Solution of a NxN System of Linear Algebraic Equations: A New Algorithm with a Low Time Complexity

Dr. V. S. Patwardhan[1]

November 2022

Abstract

A new algorithm is presented for solving a NxN system of linear algebraic equations. The main idea behind this algorithm is to generate a second system of linear algebraic equations (having the same solution) and take steepest descent steps alternately with the two systems of equations. This algorithm was tested with randomly generated problems for N up to 1000 and was found to approach the solution closely in $O(N^{2.2})$ time, which is better than earlier algorithms. It was also tested with sample problems selected from the SuiteSparse collection of matrices (which are widely used as benchmark matrices for testing sparse matrix algorithms) which have been obtained from practical applications in several different areas such as chemical process simulation, computational fluid dynamics and many others. The current algorithm successfully solves most of these problems as well, without the use of preconditioners, provided the condition number of the underlying matrix is not too large. For matrices with a very large condition number, preconditioners may have to be used, just as other algorithms do. A notable feature of the current algorithm is that it is based on taking steepest descent steps alone.

Introduction

Solving a system of linear algebraic equations is a classical problem which has many practical applications in areas such as engineering and science. Systems of large/huge size, involving millions of equations and unknowns, arise in many diverse frontier areas including process simulation, computational fluid dynamics, meshing, machine learning, computational chemistry, data mining, bioinformatics etc. Efficient methods for solving such systems are therefore becoming more and more important. Such methods ideally should converge fast in a reasonably small number of iterations, make use of the sparsity to the full extent, be able to deal with very ill-conditioned systems using appropriate preconditioners, have a low time complexity, and be suitable for parallelisation. Here we present a new algorithm which has a low time complexity and uses only matrix vector multiplications as the main computational effort. Thus, it can make full use of sparsity, and can be easily parallelized. It is shown to handle quite ill-conditioned problems without any preconditioner.

The NxN system of linear equations can be written in a matrix form as $\mathbf{A_0 x} = \mathbf{b_0}$, where $\mathbf{A_0}$ is a NxN matrix, $\mathbf{b_0}$ is an N-vector, and $\mathbf{x}$ is the solution vector to be determined. Each row of $\mathbf{A_0}$ is assumed to

---

[1] Independent researcher. Formerly, Scientist G, National Chemical Laboratory, Pune 411008, India.
Email: vspatw@gmail.com , URL : https://www.vspatwardhan.com

be normalized. There are many direct methods such as gaussian elimination and others which give an exact solution. However, it is often sufficient to get an approximate solution in practical applications. Iterative methods essentially start with a guess solution and improve it iteratively to get closer to the solution within acceptable accuracy. A quick summary of these direct as well as iterative methods can be found in standard books on linear algebra [for example, J. E. Gentle, 2007]. Iterative methods such as the steepest descent method and the conjugate gradient method are used when the matrix is symmetric and positive definite. (It is well known that $\mathbf{A_0 x} = \mathbf{b_0}$ can be put in the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} = \mathbf{A_0}^T\mathbf{A_0}$ and $\mathbf{b} = \mathbf{A_0}^T\mathbf{b_0}$. This gives a symmetric positive definite matrix $\mathbf{A}$ for any $\mathbf{A_0}$.) These are based on minimizing an appropriately defined quadratic function, using optimization techniques. Details of these methods, including convergence analysis, are available in standard books and reports [for example, J. R. Shewchuk, 1994]. Variations of the steepest descent method are available which use the momentum concept to achieve faster convergence [Y. Nesterov, 1983; I. Sutskever et. Al., 2013].

The direct methods such as Gaussian elimination and derived methods can be shown to run in $O(N^3)$ time. This computational complexity is closely related to the complexity of matrix multiplication. A straightforward multiplication of two matrices also has a complexity of $O(N^3)$. There have been steady efforts in reducing this complexity. It was shown by Strassen [1969] that the complexity can be reduced to $O(N^{2.8})$ by rearranging the computations. It was reduced further to $O(N^{2.37})$ by Coppersmith and Winograd [1990]. Recently the complexity has been reduced further to $O(N^{2.332})$ by Peng and Vempala [2021]. The question whether it can be reduced to the theoretical minimum of $O(N^2)$ is still an open question.

The algorithm developed here, which can be termed as the augmented matrix (AM) algorithm, is inspired by some geometrical observations, described in detail below. It is tested using (1) randomly generated problems which involve gaussian matrices of different sizes where each element is N(0,1), and (2) sample problems selected from the SuiteSparse collection of matrices, which are widely used as benchmark matrices. It is an iterative algorithm, which involves augmenting the coefficient matrix with an additional row (which is equivalent to adding a new, consistent equation to the system of equations) and taking steepest descent steps alternately with the original and augmented matrices.

Augmented matrix algorithm (AM algorithm)

There are three geometrical observations, made earlier, which together lead to the AM algorithm, and are described below:

(1) The first observation concerns the application of the steepest descent (SD) method for solving a NxN system of linear equations. The SD method starts with a given point and obtains subsequent points iteratively, monitoring the approach to solution in terms of the sum of squares of the residuals at each step. It is well known that the steepest descent method leads to a fast approach to the solution (i.e., gives rapid reduction in residuals) in the first few steps and slows down substantially in the following steps. There are two possible ways of bypassing this slow down [Patwardhan, 2022a] and making the steepest descent method much faster, which include random movement of the point between iterations, and possible matrix transformations between iterations. It was shown that these approaches can increase the speed of convergence of the steepest descent method by

several orders of magnitude, though the question of how to achieve random movement / matrix transformations was not addressed.

(2) The second observation concerns the geometry of the intersecting hyperplanes representing a NxN system of linear equations. It has been shown earlier [Patwardhan, 2022b] that, in a large dimensional space defined by a NxN system of linear equations with large N, several directions exist which are almost orthogonal to all the rows of the matrix. Using one or more of these directions to get a new equation (i.e., an augmented matrix), it is possible to change the orientation of the ellipsoids of the sum of squares of the residuals significantly. This makes it possible to use the SD method alternately with the original and the augmented matrices, to achieve fast convergence to the solution. It may be noted that changing the orientation of the ellipsoids is, in principle, equivalent to the random movement of the point, or possible matrix transformations, between iterations.

(3) The third observation concerns the effect of adding a new, consistent equation to the system of linear algebraic equations, i.e. $\mathbf{A_0 x} = \mathbf{b_0}$ repeatedly, on eigenvalues and eigenvectors of **A**. Let the new equation be $\mathbf{v}^T\mathbf{x} = w$. Let us assume that **v** is normalized, and that the solution **s** satisfies this equation, i.e., $\mathbf{v}^T\mathbf{s} = w$. (**v** can be one of the directions mentioned in the second observation. The choice of w is described later.) It has been shown earlier [Patwardhan, 2022b] that if this equation is added k times to the $\mathbf{A_0 x} = \mathbf{b_0}$ system, we get an augmented system with a (N+k)x(N) coefficient matrix and a (N+k)x(1) right hand side. This system can be converted to a symmetric positive definite system $\mathbf{A_k x} = \mathbf{b_k}$ where

$$\mathbf{A_k} = (\mathbf{A_0}^T\mathbf{A_0} + k\mathbf{v}\mathbf{v}^T) \text{ and } \mathbf{b_k} = \mathbf{A_0}^T\mathbf{b_0} + kw\mathbf{v} \qquad (1)$$

It has been shown earlier that for a large enough value of k, (i) **v** becomes the eigenvector of $\mathbf{A_k}$ corresponding to the largest eigenvalue (which is equal to k itself), and (ii) both the systems, i.e., $\mathbf{A_0 x} = \mathbf{b_0}$ and $\mathbf{A_k x} = \mathbf{b_k}$, have the same solution **s**., provided $\mathbf{v}^T\mathbf{s} = w$. From a geometrical viewpoint, the SS_{res} contours change orientation as k increases, while the solution **s** remains unchanged. A value of k = 4 or more was found to be "sufficiently large" for N up to 1000.

A key point is the appropriate choice of w. For a given vector **v**, the ideal choice of w is $\mathbf{v}^T\mathbf{s}$. However, since **s** is not known at the beginning, the algorithm starts with a trial value of w, which gets updated at each iteration.

A brief outline of the AM algorithm

We start with two points $\mathbf{q}_1$ and $\mathbf{q}_2$ which define a line that points approximately towards the solution (the computation of $\mathbf{q}_1$ and $\mathbf{q}_2$ is described below.) The direction **v** given by $\mathbf{q}_1$ and $\mathbf{q}_2$ is used to get a trial solution **s**, and a new equation passing through the trial solution, using $w = \mathbf{v}^T\mathbf{s}$. The original system of equations, i.e. $\mathbf{A_0 x} = \mathbf{b_0}$, is converted into two (NxN) symmetric positive definite systems, i.e. $\mathbf{A x} = \mathbf{b}$ and $\mathbf{A_k x} = \mathbf{b_k}$ using equation (1). The point $\mathbf{q}_1$ is adjusted by taking a fixed number of SD steps with the two systems of equations alternately. The point $\mathbf{q}_2$ is also adjusted similarly. The adjusted points are used to get an improved direction **v**, a new trial solution, and a new value of w. This is done iteratively till the SS_{res} at the approximate solution point becomes acceptably low, or the iteration count exceeds a set maximum value.

The augmented matrix algorithm in detail

Given: N, $\mathbf{A}_0$, and $\mathbf{b}_0$

Selectable parameters: k, n_1, n_2, m_1, m_2

Initial calculations

1 Calculate $\mathbf{A}$ (= $\mathbf{A}_0^T\mathbf{A}_0$) and $\mathbf{b}$ (= $\mathbf{A}_0^T\mathbf{b}_0$)
2 Get two random vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ with elements N(0,1)
3 Normalize $\mathbf{z}_1$ and $\mathbf{z}_2$
4 Set $\mathbf{p}_1 = \mathbf{z}_1$
5 Get $\mathbf{q}_{1,0}$ by applying m_1 steepest descent steps to $\mathbf{p}_1$, using $\mathbf{A}$ and $\mathbf{b}$
6 Calculate d = distance $\mathbf{p}_1$ - $\mathbf{q}_{1,0}$
7 Set $\mathbf{p}_2 = \mathbf{q}_{1,0} + m_2\, d\, \mathbf{z}_2$
8 Get $\mathbf{q}_{2,0}$ by applying m_1 steepest descent steps to $\mathbf{p}_2$, using $\mathbf{A}$ and $\mathbf{b}$
9 Get $\mathbf{v}_0 = \mathbf{q}_{1,0} - \mathbf{q}_{2,0}$
10 Get $\mathbf{s}_0$, the point which minimizes SS_{res} along the line $\mathbf{v}_0$ drawn through $\mathbf{q}_{2,0}$

Iterative calculations

11 Set i = 1
12 While $\sigma_{res} > \sigma_{res,crit}$, do
13 Set $w_{1,i} = \mathbf{v}_{i-1}^T\mathbf{s}_{i-1}$
14 Set $w_{2,i} = \mathbf{v}_{i-1}^T\mathbf{q}_{2,i-1}$
15 Calculate $\mathbf{A}_{k,i} = (\mathbf{A_0}^T\mathbf{A_0} + k\mathbf{v}\mathbf{v}^T)$
16 Calculate $\mathbf{b}_{k,i} = \mathbf{A_0}^T\mathbf{b_0} + kw_{1,i}\mathbf{v}$
17 For j = 1 to n_1
18 Adjust $q_{1,i-1}$ by applying n_2 SD steps with $\mathbf{A}_{k,i}$ and $\mathbf{b}_{k,I}$
19 Adjust $q_{1,i-1}$ further by applying n_2 SD steps with $\mathbf{A}$ and $\mathbf{b}$
20 Next j
21 Set $q_{1,i} = q_{1,i-1}$
22 Calculate $\mathbf{b}_{k,i} = \mathbf{A_0}^T\mathbf{b_0} + kw_{2,i}\mathbf{v}$
23 For j = 1 to n_1
24 Adjust $q_{2,i-1}$ by applying n_2 SD steps with $\mathbf{A}_{k,i}$ and $\mathbf{b}_{k,I}$
25 Adjust $q_{2,i-1}$ further by applying n_2 SD steps with $\mathbf{A}$ and $\mathbf{b}$
26 Next j
27 Set $q_{2,i} = q_{2,i-1}$
28 Calculate $\mathbf{v}_i = \mathbf{q}_{2,i} - \mathbf{q}_{1,I}$
29 Get $\mathbf{s}_i$, the point which minimizes SS_{res} along the line $\mathbf{v}_i$ drawn through $\mathbf{q}_{2,i}$
30 i = i + 1
31 End while

At the end of initial calculations (step 10) the AM algorithm comes up with $\mathbf{q}_{1,0}$, $\mathbf{q}_{2,0}$, $\mathbf{v}_0$ and $\mathbf{s}_0$. These are then improved iteratively through steps 11-31. Steps 13 and 14 are aimed at keeping $\mathbf{q}_{2,i}$ as an anchor, away from the solution, while pushing $\mathbf{q}_{1,i}$ towards the solution, which makes the direction $\mathbf{v}_i$ more accurate as iterations proceed.

The AM algorithm was tested with two sets of problems. The first set consisted of randomly generated problems with a problem size up to N = 1000, and the second set of problems was a subset selected from the SuiteSparse collection. The performance of the algorithm is characterised in terms of the variation of σ_{res} and $\sigma_{\Delta x}$ with successive iterations, the number of iterations required to reduce σ_{res} below a critical value, and the running time. The results obtained with these two sets of problems are described below.

Results using randomly generated test problems

Problem generation:

It is expected that any iterative method for solving a system of algebraic equations would give a quicker approach to the solution if the starting point were closer to the solution. To remove this source of variation, the problems generated had their solution at a fixed distance from the origin, but in a random direction. The following procedure was used for generating the problem data, i.e. $\mathbf{A}_0$ and $\mathbf{b}_0$. A random vector was generated with N(0,1) elements (i.e. standard normal deviates). The solution point **s** was chosen 10 units away from the origin in the direction of this random vector. The coefficient matrix $\mathbf{A}_0$ was generated with N(0,1) elements, and each row of $\mathbf{A}_0$ was normalized. The $\mathbf{b}_0$ vector was generated as $\mathbf{b}_0 = \mathbf{A}_0\mathbf{s}$. (The solution **s** was used only for generating the right-hand side and was completely ignored while solving the system of equations using the AM algorithm.) N was varied from 10 to 1000 using 13 values of N which were uniformly spaced on a logarithmic scale. The actual values of N are seen in Table 1. For each value of N, five independent problems were generated and solved.

Computational results:

The parameter values selected were k = 30, m_1 = 20, m_2 = 10, n_1 = 40 and n_2 = 20. The AM algorithm was applied to the system $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$. A point 10 units away from the origin, in a random direction from the origin, was taken as the starting point. The progress of iterations was monitored by calculating σ_{Res} (i.e., the root mean square of the residuals, $\mathbf{R}_i$ for the current **x**), and iterations were terminated when σ_{Res} became smaller than a critical value (which was selected as 0.0001). The final point was characterised by calculating $\sigma_{\Delta x}$ (i.e., the root mean square value of the deviations ($\mathbf{x}_i$ - $\mathbf{s}_i$), for i=1 to N). During the iterative calculations, the known solution point was used only for characterising the successive iterates as stated above. The algorithm was tested for thirteen values of N from 10 to 1000. These values of N were such that they were almost uniformly spaced on a log scale. For each N, five different problems were generated at random. The running time refers only to the time taken for calculating the solution, and excludes the time taken for problem generation or reading. The running times were normalised by using the running time for N = 100, and only the ratios are considered. This was done to filter out the effect of variations in operating speeds of different computer systems.

The computational results obtained with these randomly generated problems are shown in Table 1 below. Figure 1 shows a log-log plot of run time ratio vs. problem size N. Figure 2 shows a similar plot of the number of iterations required vs. problem size N.

No.	N	σ_{Res}	$\sigma_{\Delta x}$	Run time ratio	iterations required
1	10	1.16E-06	4.12E-06	1.58E-02	1.8
2	14	5.54E-06	2.05E-04	1.94E-02	1.8
3	20	2.25E-05	1.52E-04	3.41E-02	1.8
4	32	1.76E-05	3.20E-04	1.10E-01	3
5	50	3.07E-05	7.02E-04	2.05E-01	2.6
6	70	2.56E-05	8.66E-02	4.66E-01	2.8
7	100	4.86E-05	7.95E-03	1.00E+00	2.8
8	140	5.48E-05	7.71E-03	3.23E+00	4.2
9	200	5.62E-05	2.60E-02	8.65E+00	5.2
10	320	6.50E-05	2.34E-02	1.79E+01	4.4
11	500	7.45E-05	1.07E-02	5.54E+01	5.6
12	700	7.00E-05	1.71E-02	1.46E+02	7.4
13	1000	9.29E-05	1.92E-02	1.94E+02	5.8

Table 1. Computational results obtained for randomly generated problems of size of N = 10-1000 (average of 5 different problems for each N)

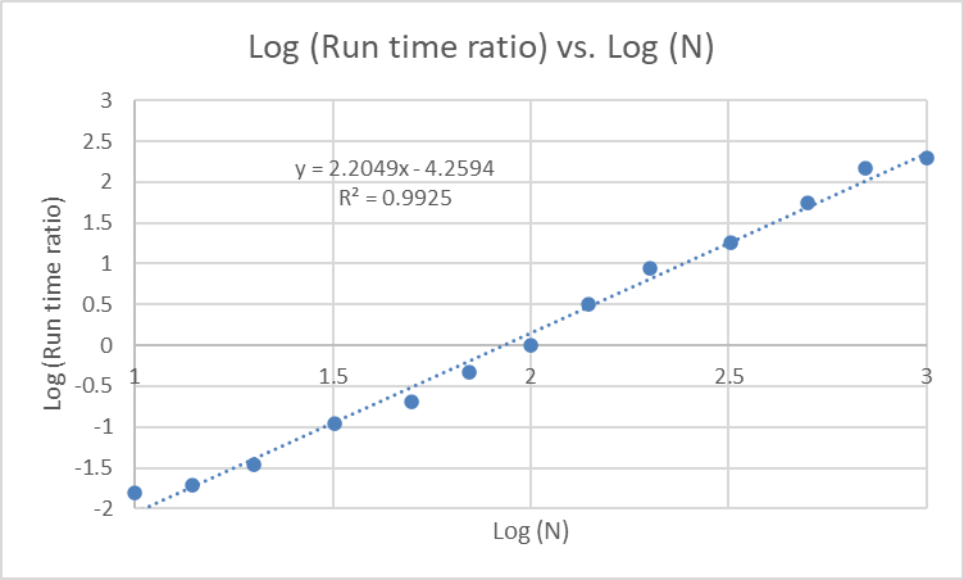

Figure 1: Reduced run time for different N values

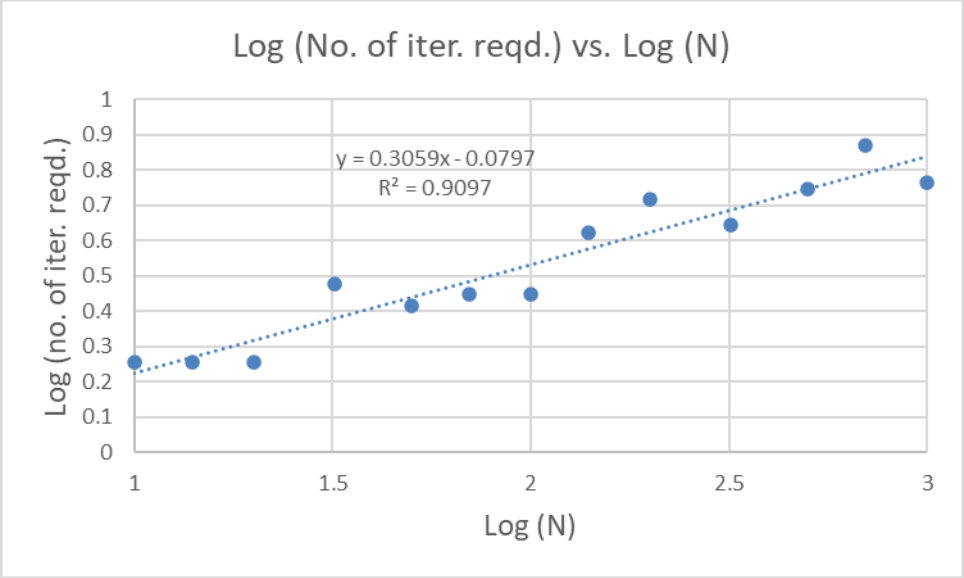

Figure 2: No. of iterations required for different N values

It is seen from Table 1 that all the problems were solved successfully, with the final σ_{Res} less than 0.0001. The $\sigma_{\Delta x}$ values increase somewhat with N, but even for N = 1000, all coordinates get calculated within about 0.02 or so. Figure 1 shows that the time complexity of the AM algorithm is about $O(N^{2.2})$. This is better than computational complexities reported so far. Figure 2 shows that the number of iterations increase very slowly with N, and show a dependence of $O(N^{0.3})$ or so. Table 1 shows that as N goes from 10 to 1000, the number if iterations increase only from 1.8 to 5.8. In other words, as N increases 100-fold, the number of iterations increases only by a factor of 3.2 or so. The main calculations within a single iteration involve a fixed number of only matrix-vector multiplications, which are $O(N^2)$. So, the overall increase in computation time with N, beyond the factor of N^2, is only due to the increase in number of iterations. However, this dependence is only a weak dependence on N.

Results using problems from the SuiteSparse collection

The test matrices used above were randomly generated dense gaussian matrices. It is well known that results obtained with artificially generated matrices can be misleading to some extent. Huge matrices that arise in practical applications in areas such as optimization, chemical process simulation, structural engineering, computational fluid dynamics etc. are highly sparse, and often have a very high condition number, which can make computations very time consuming.
The SuiteSparse Matrix Collection (formerly known as the University of Florida Sparse Matrix Collection), is a large and actively growing set of sparse matrices that arise in real applications [Davies A. and Hu, Y., 2011]. The Collection is widely used for the development and performance evaluation of sparse matrix algorithms. It allows for robust and repeatable experiments, since the matrices are from real life applications, and are publicly available in many formats.

No.	Problem Name	N	Non-zero elements	Non-zero elements per row	Condition number
1	pores_1.mtx	30	180	6	1.81E+06
2	d_ss.mtx	53	149	2.8	6.14E+08
3	impcol_b.mtx	59	312	5.2	1.64E+05
4	west0067.mtx	67	294	4.3	1.30E+02
5	steam3.mtx	80	928	11.6	4.99E+10
6	d_dyn.mtx	87	238	2.7	7.42E+06
7	d_dyn1.mtx	87	238	2.7	7.43E+06
8	tols90.mtx	90	1746	19.4	2.02E+04
9	olm100.mtx	100	396	3.9	1.53E+04
10	tub100.mtx	100	396	3.9	1.33E+04
11	lns_131.mtx	131	536	4	1.28E+15
12	lnsp_131.mtx	131	536	4	1.28E+15
13	west0132.mtx	132	414	3.1	4.21E+11
14	impcol_c.mtx	137	411	3	1.77E+04
15	west0156.mtx	156	371	2.3	2.31E+18
16	west0167.mtx	167	507	3	4.79E+10
17	bwm200.mtx	200	796	3.9	2.41E+03
18	rdb200.mtx	200	1120	5.6	3.45E+02
19	rdb200l.mtx	200	1120	5.6	1.33E+02
20	impcol_a.mtx	207	572	2.7	1.35E+08
21	ex1.mtx	216	4352	20.1	3.30E+04
22	impcol_e.mtx	225	1308	5.8	7.10E+06
23	saylr1.mtx	238	1128	4.7	7.78E+08
24	steam1.mtx	240	3762	15.6	2.83E+07
25	tols340.mtx	340	2196	6.4	2.03E+05
26	poisson2D.mtx	367	2417	6.5	1.33E+02
27	impcol_d.mtx	425	1339	3.1	2.06E+03
28	ex2.mtx	441	13640	30.9	1.03E+10
29	rdb450.mtx	450	2580	5.7	6.85E+02
30	rdb450l.mtx	450	2580	5.7	2.10E+02
31	olm500.mtx	500	1996	3.9	3.73E+05
32	pores_3.mtx	532	3474	6.5	5.61E+05
33	steam2.mtx	600	13760	22.9	3.78E+06
34	ex21.mtx	656	19144	29.1	5.68E+08
35	rdb800l.mtx	800	4640	5.8	3.23E+02
36	ex22.mtx	839	22715	27	3.28E+04
37	ex25.mtx	848	24612	29	5.11E+07
38	orsirr_2.mtx	886	5970	6.7	6.33E+04
39	DK01R.mtx	903	11766	13	5.89E+07
40	rdb968.mtx	968	5632	5.8	3.78E+02

Table 2. Problems selected from the SuiteSparse collection

The AM algorithm described above, was used for solving a sample of test matrices from the SuiteSparse collection. Computational results obtained are described below.

Problem selection: Several problems were selected from the SuiteSparse collection, from the areas of chemical process simulation and computational fluid dynamics, with N <= 1000, to keep the

computation time reasonable. The condition numbers of the selected matrices, as reported in the SuiteSparse collection, covered a wide range of 130 to 2.31×10^{18}. The problems selected are summarised in Table 2.

The problems listed in Table 2 cover a range of N up to 1000, and have a very high sparsity, as indicated by the number of non-zero elements per row. The condition numbers cover a very wide range of 130 to 2.31×10^{18}. Most of the matrices did not have a right-hand side. To test the AM algorithm (which aims to solve linear equations), right-hand sides had to be generated. For this purpose, a point was generated in a random direction, 100 units away from the origin, and was taken as the solution **s**. The right-hand side was then generated using this solution, as $\mathbf{b}_0 = \mathbf{A}_0\mathbf{s}$. (The solution **s** was used only for generating the right-hand side and was completely ignored while solving the system of equations using the AM algorithm.) A point 10 units away from the origin, in a random direction, was taken as the starting point for the AM algorithm.

Computational results:

The parameter values selected were k = 30, $m_1 = 20$, $m_2 = 10$, $n_1 = 40$ and $n_2 = 20$. Table 3 shows the computational results obtained with the AM algorithm for the problems listed in Table 2. It is seen that 27 of the 40 problems selected are solved very well by the AM algorithm. For these problems, the AM algorithm rapidly approaches the solution, often in just a few iterations. For these problems, the average deviation in solution coordinates is less than 0.0005, the average number of iterations required is 11, and the condition numbers cover a range from 130 to 4.99×10^{10}. For the 13 problems which are not solved within 50 iterations, the speed of approaching the solution becomes too low. This needs further investigation, and probably, some modification of the AM algorithm. The condition numbers cover a range from 6.33×10^4 2.31×10^{18}. This range is much higher than that covered by problems which were successfully solved by the AM algorithm. The condition number values appear to correlate with the success/failure of the AM algorithm in the current form. It may be noted that the AM algorithm does not make use of any preconditioner matrices. If preconditioners are used, all these problems are likely to get solved successfully. This aspect needs to be investigated further.

Discussion

The results presented in Table 1 and Figures 1 and 2 indicate that the AM algorithm successfully solved all the 65 randomly generated problems with $N \leq 1000$. The final solution given by the AM algorithm was very close to the actual solution. The approach to the solution was fast, and the algorithm took just a few iterations. The number of iterations was found to have only a weak dependence on the problem size. The overall time complexity was found to be in $O(N^{2.2})$, which is very attractive.

The AM algorithm uses k, n_1, n_2, m_1 and m_2 as user-selected parameters. It has been mentioned earlier that k = 4-5 is sufficient for a shift of the eigenvector for the largest eigenvalue to **v**. However, a higher value of 30 was found to be more suitable for computations. The parameters n_1 and n_2 (= 40 and 20 respectively) define the number of SD steps taken by the algorithm. If these can be reduced, the algorithm would run faster. Work in this direction is in progress. The parameters m_1 and m_2 (= 20 and 10 respectively) which appear in the initial calculations of the AM algorithm are relatively less critical and can be changed if desired.

No.	Problem Name	N	Condition number	Iterations reqd.	$\sigma_{\Delta x}$	Solved ?
1	pores_1.mtx	30	1.81E+06	30	0.0004267	Yes
2	d_ss.mtx	53	6.14E+08	8	7.332E-05	Yes
3	impcol_b.mtx	59	1.64E+05	8	3.275E-05	Yes
4	west0067.mtx	67	1.30E+02	4	1.568E-06	Yes
5	steam3.mtx	80	4.99E+10	5	0.0002091	Yes
6	d_dyn.mtx	87	7.42E+06	10	0.0002744	Yes
7	d_dyn1.mtx	87	7.43E+06	11	0.0002571	Yes
8	tols90.mtx	90	2.02E+04	1	1.179E-13	Yes
9	olm100.mtx	100	1.53E+04	9	0.00011	Yes
10	tub100.mtx	100	1.33E+04	47	0.0009838	Yes
14	impcol_c.mtx	137	1.77E+04	2	1.417E-08	Yes
17	bwm200.mtx	200	2.41E+03	21	0.0007754	Yes
18	rdb200.mtx	200	3.45E+02	4	2.199E-05	Yes
19	rdb200l.mtx	200	1.33E+02	3	1.324E-05	Yes
21	ex1.mtx	216	3.30E+04	4	0.000119	Yes
22	impcol_e.mtx	225	7.10E+06	35	0.0004946	Yes
24	steam1.mtx	240	2.83E+07	1	1.235E-06	Yes
25	tols340.mtx	340	2.03E+05	1	9.408E-15	Yes
26	poisson2D.mtx	367	1.33E+02	4	7.775E-07	Yes
27	impcol_d.mtx	425	2.06E+03	10	9.025E-05	Yes
28	ex2.mtx	441	1.03E+10	51	0.007835	Yes
29	rdb450.mtx	450	6.85E+02	7	6.565E-05	Yes
30	rdb450l.mtx	450	2.10E+02	3	4.967E-06	Yes
33	steam2.mtx	600	3.78E+06	1	0.0002216	Yes
35	rdb800l.mtx	800	3.23E+02	5	3.249E-05	Yes
36	ex22.mtx	839	3.28E+04	9	3.128E-05	Yes
40	rdb968.mtx	968	3.78E+02	5	4.832E-05	Yes
11	lns_131.mtx	131	1.28E+15	51	16.736189	No
12	lnsp_131.mtx	131	1.28E+15	51	10.485417	No
13	west0132.mtx	132	4.21E+11	51	23.293316	No
15	west0156.mtx	156	2.31E+18	40	43.194152	No
16	west0167.mtx	167	4.79E+10	51	23.046129	No
20	impcol_a.mtx	207	1.35E+08	51	8.7734968	No
23	saylr1.mtx	238	7.78E+08	51	44.046211	No
31	olm500.mtx	500	3.73E+05	51	18.791045	No
32	pores_3.mtx	532	5.61E+05	51	25.920135	No
34	ex21.mtx	656	5.68E+08	51	10.6075	No
37	ex25.mtx	848	5.11E+07	51	14.635598	No
38	orsirr_2.mtx	886	6.33E+04	51	2.2424177	No
39	DK01R.mtx	903	5.89E+07	51	13.488244	No

Table 3. Computational results for the problems listed in Table 2

Results obtained for the problems selected from the SuiteSparse collection, presented in Tables 2 and 3 indicate that 27 of the 40 problems selected were solved successfully by the AM algorithm, and the average number of iterations required was equal to 11, which is comparable to the number for the randomly generated problems. The remaining 13 problems did not get solved well. These problems covered a higher range of condition numbers. It may be noted that no preconditioners

were used in this work. If preconditioners are used, even these problems may get solved successfully. This is an aspect that is being investigated.

Conclusions

The AM algorithm presented here, which is based on three earlier geometrical observations, shows a low time complexity of $O(N^{2.2})$ for randomly generated problems. It converges to the solution in a small number of iterations for N up to 1000. For problems selected from the SuiteSparse collection, most of the problems are successfully solved, and that too, without using any preconditioner. Most of the computational effort of the AM algorithm is for matrix-vector multiplications. Therefore, it can make full use of sparsity, and can be efficiently parallelised.

Nomenclature

$\mathbf{A_0}$	the coefficient matrix in the system $\mathbf{A_0x} = \mathbf{b_0}$
$\mathbf{b_0}$	the right-hand side in the system $\mathbf{A_0x} = \mathbf{b_0}$
$\mathbf{A}$	$= \mathbf{A_0}^T\mathbf{A_0}$
$\mathbf{A_k}$	$= \mathbf{A_0}^T\mathbf{A_0} + k\mathbf{v}\mathbf{v}^T$
$\mathbf{b}$	$= \mathbf{A_0}^T\mathbf{b_0}$
$\mathbf{b_k}$	$= \mathbf{A_0}^T\mathbf{b_0} + kw\mathbf{v}$
d	the distance $\mathbf{p}_1 - \mathbf{q}_{1,0}$
i	iteration number
j	loop variable
k	number of times the new equation is added to the system of equations
n_1, n_2, m_1, m_2	parameters in the AM algorithm
N	system size (the number of equations/unknowns)
$\mathbf{p}_1$, $\mathbf{p}_2$	points appearing in the AM algorithm
$\mathbf{q}_1$, $\mathbf{q}_2$	points appearing in the AM algorithm
$\mathbf{R}$	residuals = $\mathbf{b}_0 - \mathbf{A_0x}$
$\mathbf{s}$	solution vector for the system $\mathbf{A_0x} = \mathbf{b_0}$
SS_{res}	sum of squares of the residuals
$\mathbf{v}$	coefficient vector in the new equation, $\mathbf{v}^T\mathbf{x} = w$
w	right-hand side of the new equation, $\mathbf{v}^T\mathbf{x} = w$
w_1 , w_2	defined in the AM algorithm (steps 13 and 14)
$\mathbf{x}$	the vector of unknowns
$\mathbf{z}_1$, $\mathbf{z}_2$	vectors with elements N(0,1)
σ_{res}	root mean square value of the residuals (= $\mathbf{R}$)
$\sigma_{res,crit}$	critical value of σ_{res}
$\sigma_{\Delta x}$	root mean square value of the deviations (= $\mathbf{x} - \mathbf{s}$)

Subscripts

o	initial values
i	i^{th} iteration

Superscripts

T transpose of a matrix

References

[1] Gentle, J. E.,
Matrix Algebra - Theory, Computations, and Applications in Statistics
Pub. Springer, New York, USA (2007)

[2] Shewchuk J. R.,
An Introduction to the Conjugate Gradient Method
https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf (1994)

[3] Nesterov, Y.,
A Method of Solving a Convex Programming Problem with Convergence Rate O(1/k^2)
Soviet Mathematics Doklady, **27**, 372-376 (1983)

[4] Sutskever I., Martens J., Dahl G., and Hinton G.,
On the Importance of Initialization and Momentum in Deep Learning
Proceedings of the 30th International Conference on Machine Learning,
PMLR **28(3)**, 1139-1147 (2013)

[5] Strassen, V.
Gaussian Elimination is not Optimal
Numer. Math. **13 (4)**, 354–356 (1969)

[6] Coppersmith, D. and Winograd, S.
Matrix multiplication via arithmetic progressions
Journal of Symbolic Computation, 9 (3): 251 (1990)

[7] Peng R. and Vempala S.
Solving Sparse Linear Systems Faster than Matrix Multiplication
arXiv:2007.10254 [cs.DS] (2021)

[8] Patwardhan, V.,
Solution of a NxN System of Linear algebraic Equations: 1 -- The Steepest Descent Method Revisited
https://doi.org/10.48550/arXiv.2206.07482 (2022a)

[9] Patwardhan V.,
Some Geometrical Properties of a NxN System of Linear Equations
doi: 10.13140/RG.2.2.27971.27687 (2022b)

[10] Timothy A. Davis and Yifan Hu. 2011.
The University of Florida Sparse Matrix Collection
ACM Transactions on Mathematical Software 38, 1, Article 1 (December 2011), 25 pages
doi: https://doi.org/10.1145/2049662.2049663