 0)$. Let's first inspect the case where $j < i$; We have $d_j^T A d_i = 0$ precisely because we are building the basis to be A -orthogonal. By using the induction hypothesis, we also have $d_j^T r_i = 0$ and thus we have proven for $j < i$ that the induction works. What about $j = i$? Let's right multiply (9) by d_i^T . We get:

$$d_i^T r_{i+1} = d_i^T r_i + \alpha_i d_i^T A d_i = d_i^T r_i + \left(-\frac{d_i^T r_i}{d_i^T A d_i}\right) d_i^T A d_i = d_i^T r_i - d_i^T r_i = 0 \quad (23)$$

The claim is proved. This is again an extension of what is usually known.

3.3 Gram-Schmidt 2

Let's use the results of the previous section to simplify the definition of $\beta_{i,j}$ (22). The problem in this definition is that we have an A in the numerator, which we want to get rid of. So let's do a trick and use the recurrence equation (9):

$$r_i^T r_{j+1} = r_i^T (r_j + \alpha_j r_i^T A d_j) = r_i^T r_j + \alpha_j r_i^T A d_j \quad (24)$$

And so:

$$r_i^T A d_j = \frac{r_i^T r_{j+1} - r_i^T r_j}{\alpha_j} \quad (25)$$

$$\beta_{i,j} = -\frac{1}{\alpha_j} \cdot \frac{r_i^T r_{j+1} - r_i^T r_j}{d_j^T A d_j} \quad (26)$$

For $j < i - 1$, both $r_i^T r_{j+1}$ and $r_i^T r_j$ are zero by section 3.2 and $\beta_{i,j} = 0$. This is great, it kills many terms.

For $j = i - 1$, we have:

$$\beta_{i,i-1} = -\frac{1}{\alpha_{i-1}} \cdot \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}} \quad (27)$$

$$\beta_{i,i-1} = \frac{d_{i-1}^T A d_{i-1}}{d_{i-1}^T r_{i-1}} \cdot \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}} \quad (28)$$

$$\beta_{i,i-1} = \frac{r_i^T r_i}{d_{i-1}^T r_{i-1}} \quad (29)$$

The proof of the section above greatly simplifies and speeds up the Gram-Schmidt equation.

We only have one vector to remove.

$$\boxed{d_i = r_i + \beta_i d_j} \quad (30)$$

$$\boxed{\beta_i = \frac{r_i^T r_i}{d_{i-1}^T r_{i-1}}} \quad (31)$$

4 The algorithm

4.1 Equations

We are almost there. There is one little trick left. It is not necessary, but it makes the code beautiful.

Given the updating algorithm of to find the directions d_i is (30), and so we see that $d_k^T r_k = d_k^T d_k$. With this, let us rewrite the boxed equations (4), (7), (9), (30), and (31):

$$\alpha_i = -\frac{r_i^T r_i}{d_i^T A d_i} \quad (32)$$

$$x_{i+1} = x_i + \alpha_i d_i \quad (33)$$

$$r_{i+1} = r_i + \alpha_i A d_i \quad (34)$$

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}} \quad (35)$$

$$d_i = r_i + \beta_i d_j \quad (36)$$

4.2 MATLAB

```
function cg
```

```
fclose all;
```

```
close all;
```

```
clear all;
```

```
A = reshape(mod((1:9)*11 - 3.5,7), 3, 3);
```

```
A = A + A';
```

```
b = [-1; 2; 4];
```

```
x = zeros(3, 1);
```

```

r = A*x - b;
d = A*x - b;

for ii = 1:3
    alpha = -(r' * r) / (d' * A * d);

    x = x + alpha * d;

    rn = r + alpha * A * d; % or A * x - b

    beta = (rn' * rn) / (r' * r);

    d = rn + beta * d;

    r = rn;
end

x

A \ b

```

References

- [1] <http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf> - *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain* - Jonathan Richard Shewchuk
- [2] <http://academicearth.org/lectures/conjugate-gradient-method> - *Mathematical Methods for Engineers II, Conjugate Gradient Methods, MIT OCW 18.086* - Gilbert Strang