Numerical Optimisation: Assignment 4

Student Number:13064947

February 2018

1 Exercise 1

Question: Consider the linear system $Ax = b$ with $A \in R^n, b \in R^n$. Starting at $x_0 = (0, \ldots, 0)^T$, a tolerance $tol = 1e-12$ and a dimension $n = 100$. Implement the pre-conditioned conjugate gradient method. (Here we are given an xtrue).

1.1 Part a)

Implemented in Cody coursework

1.2 Part b)

Solve the linear system for the following A matrices.

- $A1 = diag(1 : n)$
- $A2 = diag([ones(n-1, 1); 100])$
- 1D negative laplacian (100 dimensions)

Given we are given xtrue and our implementation was already tested on other tests on cody coursework, I am presuming by 'solve the question' the question is really about analysing the convergence properties of the conjugate gradient method for the linear systems given above. We already know that given n distinct eigenvalues we will have convergence in at most n iterations. However it turns out that the distribution of eigenvalues is also important.

The key intuition can be found in Theorem 5.5 in Nocedal.

If A has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, then we have that:

$$||x_{k+1} - x^*||_A^2 \leq \left(\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1}\right)^2 ||x_0 - x^*||_A^2 \tag{1}$$

Where x_0 is our initial guess and x^* the solution.
Basically what the equation is saying that after an initial guess the 'distance'

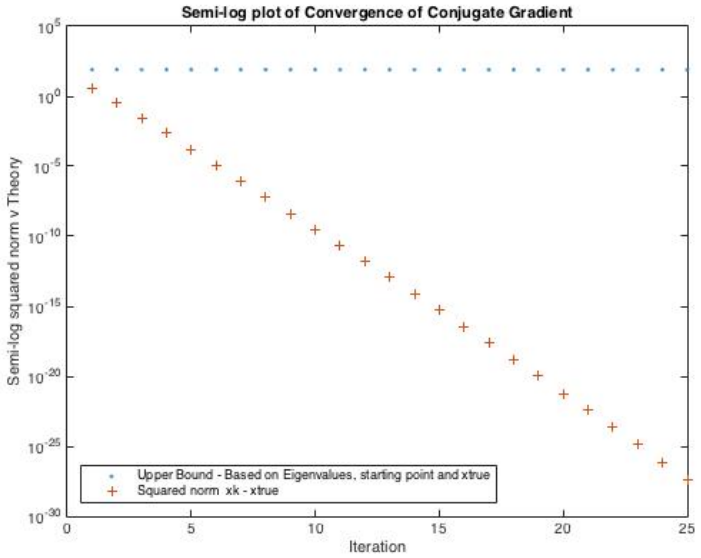


Figure 1: Matrix A1

between our value at some iteration k+1 and the true value of the optimisation is bounded by a constant multiplied by a function of our eigenvalues. The key aspect of this function is if one orders the eigenvalues in descending order, then the speed of convergence is effectively dependent upon how close the $n-k$th eigenvalue is to the smallest eigenvalue. So we need only look at the distribution of eigenvalues of A to have all the information we need about the speed of convergence of the conjugate gradient method (linear).

1.2.1 $A1 = diag(1:n)$

Clearly in 100 dimensions we simply have $\lambda_{100} = 100, \lambda_{99} = 99, \ldots, \lambda_1 = 1$. In terms of our upper bound this means that $\left(\frac{\lambda_{n-k}-\lambda_1}{\lambda_{n-k}+\lambda_1}\right)^2 \approx 0.5$ even after many eigenvalues and thus many iterations. A semi-log graph of the squared norm of the error by iterations, against the theoretical upper bound is provided in fig 1.

Note in practise that the CG algorithm actually converged in far fewer iterations than the upper bound might imply. For this particular matrix the upper bound holds but appears at first sight to be pessimistic.

1.2.2 $A2 = diag([ones(n-1,1);100])$

In this case we have $\lambda_{100} = 100, \lambda_{99} = 1, \ldots, \lambda_1 = 1$. This is exceptional and really there is nothing to show as after the first iteration the denominator on

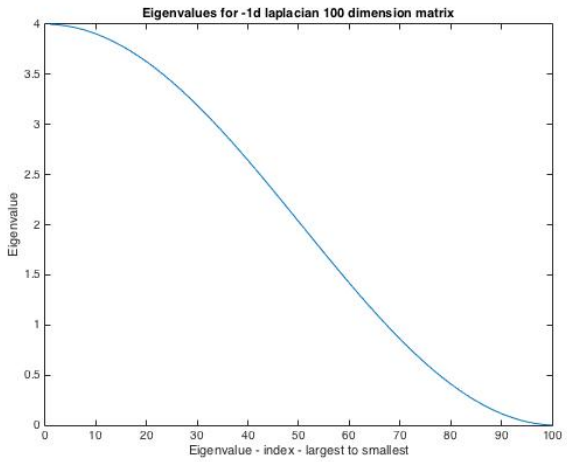


Figure 2: -1D Laplacian eigenvalues

the upper bound will be zero.

When tested in practise the CG algorithm also converged straight away (thus supporting the theory), but also leaving one bereft of anything to show graphically!

1.2.3 1D negative laplacian (100 dimensions)

In the case of the 1D negative laplacian we have somewhat worse convergence results. To simplify the discussion fig 2, shows the eigenvalues in descending order. What is clear is that the smallest eigenvalue is close to zero, and the eigenvalues reduce slowly. This is thus worse than for A1 (where the ratio was close to 0.5). In this case $\left(\frac{\lambda_{n-k}-\lambda_1}{\lambda_{n-k}+\lambda_1}\right)^2 \approx 1.0$ for a number of iterations.

Again the semi-log plot of the squared error of the norm by iteration is given against this theoretical upper limit and here we see the pessimism is justified as we continue to 100 iterations.

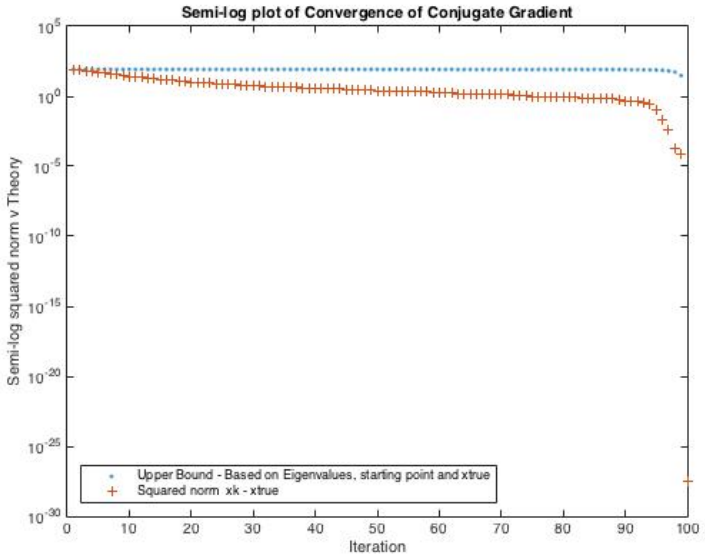


Figure 3: Matrix A3

2 Exercise 2

2.1 Parts a and b)

The implementations of Fletcher-Reeves and Polyak-Ribiere were again submitted to cody coursework.

2.2 Part c)

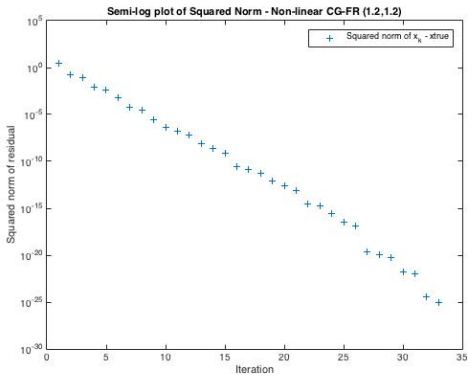
Minimise the function $f(x, y) = x^2 + 5x + 10y^2$ from the initial points $x_0 = (1.2, 1.2)^T$ and $x_0 = (-1, -1)^T$, tolerance as above for both non-linear conjugate gradient methods.

2.2.1 Start point (1.2,1.2)

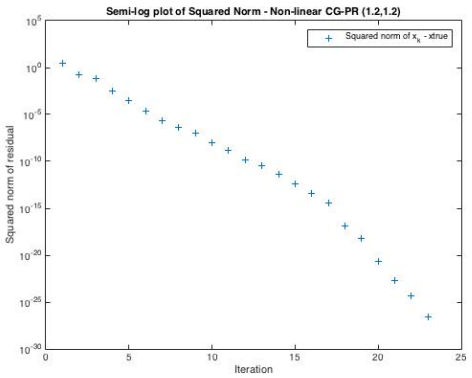
Fig. 4, shows a similar convergence analysis of Fletcher-Reeves versus Polyak-Ribiere as given above. That is illustrate the squared error by iteration (semi-log y-axis). In the first test we see that Polyak-Ribiere converges in fewer iterations.

2.2.2 Start point (-1,-1)

Fig. 5, shows the convergence properties using a different starting point. In the second case the differences in convergence are very small, indeed Fletcher-Reeves starts out very slightly better. This is a short empirical demonstration



(a) Fletcher Reeves (1.2,1.2)



(b) Polyak-Ribiere (1.2,1.2)

Figure 4: FR v PR Conjugate Gradient descent

but doesn't really explain some deeper properties of the respective algorithms. A few brief intuitions will be given in the next section.

2.2.3 Brief summary of Convergence features of FR v PR

The line search method is important. If the line search is exact then we have a minimiser of our function. In the case of Fletcher-Reeves is turns out that as long as the line search meets the strong Wolfe conditions then we get convergence guarantees.(Equations (5.43a,b) Nocedal).

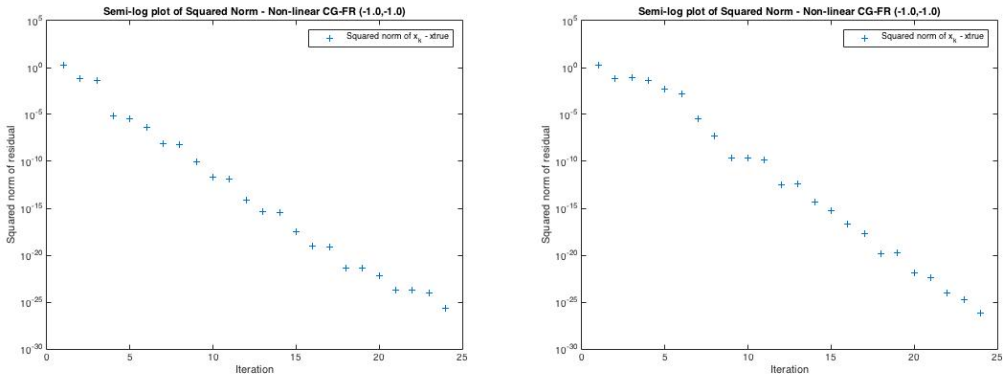
The difference with Polyak-Ribiere is that the strong Wolfe conditions **do not** guarantee that we have a descent direction and thus without adjustment we can end up in an infinite cycle without reaching the solution.

This can be found in Nocedal with various convergence proofs and is considered surprising because despite the lack of guarantees Polyak-Ribiere is deemed to typically converge quicker and be somewhat more robust (at least according to the book and our tiny empirical example).

Ensuring convergence :- So we know that Fletcher-Reeves converges under the strong Wolfe conditions, it also turns out a slight adjustment to Polyak-Ribiere can be made to ensure convergence (under adjusted strong Wolfe conditions). The adjustment is to change β^{PR}.

In Polyak-Ribiere:

$$\beta_{k+1}^{PR} = \frac{\nabla f_{k+1}^T(\nabla f_{k+1} - \nabla f_k)}{||\nabla f_k||^2} \tag{2}$$



(a) Fletcher Reeves (-1,-1)

(b) Polyak-Ribiere (-1,-1)

Figure 5: FR v PR Conjugate Gradient descent

If one instead uses $\beta_{k+1} = \max\left(\beta_{k+1}^{PR}, 0\right)$ and adjusts the strong Wolfe conditions then we always maintain a descent direction in Polyak-Ribiere and thus converge.

Much of this is clearly from looking at Nocedal. What I found unsatisfying about this explanation is this is all about asymptotic converge properties without understanding yet what can happen in practice. Here what is interesting is the β, which plays a key part. The short story is that overall - resetting β to zero periodically, guarantees convergence (in other proofs in Nocedal) and of course works for the adjustment to Polyak-Ribiere.

What I found more interesting was the different behaviour described for Fletcher-Reeves and Polyak Ribiere in practice when we have a poor search direction. The key equation is:

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{||\nabla f_k||||p_k||} \tag{3}$$

where p_k is our line search direction and $-\nabla f_k^T$ of course our direction of steepest descent. With θ_k being the angle between the two. Again the material is in Nocedal - in short if p_k is close to orthogonal to the gradient, then not only will the step size be very small but because of the small steps the next gradient will be very similar and $\beta \approx 1$ so the next line search direction will be similar to the last...i.e. poor.

This is in sharp contrast to Polyak-Ribiere, where under similar circumstances with $\nabla f_k \approx \nabla f_{k+1}$ instead gives a $\beta \approx 0$ and so the next search direction will be similar to steepest descent. Effectively equivalent to a restart and

perhaps giving intuition as to why it often converges quicker in practice and as to why restarts can aid convergence.

The function optimised in this case was rather simple, so although Polyak-Ribiere did converge quicker in the first case. There are no doubt far more extreme functions where the different behaviours described above can be shown more starkly.