Solution of a NxN System of Linear Algebraic Equations: The AM2 Algorithm with a Low Time Complexity

Dr. V. S. Patwardhan[1]

September 2024

Abstract

The AM2 algorithm is a much-improved version of the AM algorithm presented earlier, which solves a NxN system of linear algebraic equations using a novel idea. The main idea behind the AM algorithm was 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. In this paper, a very useful modification of the algorithm is presented. This modification involves transforming the coefficient matrix into a form which is equivalent to scaling all the variables in a special manner and improve the condition number of the matrix suitably. This modified algorithm (termed as AM2 here) was tested with (1) sample problems (with N < 1000) selected from the SuiteSparse collection of highly sparse matrices (which are widely used as benchmark matrices for testing sparse matrix algorithms and have been obtained from practical applications in several different areas such as chemical process simulation, computational fluid dynamics and many others) and (2) randomly generated dense problems for N up to 1000. The AM2 algorithm was found to solve the SuiteSparse problems with an $O[(Nnz)^{0.83}]$ time complexity. It solved the randomly generated problems in $O(N^{2.08})$ time. Comparison of the results obtained with the AM2 algorithm with those from the AM algorithm clearly show the superiority of the AM2 algorithm. The AM2 algorithm is an iterative method which terminates when residuals become less than a critical limit. A useful approach is suggested here to determine how closely the solution point itself is approached.

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 very 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, have a low time complexity, and be suitable for parallelisation. The AM2 algorithm developed here, 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 successfully.

---

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

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. 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, i.e., to reduce residuals below some small critical value. 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 are known 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 AM2 algorithm presented here comes very close to this limit.

The AM2 algorithm presented here, is a modification of the earlier AM algorithm, made by using a scaling technique. Before proceeding, it is useful to look at a summary of the AM algorithm itself.

A summary of the earlier AM algorithm (Augmented matrix algorithm)

The AM algorithm presented earlier [Patwardhan, 2022a], is based on three geometrical observations, which are described below:

1. The first observation concerns the application of the steepest descent (SD) method for solving a NxN system of linear equations. 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 avoiding this slow down [Patwardhan, 2022b], i.e., 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.
2. The second observation concerns the geometry of the intersecting hyperplanes representing a NxN system of linear equations. It has been shown earlier [Patwardhan, 2022c] 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 good convergence to the solution.

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 matrix **A**. Let the new equation be $\mathbf{v}^T\mathbf{x} = w$. Let us assume 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.) If this equation is added k times to the linear system $\mathbf{A_0 x} = \mathbf{b_0}$, 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

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

   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. The parameter k is a selectable parameter.

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 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 AM 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 **A** (= $\mathbf{A}_0^T\mathbf{A}_0$) and **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 **A** and **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 **A** and **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 **A** and **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 **A** and **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 earlier with two sets of problems. (1) Forty problems were selected from the SuiteSparse collection and solved using the AM algorithm. Twenty-seven of these were solved satisfactorily. For the others, residuals did not get reduced below the critical value even after 50 iterations, and the solution was not approached closely. The matrices for these 40 problems were very ill-conditioned, with the condition numbers covering a range of 130 to 2.31×10^{18}. The details of these calculations are available [Patwardhan, 2022a]. (2) Randomly generated problems with a problem size up to N = 1000, were solved satisfactorily and gave a very close approach to the solution. The algorithm gave a time complexity of $O(N^{2.2})$.

The AM2 algorithm

The results obtained with the AM algorithm indicate that the performance can be improved if the conditioning of the coefficient matrix can be improved. This can be achieved by transforming the $\mathbf{A_0}$ matrix by using a pre-multiplier and a post-multiplier to get a transformed system of equations. The original system $\mathbf{A_0x} = \mathbf{b_0}$ can be written as $\mathbf{A_1y} = \mathbf{b}_1$ where $\mathbf{A}_1 = \boldsymbol{\alpha}\mathbf{A_0}\boldsymbol{\beta}$, $\mathbf{b}_1 = \boldsymbol{\alpha}\mathbf{b_0}$ and $\mathbf{y} = \boldsymbol{\beta}^{-1}\mathbf{x}$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are appropriately selected diagonal matrices. The matrix $\boldsymbol{\beta}$ essentially represents a diagonal scaling of the $\mathbf{x}$ variables which changes the condition number of the coefficient matrix. If the system $\mathbf{A}_1\mathbf{y} = \mathbf{b}_1$ can be solved for $\mathbf{y}$, then we can recover $\mathbf{x}$, the solution of the original system (i.e., $\mathbf{A_0x} = \mathbf{b_0}$) simply by using $\mathbf{x} = \boldsymbol{\beta}\mathbf{y}$. These equations are valid for any arbitrary choice of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

Selection of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$:

The easiest way to get rid of large numbers in the coefficient matrix is to normalize all rows of $\mathbf{A_0}$, which is equivalent to normalizing each equation in the system $\mathbf{A_0x} = \mathbf{b_0}$. Taking this further, is it possible to simultaneously normalize the columns of $\mathbf{A_0}$ as well?

Consider a matrix $\mathbf{A}_{sq}$ whose entries are squares of corresponding entries of $\mathbf{A_0}$, and therefore are nonnegative. If $\mathbf{A_0}$ can be put in a form where all its rows as well as columns are normalized (i.e., their Euclidian norm is 1), then the corresponding $\mathbf{A}_{sq}$ matrix would be double stochastic (i.e., each row as well as each column would add up to 1). A simple iterative method to get the double stochastic matrix is to alternately rescale all rows and all columns of $\mathbf{A}_{sq}$ to sum to 1. This is the well-known Sinkhorn and Knopp algorithm [Sinkhorn and Knopp, 1967]. This algorithm can be shown to converge to the double stochastic form, if the matrix has support, i.e., it contains at least one diagonal with only positive elements. This condition is satisfied if the matrix is invertible. The final double stochastic form can be transformed to the final form of $\mathbf{A_0}$ just by taking square roots of corresponding entries in $\mathbf{A}_{sq}$, keeping the sign of each entry as in the original $\mathbf{A_0}$. Alternately, row and column normalization (using the Euclidian norm) can be applied to the original $\mathbf{A_0}$ repeatedly to get the final form. It may be noted that converting $\mathbf{A}_{sq}$ to the double stochastic form is computationally more efficient than converting $\mathbf{A_0}$ by repeated normalization of rows and columns, as the latter involves squaring entries at each iteration. However, this choice is not important, because calculations show that the time taken for double normalization is negligible compared to the total time taken by the AM2 algorithm.

Double normalisation (i.e., normalisation of rows as well as columns) is an idea that is used widely in data analysis in various fields (for example, see Olshen and Rajaratnam [2010]). Many mathematical libraries provide standard callable procedures for double normalization using different norms such as maximum entry, sum of absolute values etc, as well as the Euclidian norm.

It is obvious that normalizing matrix rows can be represented as pre-multiplication by a diagonal matrix, while normalizing matrix columns can be represented as post-multiplication by another diagonal matrix. While normalizing rows and columns repeatedly, these diagonal matrices can be collected at each step, and can be combined to get $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. (The details of these calculations are described in Appendix 1.) Let us now assume that $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathbf{A}_1$ and $\mathbf{b}_1$ have been calculated from $\mathbf{A_0}$ and $\mathbf{b}_0$. At this point, the AM algorithm described above can be applied to the system $\mathbf{A}_1\mathbf{y} = \mathbf{b}_1$ to get the solution $\mathbf{y}$, and then the solution of the original system, $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$, can be calculated using $\mathbf{x} = \boldsymbol{\beta}\mathbf{y}$.

The AM2 algorithm thus involves (1) normalization of the rows as well as columns of the $\mathbf{A}_0$ matrix and (2) subsequent application of the AM algorithm presented earlier. The following sections describe the numerical results obtained with the AM2 algorithm, using two different sets of problems: (1) problems selected from the SuiteSparse collection of matrices, and (2) randomly generated test problems. The results are also compared with the results of the AM algorithm (which does not involve double normalization).

Results using problems from the SuiteSparse collection

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 and Hu, 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.

The AM2 algorithm described above, was used for solving a set of selected test problems 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 1. Totally 40 problems were selected.

The problems listed in Table 1 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 AM2 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 AM2 algorithm.) A point 10 units away from the origin, in a random direction, was taken as the starting point for the AM2 algorithm.

Computational results:

The parameter values selected were k = 30, m_1 = 20, m_2 = 10, n_1 = 40 and n_2 = 20. The maximum number of iterations permitted was fixed at 50. The AM2 algorithm was applied to the system $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$. The progress of iterations was monitored by (1) calculating σ_{Res} (i.e., the root mean square of the

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 1. Problems selected from the SuiteSparse collection

residuals, R_i for the current **y**), and (2) calculating current **x** from current **y** to get the deviations ($\mathbf{x}_i$ - $\mathbf{s}_i$). The 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 the system $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$, for i = 1 to N).

No.	Problem name	N	Nnz	Iter reqd.	Run time, reduced	σ_{Res}	$\sigma_{\Delta x}$
1	pores_1.mtx	30	180	5	0.108	2.85E-09	6.73E-07
2	d_ss.mtx	53	149	5	0.115	3.22E-08	2.24E-07
3	impcol_b.mtx	59	312	3	0.231	3.98E-13	7.52E-11
4	west0067.mtx	67	294	3	0.114	9.40E-07	1.45E-05
5	steam3.mtx	80	928	1	0.081	1.01E-09	9.57E-07
6	d_dyn.mtx	87	238	1	0.030	7.45E-11	1.22E-10
7	d_dyn1.mtx	87	238	2	0.057	2.03E-13	1.32E-12
8	tols90.mtx	90	1746	1	0.122	6.33E-15	1.82E-14
9	olm100.mtx	100	396	5	0.191	8.10E-07	1.79E-04
10	tub100.mtx	100	396	40	1.552	8.55E-07	8.61E-04
11	lns_131.mtx	131	536	3	0.159	6.54E-08	1.92E-05
12	lnsp_131.mtx	131	536	1	0.052	8.03E-07	7.43E-05
13	west0132.mtx	132	414	5	0.214	1.16E-09	3.39E-08
14	impcol_c.mtx	137	411	2	0.086	3.34E-08	2.69E-07
15	west0156.mtx	156	371	2	0.088	1.02E-07	3.80E-01
16	west0167.mtx	167	507	4	0.209	2.24E-10	3.42E-09
17	bwm200.mtx	200	796	28	2.067	9.67E-07	8.09E-04
18	rdb200.mtx	200	1120	4	0.370	6.47E-07	9.35E-06
19	rdb200l.mtx	200	1120	3	0.278	7.73E-08	6.77E-07
20	impcol_a.mtx	207	572	4	0.252	4.58E-08	6.49E-06
21	ex1.mtx	216	4352	1	0.295	7.90E-13	9.25E-11
22	impcol_e.mtx	225	1308	1	0.109	8.04E-11	2.35E-09
23	saylr1.mtx	238	1128	50	4.978	1.39E-04	4.14E+01
24	steam1.mtx	240	3762	1	0.257	6.59E-15	1.22E-11
25	tols340.mtx	340	2196	1	0.178	9.70E-15	1.14E-14
26	poisson2D.mtx	367	2417	4	0.770	1.27E-08	2.69E-07
27	impcol_d.mtx	425	1339	5	0.680	2.65E-09	9.56E-08
28	ex2.mtx	441	13640	5	4.249	1.23E-09	6.08E-08
29	rdb450.mtx	450	2580	8	1.694	8.70E-07	1.73E-04
30	rdb450l.mtx	450	2580	4	0.846	5.57E-08	3.24E-06
31	olm500.mtx	500	1996	50	9.186	5.26E-04	4.38E+00
32	pores_3.mtx	532	3474	50	13.823	1.79E-04	1.69E+01
33	steam2.mtx	600	13760	1	0.882	2.72E-15	3.29E-11
34	ex21.mtx	656	19144	4	4.788	6.14E-08	1.47E-03
35	rdb800l.mtx	800	4640	5	1.880	1.41E-07	1.52E-05
36	ex22.mtx	839	22715	4	5.696	8.01E-08	2.59E-06
37	ex25.mtx	848	24612	5	9.896	8.23E-08	1.62E-04
38	orsirr_2.mtx	886	5970	50	29.409	6.85E-05	2.92E-01
39	DK01R.mtx	903	11766	6	4.898	3.68E-08	2.55E-04
40	rdb968.mtx	968	5632	5	2.474	6.09E-07	7.58E-05

Table 2. Computational results with the AM2 algorithm for problems selected from The SuiteSparse collection (N <= 1000)

The run time, in seconds, of any algorithm depends not only on the algorithm itself, but other factors such as the computer speed, the coding language, the development environment used etc. In order to eliminate the effect of these external factors, a reference run time T_{ref} was obtained for a standard procedure, and the run times obtained for different problems were normalised by dividing by T_{ref} to get a run time ratio. The standard procedure selected was the solution of a dense random

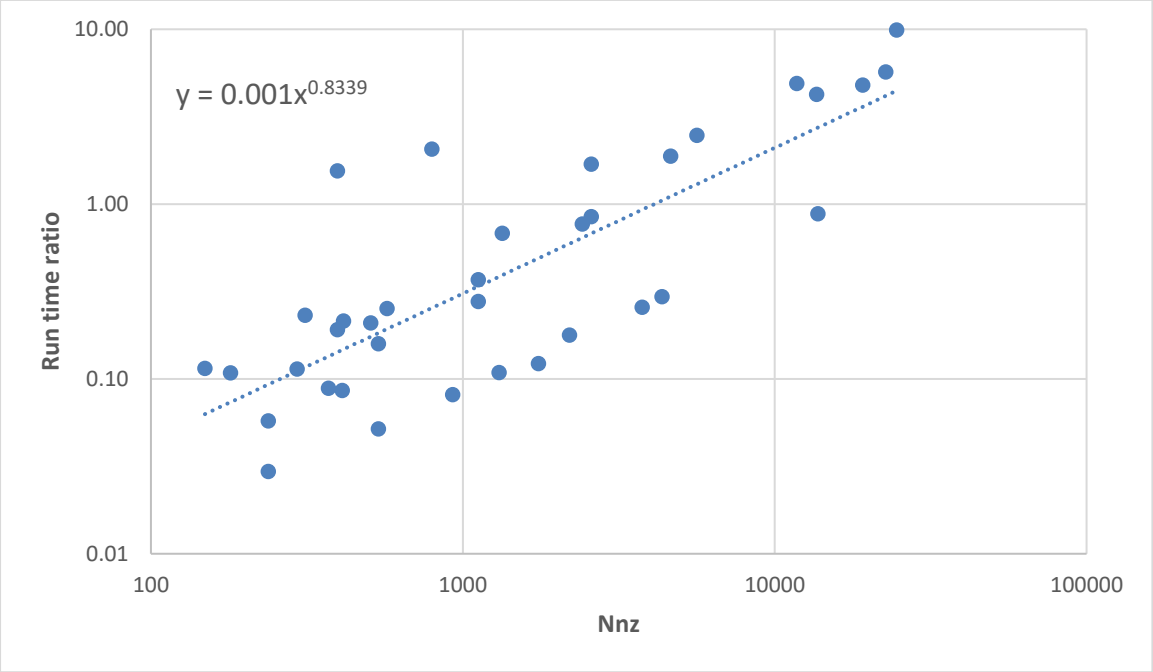

Figure 1. Run time ratio vs. Nnz for different SuiteSparse problems.

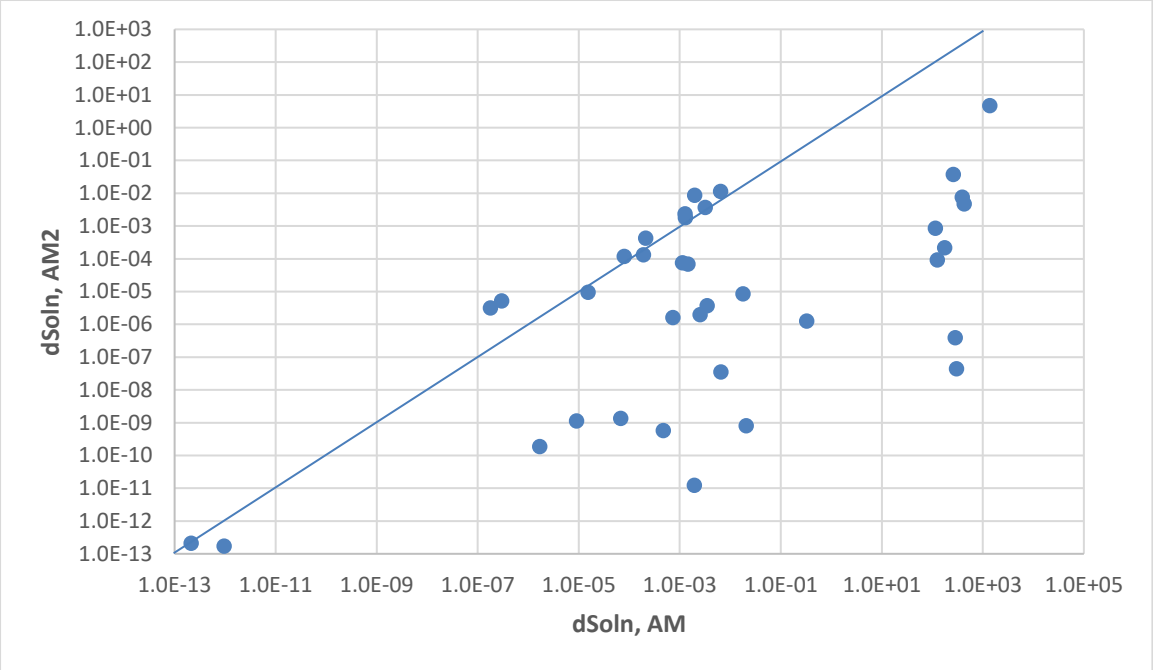

Figure 2. A comparison of approach to the solution with/without double normalization

square matrix of dimension 1000 by using Gaussian elimination. This way, the run time ratios reported here should be reproducible on other computer systems.

Table 2 shows the computational results obtained with the AM2 algorithm for the problems listed in Table 1. It is seen that 36 of the 40 problems gave convergence in less than 50 iterations, and thus were solved very well by the AM2 algorithm. For these problems, the AM2 algorithm rapidly approaches the solution, often in just a few iterations. The average deviation in solution coordinates is less than 0.01, and the average number of iterations required is 5. For the 4 problems (problem numbers 23, 31, 32 and 38 in Table 2) which are not solved within 50 iterations, the average deviation in solution coordinates is 15.73. For these problems, the speed of approaching the solution becomes too low after some point. It may be noted that the AM2 algorithm does not make use of any explicit preconditioners. For these four problems, preconditioners may have to be used. The reduced run time for the 36 problems is shown in Figure 1 against Nnz, the number of nonzero elements. The time complexity is seen to be $O(Nnz^{0.83})$. It may be noted here that Nnz is used instead of N, because the computation procedure uses sparse matrix routines, whose computation time is related to Nnz, and not N. For dense matrices, $Nnz = N^2$, while for sparse matrices, $Nnz << N^2$, as seen in Table 1.

It has been mentioned earlier that the AM2 algorithm consists of double normalization followed by the AM algorithm. It is interesting to see the effect of the double normalization on the results. It may be noted that d_{soln}, i.e., the distance between the approximate and the exact solutions, is equal to $\sigma_{\Delta x}(N)^{0.5}$. Figure 2 shows $d_{soln,AM2}$ and $d_{soln,AM}$, i.e., the d_{soln} values for the AM2 and the AM algorithms respectively, for the 36 SuiteSparse problems. The line represents equality of the two values. It is seen that the AM2 algorithm approaches the solution much more closely than the AM algorithm. This is the result of the double normalization.

It is interesting to see some results obtained during the double normalization of the $\mathbf{A}_0$ matrix. The detailed procedure for double normalization is shown in Appendix 1. Table 3 shows a summary of the elements of $\boldsymbol{\beta}$ matrices obtained for selected SuiteSparse problems. The elements of $\boldsymbol{\beta}$ vary between 1.66×10^{-5} and 1.04×10^{7}, which is a very wide range. It has been mentioned earlier that the $\boldsymbol{\beta}$ values represent scaling of the x-variables in the original problem. These $\boldsymbol{\beta}$ values indicate a very strong scaling which is a result of the double normalization procedure. This is the reason for the superior performance of the AM2 algorithm over the AM algorithm. The wide range of $\boldsymbol{\beta}$ values is related to the very high condition numbers of the SuiteSparse problems. The average number of iterations taken by the double normalization procedure is 25. These results are quite in contrast to similar results presented later for randomly generated problems.

The test problems considered so far were those selected from the SuiteSparse collection. It is interesting to examine similar results obtained for randomly generated problems involving gaussian matrices.

Results using randomly generated test problems

Randomly generated test problems have certain properties arising out of the random procedure used for generating these, and the computational results obtained may be different from those presented above for the SuiteSparse problems. However, it is interesting to see the results obtained with randomly generated problems.

No.	fName	N	Nnz	**β** Max	**β** Min	**β** Ratio	Iter-ations
1	pores_1.mtx	30	180	4.17E+02	1.31E-01	3.15E-04	14
2	d_ss.mtx	53	149	1.81E+01	9.71E-03	5.37E-04	26
3	impcol_b.mtx	59	312	5.73E+01	1.92E-02	3.35E-04	43
4	west0067.mtx	67	294	7.93E+00	4.56E-01	5.75E-02	11
5	steam3.mtx	80	928	4.56E+03	2.83E-01	6.19E-05	19
6	d_dyn.mtx	87	238	1.84E+01	1.21E-03	6.55E-05	38
7	d_dyn1.mtx	87	238	6.22E+01	8.25E-04	1.33E-05	47
8	tols90.mtx	90	1746	3.49E+00	8.02E-01	2.30E-01	2
9	olm100.mtx	100	396	1.92E+00	6.23E-01	3.24E-01	7
10	tub100.mtx	100	396	1.24E+00	8.72E-01	7.02E-01	1
11	lns_131.mtx	131	536	3.77E+02	1.19E-03	3.16E-06	47
12	lnsp_131.mtx	131	536	3.77E+02	1.19E-03	3.16E-06	47
13	west0132.mtx	132	414	3.24E+03	1.45E-04	4.49E-08	74
14	impcol_c.mtx	137	411	2.24E+01	4.77E-02	2.13E-03	46
15	west0156.mtx	156	371	1.04E+07	1.30E-03	1.26E-10	68
16	west0167.mtx	167	507	3.67E+03	2.82E-04	7.67E-08	62
17	bwm200.mtx	200	796	1.02E+00	9.83E-01	9.67E-01	1
18	rdb200.mtx	200	1120	1.45E+00	7.78E-01	5.36E-01	4
19	rdb200l.mtx	200	1120	1.27E+00	8.50E-01	6.67E-01	1
20	impcol_a.mtx	207	572	1.01E+04	5.61E-04	5.53E-08	81
21	ex1.mtx	216	4352	6.41E+01	4.29E-01	6.69E-03	11
22	impcol_e.mtx	225	1308	4.85E+03	9.03E-03	1.86E-06	69
23	saylr1.mtx	238	1128	3.17E+00	4.60E-01	1.45E-01	12
24	steam1.mtx	240	3762	1.80E+03	4.77E-01	2.65E-04	3
25	tols340.mtx	340	2196	2.67E+00	8.65E-01	3.24E-01	1
26	poisson2D.mtx	367	2417	1.01E+00	9.70E-01	9.57E-01	1
27	impcol_d.mtx	425	1339	1.23E+01	2.55E-02	2.08E-03	51
28	ex2.mtx	441	13640	9.85E+06	1.66E-05	1.68E-12	100
29	rdb450.mtx	450	2580	1.17E+00	8.82E-01	7.57E-01	7
30	rdb450l.mtx	450	2580	1.95E+00	7.50E-01	3.84E-01	2
31	olm500.mtx	500	1996	1.89E+00	6.29E-01	3.32E-01	7
32	pores_3.mtx	532	3474	5.91E+01	1.37E-01	2.31E-03	18
33	steam2.mtx	600	13760	4.37E+03	3.09E-01	7.06E-05	18
34	ex21.mtx	656	19144	5.55E+02	4.23E-02	7.62E-05	13
35	rdb800l.mtx	800	4640	1.52E+00	7.69E-01	5.05E-01	4
36	ex22.mtx	839	22715	2.71E+01	2.44E-01	8.99E-03	6
37	ex25.mtx	848	24612	1.42E+03	9.48E-02	6.67E-05	9
38	orsirr_2.mtx	886	5970	1.28E+00	8.57E-01	6.70E-01	3
39	DK01R.mtx	903	11766	6.19E+03	6.35E-02	1.02E-05	18
40	rdb968.mtx	968	5632	1.40E+00	7.86E-01	5.60E-01	5

Table 3. A summary of **β** values for SuiteSparse matrices (N upto 1000)

Problem generation:

Random problems were generated such that (1) the solution point **s** was at a fixed distance from the origin, but in a random direction, (2) the entries in the coefficient matrix $\mathbf{A}_0$ were generated as

No.	N	σ_{Res}	$\sigma_{\Delta x}$	Run time ratio	iterations required
1	10	1.1E-05	3.41E-05	0.004038	1.0
2	14	6.07E-06	8.6E-05	0.007279	1.6
3	20	1.66E-05	0.000263	0.013062	1.8
4	32	4.31E-05	0.001187	0.032457	2.8
5	50	1.61E-05	0.00031	0.057883	2.6
6	70	4.96E-05	0.091657	0.089442	2.2
7	100	2.25E-05	0.001454	0.214981	2.8
8	140	5.45E-05	0.011221	0.608348	4.0
9	200	6.72E-05	0.024496	1.401912	4.8
10	320	7.07E-05	0.023607	2.646428	3.2
11	500	7.68E-05	0.011599	10.9888	5.0
12	700	8.82E-05	0.018434	23.82379	5.2
13	1000	8.28E-05	0.018607	58.41226	6.2

Table 4. Computational results obtained for randomly generated Problems, N = 10-1000 (average of 5 problems for each N)

N(0,1), and then 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 point **s** was used only for generating the right-hand side and was never used while solving the resulting system of equations (i.e., $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$) using the AM2 algorithm. The details of this procedure were described earlier [Patwardhan, 2022c]. N was varied from 10 to 1000 using 13 values of N which were almost uniformly spaced on a logarithmic scale.

Computational results:

The parameter values selected were k = 4, m_1 = 20, m_2 = 5, n_1 = 10 and n_2 = 20. The AM2 algorithm was applied to the system $\mathbf{A}_0\mathbf{x} = \mathbf{b}_0$. The origin was taken as the starting point. During the iterative calculations, the known solution point was used only for characterising the successive iterates as stated above and was completely ignored in the solution process. The algorithm was tested for N = 10 to 1000. For each N, five different problems were generated independently. The running time refers only to the time taken for calculating the solution, and excludes the time taken for reading or generating the problem data.

The computational results obtained with these randomly generated problems are shown in Table 4. Figure 3 shows a log-log plot of run time ratio vs. problem size N. Figure 4 shows a similar plot of the number of iterations required vs. problem size N. Table 5 shows results obtained during double normalization.

It is seen from Table 4 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 of **s** get calculated within about 0.02 or so. It also shows that as N goes from 10 to 1000, the number of iterations increase only from 1 to 6.2. In other words, for a 100-fold increase in N, the number of iterations increases only 6 times or so. Figure 3 shows that the time complexity of the AM2

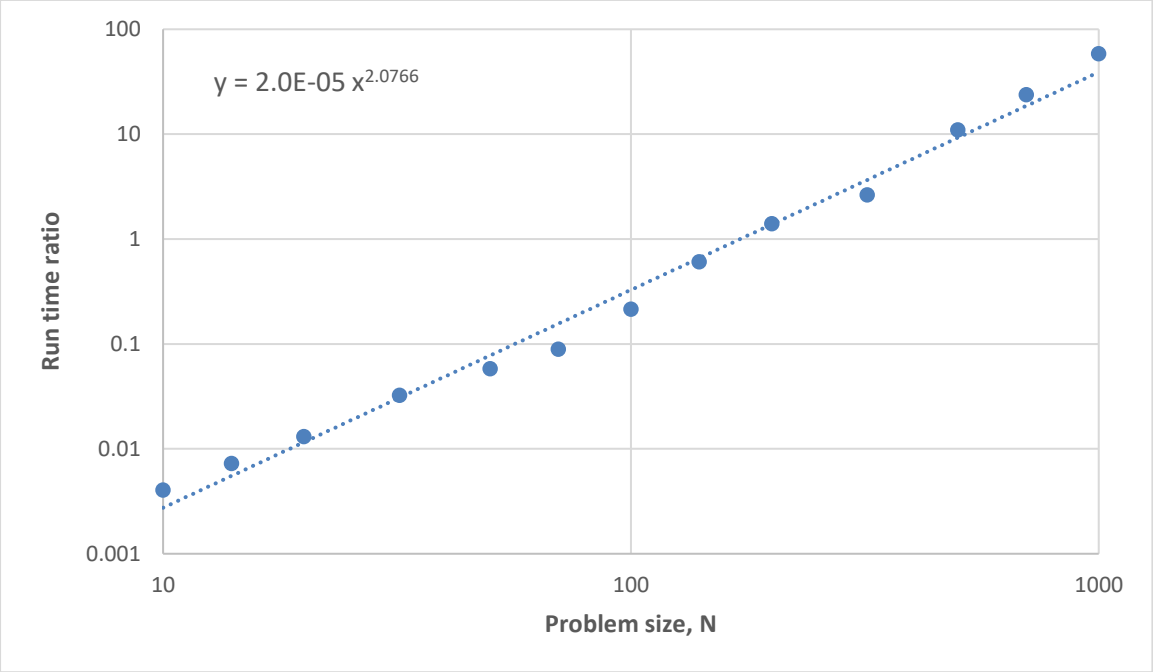

Figure 3. Run time ratio vs. N for randomly generated problems

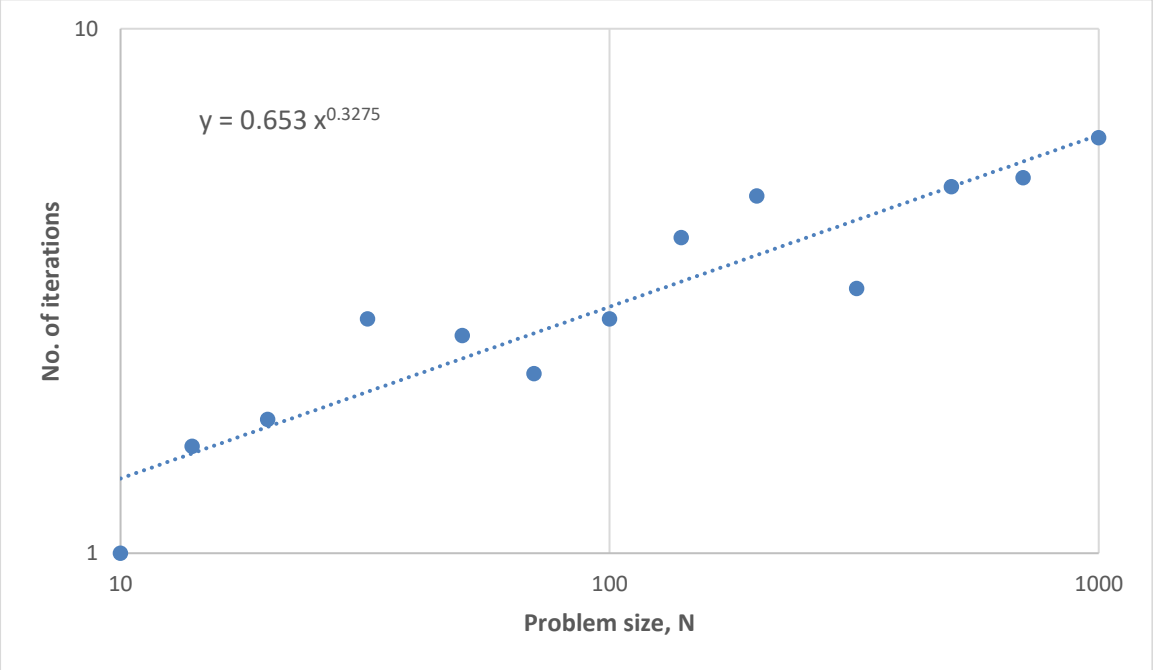

Figure 4. No. of iterations required for different problem sizes (N)

No.	N	**β** Max	**β** Min	**β** Ratio	Iter-ations
1	10	1.69	0.70	0.43	2.6
2	14	1.39	0.75	0.55	2.2
3	20	1.41	0.79	0.59	2.0
4	32	1.40	0.81	0.58	2.0
5	50	1.36	0.80	0.59	1.8
6	70	1.21	0.84	0.70	1.4
7	100	1.21	0.85	0.70	1.0
8	140	1.19	0.88	0.74	1.0
9	200	1.15	0.87	0.76	1.0
10	320	1.11	0.88	0.80	1.0
11	500	1.10	0.91	0.83	1.0
12	700	1.09	0.92	0.85	1.0
13	1000	1.08	0.93	0.86	1.0

Table 5. A summary of **β** for randomly generated problems, N = 10-1000 (Average of 5 problems for each N)

algorithm is about $O(N^{2.08})$. This is better than computational complexities reported so far and is surprisingly close to the theoretical limit of 2.0. Figure 4 shows that the number of iterations increase very slowly with N, and show a dependence of $O(N^{0.3})$ or so.

It is interesting to see some results obtained during the double normalization of the $\mathbf{A}_0$ matrix. The detailed procedure for double normalization is shown in Appendix 1. Table 5 shows a summary of the elements of **β** matrices obtained for randomly generated problems. The elements of **β** vary between 0.7 and 1.69, which is a narrow range close to 1 (in contrast to the very large range presented above for the SuiteSparse problems). It appears that since the matrices themselves are generated at random, it does not take much correction to achieve normalization of both rows and columns, and it takes only a few iterations to achieve it. This is in contrast with the large number of iterations required for SuiteSparse matrices, as presented above.

Discussion

Problems from the SuiteSparse collection:

The condition numbers for the 40 problems selected from the SuiteSparse collection were in the very wide range of 130 to 2.31×10^{18}. These represent very elongated ellipsoids as the quadratic SS_{res} surfaces. The AM2 algorithm essentially takes steepest descent steps alternately with the original and the augmented matrix systems. It approaches the solution closely in most of the cases and solves 36 of the 40 problems very well. However, for the remaining 4 problems, it gets stuck at a point quite far away from the solution. The average d_{soln} for the 36 problems is 0.13, while that for the 4 problems is 283.5! (Out of the 36 problems, problem no. 15 has $d_{soln} = 4.75$, which is high despite the good convergence in two steps. If this point is left out, the average d_{soln} for the 35 problems is 0.0023, which is much better.) This can be interpreted in terms of the geometry of the N hyperplanes. The 4 problems give approximate solution points which have small residual values from

No.	Problem Name	N	d_{Soln}, RHS=**0**	d_{Soln}, RHS=$\mathbf{b_0}$
1	pores_1.mtx	30	1.09E-05	3.69E-06
2	d_ss.mtx	53	4.38E-06	1.63E-06
3	impcol_b.mtx	59	1.57E-05	5.77E-10
4	west0067.mtx	67	3.58E-05	1.19E-04
5	steam3.mtx	80	1.63E-05	8.56E-06
6	d_dyn.mtx	87	1.15E-13	1.14E-09
7	d_dyn1.mtx	87	5.25E-06	1.23E-11
8	tols90.mtx	90	1.17E-26	1.72E-13
9	olm100.mtx	100	2.46E-05	1.79E-03
10	tub100.mtx	100	5.83E-03	8.61E-03
11	lns_131.mtx	131	2.03E-04	2.20E-04
12	lnsp_131.mtx	131	4.82E-07	8.50E-04
13	west0132.mtx	132	7.87E-06	3.89E-07
14	impcol_c.mtx	137	1.31E-05	3.15E-06
15	west0156.mtx	156	1.09E+01	4.75E+00
16	west0167.mtx	167	3.68E-06	4.42E-08
17	bwm200.mtx	200	7.73E-03	1.14E-02
18	rdb200.mtx	200	3.05E-04	1.32E-04
19	rdb200l.mtx	200	5.89E-04	9.58E-06
20	impcol_a.mtx	207	1.50E-06	9.34E-05
21	ex1.mtx	216	2.21E-11	1.36E-09
22	impcol_e.mtx	225	3.14E-10	3.52E-08
23	saylr1.mtx	238	6.49E+01	6.38E+02
24	steam1.mtx	240	5.07E-22	1.89E-10
25	tols340.mtx	340	2.79E-22	2.11E-13
26	poisson2D.mtx	367	6.30E-06	5.16E-06
27	impcol_d.mtx	425	2.00E-06	1.97E-06
28	ex2.mtx	441	1.20E-05	1.28E-06
29	rdb450.mtx	450	1.24E-03	3.66E-03
30	rdb450l.mtx	450	1.97E-05	6.87E-05
31	olm500.mtx	500	4.14E+01	9.79E+01
32	pores_3.mtx	532	5.55E+01	3.89E+02
33	steam2.mtx	600	1.31E-59	8.05E-10
34	ex21.mtx	656	5.85E-03	3.78E-02
35	rdb800l.mtx	800	1.45E-04	4.29E-04
36	ex22.mtx	839	2.73E-05	7.49E-05
37	ex25.mtx	848	2.66E-05	4.73E-03
38	orsirr_2.mtx	886	2.67E-01	8.68E+00
39	DK01R.mtx	903	2.25E-02	7.67E-03
40	rdb968.mtx	968	3.08E-03	2.36E-03

Table 6. d_{soln} with the AM2 algorithm, with original $\mathbf{b_0}$, and with $\mathbf{b_0} = \mathbf{0}$

the N equations, even though they are far away from the exact solution. This means the cone made by the hyperplanes (with its apex at **s**) which contains the approximate solution point is extremely sharp. This is a geometrical characteristic of the N hyperplanes which prevents good convergence.

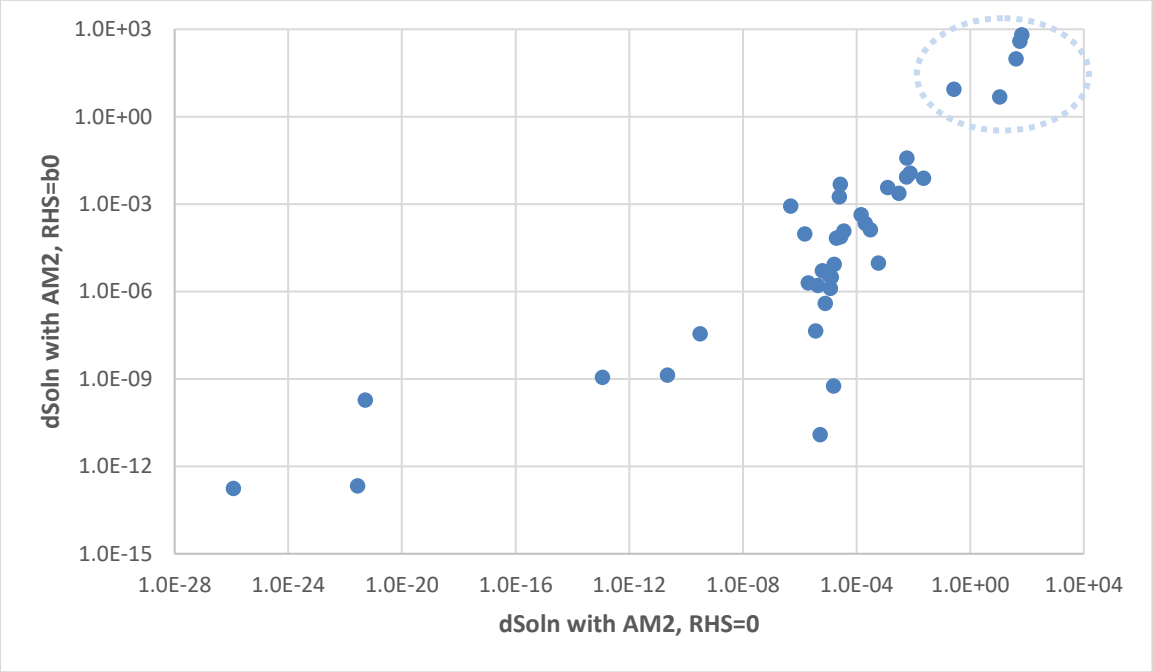

Figure 5. d_{soln} with the AM2 algorithm for SuiteSparse problems, with RHS = $\mathbf{b_0}$, and with RHS = **0**

This brings us to an important question. In this study, we have characterised the final solution point in terms of the residuals, as well as in terms of its distance from the exact solution. However, when we solve a new practical problem with the AM2 algorithm, we will get the approximate solution based on the reduction of σ_{Res} below a critical value. However, the exact solution will be unknown! In such a situation, it is possible to get some idea about d_{soln}? It has been argued that the ease (or otherwise) of convergence of the AM2 algorithm is related to the geometrical structure of the system of equations (i.e., the N hyperplanes) itself. If we change the right-hand side, i.e., $\mathbf{b_0}$ in the system of equations $\mathbf{A_0x} = \mathbf{b_0}$, without changing $\mathbf{A_0}$, then the geometrical characteristics of the hyperplanes would remain unchanged. Changing $\mathbf{b_0}$ is equivalent to a translation of the hyperplanes to a new position, without changing their orientations. This changes the solution point, but the convergence behaviour of the AM2 algorithm remains unchanged. Let us change the right-hand side to zero, and consider the homogeneous system $\mathbf{A_0x} = \mathbf{0}$, where **0** is a vector of zeros. This system has the origin itself as the solution, i.e., $\mathbf{s} = \mathbf{0}$. Solving this system with the AM2 algorithm will give us the approximate solution (obtained by reducing σ_{Res} to a value smaller than a critical value, subject to the maximum limit on the number of iterations). Let $d_{soln,0}$ be the distance between the approximate and the exact solution for this homogeneous system. Now we can calculate $d_{soln,0}$ since we know **s**! All the 40 SuiteSparse problems selected were solved using this approach, and the $d_{soln,0}$ values obtained are shown in Table 6 along with the d_{soln} values obtained for the original problems. Figure 5 shows these values graphically.

It is seen from figure 5 that the d_{soln} and $d_{soln,0}$ vary monotonically. The five points inside the dashed ellipse show that when the $d_{soln,0}$ is high, the d_{soln} values are also high (i.e., 0.3 to 1000). On the other hand, for all the other points, both $d_{soln,0}$ and d_{soln} values are small, less than 0.02 or so. This gives us a way of estimating the accuracy of the approximate solution of a new problem (given by $\mathbf{A_0x} = \mathbf{b_0}$)

obtained by using the AM2 algorithm. All we need to do is to solve the corresponding homogeneous system, i.e., $\mathbf{A_0x} = \mathbf{0}$, to get the approximate solution with the AM2 algorithm, and get $d_{soln,0}$, which is just the distance between the approximate solution and the origin (which is the exact solution of the homogeneous system). If $d_{soln,0}$ is small enough, then the approximate solution of $\mathbf{A_0x} = \mathbf{b_0}$ will also be close enough to **s**. However, if $d_{soln,0}$ is rather large, then the approximate solution of $\mathbf{A_0x} = \mathbf{b_0}$ will not be very accurate, and some other approach such as using a preconditioner needs to be considered, if a greater accuracy is required. Many times, the solution obtained by reducing σ_{Res} below a critical value, may be quite acceptable in practice, though d_{soln} may not be too small.

Randomly generated problems:

The results presented in Table 4 and Figures 3 and 4 indicate that the AM2 algorithm successfully solved all the 65 randomly generated problems with N ≤ 1000. The final solution given by the AM2 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 $O(N^{2.08})$, which is very attractive, and compares very closely with the theoretical minimum of $O(N^2)$. The number of iterations shows a weak dependence on problem size, as seen in Figure 4. Table 5 shows that the **β** values obtained in double normalization cover a rather narrow range of 0.70 to 1.69. (In contrast, the **β** values for SuiteSparse problems cover a very wide range of 1.66×10^{-5} to 1.04×10^{7}.) It is obvious that the matrices, generated at random, do not need much effort to achieve normalization of both rows and columns, and it takes only a few iterations to achieve it. If these problems are solved with the AM algorithm instead of the AM2 algorithm, the results are not significantly different, and are not presented here, for the sake of brevity.

The computation time complexity for the SuiteSparse problems is $O(Nnz^{0.83})$, and that for randomly generated dense problems is $O(N^{2.08})$. These are quite comparable since $Nnz = N^2$ for dense matrices.

Conclusions

1. The AM2 algorithm presented here consists of double normalization of the coefficient matrix, followed by the application of the AM algorithm presented earlier. The performance of the AM2 algorithm is shown to be much superior compared to that of the AM algorithm. Thus, double normalization plays a very important role in the AM2 algorithm. Double normalization is shown to be equivalent to scaling the variables. The AM2 algorithm solved most (36 out of 40) of the problems selected from the SuiteSparse collection (up to N < 1000) successfully and gave a time complexity of $O(Nnz^{0.83})$, where Nnz is the number of nonzero elements. This low time complexity makes the AM2 algorithm attractive for huge systems (N >> 1000).
2. The convergence criterion used by the AM2 algorithm consists of reduction of σ_{Res} below a critical value. A procedure is suggested here for estimating the distance between the exact and approximate solutions, which is based on solving a homogeneous system.
3. The AM2 algorithm was also used for solving randomly generated problems for N = 10 to 1000. All 65 problems were solved successfully, and the approximate solution was found to be very close to the exact solution. The time complexity was found to be in $O(N^{2.08})$, which is almost the same as the theoretical minimum of $O(N^2)$.
4. The scaling produced by the double normalization procedure was found to be severe for the highly ill-conditioned SuiteSparse problems, while it was rather mild for the randomly generated

problems. Most of the computational effort of the AM2 algorithm is for matrix-vector multiplications. Therefore, it can make full use of sparsity, and can be efficiently parallelized.

Nomenclature

$\mathbf{A}$	$= \mathbf{A_0}^T\mathbf{A_0}$
$\mathbf{A_0}$	the coefficient matrix in the system $\mathbf{A_0x} = \mathbf{b_0}$
$\mathbf{A_1}$	$= \boldsymbol{\alpha}\mathbf{A_0}\boldsymbol{\beta}$, the matrix obtained by double normalization of $\mathbf{A_0}$
$\mathbf{A_k}$	$= \mathbf{A_0}^T\mathbf{A_0} + k\mathbf{v}\mathbf{v}^T$
$\mathbf{A}_{sq}$	a matrix whose entries are squares of corresponding entries of $\mathbf{A_0}$
$\mathbf{b}$	$= \mathbf{A_0}^T\mathbf{b_0}$
$\mathbf{b_0}$	the right-hand side in the system $\mathbf{A_0x} = \mathbf{b_0}$
$\mathbf{b}_1$	$= \boldsymbol{\alpha}\mathbf{b_0}$
$\mathbf{b_k}$	$= \mathbf{A_0}^T\mathbf{b_0} + kw\mathbf{v}$
d	the distance $\mathbf{p}_1 - \mathbf{q}_{1,0}$
d_{soln}	distance from the exact solution $\mathbf{s}$
$d_{soln,0}$	distance from the origin
$d_{soln,AM}$	d_{soln} obtained with the AM algorithm
$d_{soln,AM2}$	d_{soln} obtained with the AM2 algorithm
i	iteration number
$\mathbf{I}$	the identity matrix
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)
Nnz	number of nonzero elements in matrix $\mathbf{A}$
$\mathbf{p}_1$, $\mathbf{p}_2$	points appearing in the AM algorithm
$\mathbf{q}_1$, $\mathbf{q}_2$	points appearing in the AM algorithm
$\mathbf{R}$	vector of residuals, $= \mathbf{b}_0 - \mathbf{A_0x}$, or $= \mathbf{b}_1 - \mathbf{A}_1\mathbf{y}$
R_i	Residual for the i^{th} equation
$\mathbf{s}$	solution vector for the system $\mathbf{A_0x} = \mathbf{b_0}$
SS_{res}	sum of squares of the residuals
T_{ref}	run time for a standard procedure
$\mathbf{v}$	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
$\mathbf{y}$	$= \boldsymbol{\beta}^{-1}\mathbf{x}$, the vector of transformed unknowns
$\mathbf{z}_1$, $\mathbf{z}_2$	vectors with elements N(0,1)
$\mathbf{0}$	a vector of zeros
$\boldsymbol{\alpha}$	a diagonal matrix, obtained during double normalization of $\mathbf{A_0}$
$\boldsymbol{\beta}$	a diagonal scaling matrix, obtained during double normalization of $\mathbf{A_0}$
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 (= **x** – **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: A New Algorithm with a Low Time Complexity
doi: https://doi.org/10.13140/RG.2.2.12918.27204 (2022a)

[9] 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 (2022b)

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

[11] Sinkhorn, R. and Knopp, P.,
Concerning nonnegative matrices and doubly stochastic matrices
Pacific J. Math. **21**, 343–348 (1967)

[12] Olshen R. A. and Rajaratnam B.,
Successive Normalization of Rectangular Arrays
Ann Stat., **38(3)**, 1638-1664 (2010)

[13] Davis T. A. and Yifan Hu.
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 (2011)

Appendix 1

It has been mentioned in the main text that the original system, $\mathbf{A_0 x} = \mathbf{b_0}$ can be written as $\mathbf{A_1 y} = \mathbf{b}_1$ where $\mathbf{A}_1 = \boldsymbol{\alpha}\mathbf{A_0}\boldsymbol{\beta}$, $\mathbf{b}_1 = \boldsymbol{\alpha}\mathbf{b_0}$ and $\mathbf{y} = \boldsymbol{\beta}^{-1}\mathbf{x}$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are appropriately selected diagonal matrices. The matrix $\boldsymbol{\beta}$ essentially represents a diagonal scaling of the **x** variables which changes the condition number of the coefficient matrix.

Double normalization (i.e., repeated normalization of both rows and columns, alternately) eventually gives a matrix whose rows as well as columns are normalized. Here we use the Euclidian norm for normalization. In this appendix we describe how to calculate $\mathbf{A}_1$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ by using double normalization in an iterative manner.

Let $\mathbf{M}_k$ be a (N x N) matrix, and $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ be diagonal (N x N) matrices, at the k^{th} iteration. Let $\mathbf{D}_R$ and $\mathbf{D}_C$ also be diagonal (N x N) matrices. At the beginning, we take $\mathbf{M}_0 = \mathbf{A}_0$, $\boldsymbol{\alpha}_0 = \mathbf{I}$ and $\boldsymbol{\beta}_0 = \mathbf{I}$, where **I** is the identity matrix. In the k^{th} iteration, we first normalize all rows and generate $\mathbf{D}_R$. Then we normalize all columns and generate $\mathbf{D}_C$. $\mathbf{M}_{k+1}$ also gets generated during these normalizations. Then we update $\boldsymbol{\alpha}_k$ and $\boldsymbol{\beta}_k$ using $\mathbf{D}_R$ and $\mathbf{D}_C$ to get $\boldsymbol{\alpha}_{k+1}$ and $\boldsymbol{\beta}_{k+1}$. A convergence criterion is used to terminate iterations.

The details of the calculations are given below:

Initial values: k=0, $\mathbf{M}_0 = \mathbf{A}_0$, $\boldsymbol{\alpha}_0 = \mathbf{I}$ and $\boldsymbol{\beta}_0 = \mathbf{I}$, $SS_{dev} = 1$, $SS_{dev,crit} = 0.01$

Do while $SS_{dev} > SS_{dev,crit}$

$SS_R(i) = \sum_{j=1}^{N} [\mathbf{M}_k(i,j)]^2 \qquad for\ i = 1\ to\ N$

$\mathbf{D}_R(i,j) = 1/[SS_R(i)]^{0.5} \qquad$ for i = j

= 0 otherwise

$\mathbf{M}_{temp} = \mathbf{D}_R \mathbf{M}_k$

$SS_C(j) = \sum_{i=1}^{N} \left[\mathbf{M}_{temp}(i,j)\right]^2 \qquad for\ j\ =\ 1\ to\ N$

$\mathbf{D}_C(i,j) = 1/[SS_C(j)]^{0.5}$ for i = j
= 0 otherwise

$\mathbf{M}_{k+1} = \mathbf{M}_{temp} \mathbf{D}_C$

$\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k \mathbf{D}_R$

$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k \mathbf{D}_C$

$SS_R(i) = \sum_{j=1}^{N} [\mathbf{M}_{k+1}(i,j)]^2 \qquad for\ i\ =\ 1\ to\ N$

$SS_{dev} = \{ \sum_{i=1}^{N} abs[SS_R(i) - 1] \} / N$

k = k+1

Loop

Final values: $\mathbf{A}_1 = \mathbf{M}_{k+1}$, $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{k+1}$, $\boldsymbol{\beta} = \boldsymbol{\beta}_{k+1}$
