Solution of Linear Systems Using Randomized Rounding

Kenneth L. Clarkson
Bell Laboratories
Murray Hill, New Jersey 07974
e-mail: `clarkson@research.bell-labs.com`

March 18, 2003

Abstract

This paper gives an algorithm for solving linear systems, using a randomized version of incomplete LU factorization together with iterative improvement. The factorization uses Gaussian elimination with partial pivoting, and preserves sparsity during factorization by randomized rounding of the entries. The resulting approximate factorization is then applied to estimate the solution. This simple technique, combined with iterative improvement, is demonstrated to be effective for a range of linear systems. When applied to medium-sized sample matrices for PDEs, the algorithm is qualitatively like multigrid: the work per iteration is typically linear in the order of the matrix, and the number of iterations to achieve a small residual is typically on the order of fifteen to twenty. The technique is also tested for a sample of asymmetric matrices from the *Matrix Market*, and is found to have similar behavior for many of them.

1 Introduction

This paper gives an algorithm for solving the linear system $Ax = b$, where A is an $n \times n$ real matrix, and b is a real n-vector. The algorithm does not use any special information or structure in the matrix; this includes even structural symmetry.

The algorithm uses a form of incomplete LU factorization: it approximates Gaussian elimination with partial pivoting, and does not retain all nonzero entries of the full factorization. This approximation involves *randomized rounding*: as A is factored, an entry a_{ij} of A is rounded using a random variable, and set to a nonzero value with a probability proportional to its magnitude $|a_{ij}|$. Because of the rounding, the number of entries per row is bounded by a selected value K. Using this approximate factorization, the solution is estimated by backsolving.

This scheme is incorporated into iterative improvement, solving for a correction by using the residual as the new b-vector. At each iteration, a new factorization is computed: this factorization will be distinct for each iteration, because of the random choices made in the algorithm. If a new factorization is not done at each iteration, the algorithm fails: iterative improvement using a fixed instance of one of these factorizations yields little change in the residual after a few iterations. New factorizations are therefore necessary and sufficient for iterative improvement to rapidly yield a solution. This property seems to be a distinctive characteristic of the randomized method, which otherwise seems similar in flavor to an algebraic multigrid method[Wag99], or to the multi-level graph methods of Bank and Smith[BS02, BS99].

The next section gives the algorithm; this is followed by experimental results, and some concluding remarks.

2 The Algorithm

The next subsection gives the basic algorithm, and following subsections give some refinements.

2.1 The ILU-$R(K)$ algorithm

The idea of the factorization method is to perform elimination steps row by row as usual, but to preserve sparsity by rounding the entries of a resulting row using randomization. This is done in such a way that the expected value of an entry of the rounded row is equal to the original value of the entry, but the total number of non-zero entries is bounded.

Specifically, the rounding procedure is as follows. Gaussian elimination is being done, to create an upper triangular matrix. To round entries a_{ik} of row a_i to entries a'_{ik}, the estimates a'_{ik} are all initialized to zero. Let s_i denote $\sum_{1\le m\le n} |a_{im}|$, the ℓ_1 norm of a_i. For K trials, an integer k is chosen with probability $|a_{ik}|/s_i$, and a'_{ik} is incremented by $\operatorname{sign}(a_{ik})s_i/K$.

It is easy to show that the expected value $Ea'_{ik} = a_{ik}$.

Running time. With the use of hashing, the updating of entries can be done in expected (and observed) $O(1)$ time.

In the K or K_2 trials for rounding, the indices k are chosen by first making an array of partial sums

$$c_j \equiv \sum_{k\le j} |a_{ik}|/s_i,$$

for $j = 1 \dots m$. For each trial, we generate a $U[0,1]$ random value v and find the smallest k such that $v \le c_k$. This generates the index k with probability proportional to $|a_{ik}|/s_i$, as desired. The worst-case time for each trial could be accelerated by using binary search on the partial sums, but this was not done, as the time for the step is relatively small anyway.

2.2 The *ILU-R*(K, K_2) algorithm

For some problems, a more complicated procedure may be useful: here the elimination step itself uses randomized rounding, allowing K_2 additional nonzeros.

The elimination step, for using row a_i to reduce the a_{ji} entry of a_j to zero, is as follows. Perform the following K_2 times: pick an integer k in the range $i+1\dots n$, where k is chosen with probability

$$|a_{ik}|/s_i,$$

and decrement a_{jk} by $s_i \operatorname{sign}(a_{ik})a_{ji}/a_{ii}K_2$. This yields the appropriate expectation, for any given such operation.

Note that *ILU-R*$(n, 1)$ preserves the sparsity of each row. Some linear systems can be solved by our method with $K_2 = 4$, sometimes less. However, in general, choosing $K_2 = K/2$ or $K_2 = K$ seems most effective, and the experiments below were done under the latter condition.

2.3 A refinement in rounding

Suppose row a_i is being rounded, and some entry is much larger in absolute value than s_i/K; such an entry is likely to yield at least a nonzero coefficient in the randomized-rounded version of row a_i. For such an entry, it would not be wasteful of space to simply include it in the rounded version of the row. (Here the assumption is that a rounded entry is represented using in the same way as an unrounded entry, for example using a double. This is somewhat wasteful of space, and one could alternatively represent the rounded entries in a more compact way, but such a scheme was not persued here.)

Having set aside such an entry, there remain $K-1$ rounded entries to choose, in a row with a smaller sum of absolute values s'_i. But now, what if some entry is larger than $s'_i/(K-1)$? It is again likely to yield a coefficient in the rounded version of row i.

These considerations suggest the following procedure: sort the list of m entries in the row in decreasing order of absolute value; let r_g be the g'th entry of this ordered list, for $g = 1\dots m$. Walk down the list, until an entry $r_{\hat{g}}$ is found such that

$$(K-\hat{g})|r_{\hat{g}}| \le \sum_{\hat{g}\le \ell \le m} |r_\ell|. \tag{1}$$

If no such entry exists, set $\hat{g}$ to $m+1$. Apply the randomized rounding procedure only to the entries in the list from $\hat{g}$ up to m.

Note that if $m \le K$, then this procedure finds $\hat{g} = m+1$, and so does nothing to the row. On the other hand, if there are more than K entries, and the entries are all equal, then randomized rounding is applied to all entries.

It is tempting to consider the deterministic procedure of simply "rounding" by using the largest K entries (in absolute value). Such a deterministic scheme yields only one approximate *LU* factorization, however, and not a family of them. The latter property is helpful for the algorithm given here.

It isn't actually necessary to sort the list of entries in the row: a procedure akin to Hoare's QUICKMEDIAN can be done. Briefly, we pick an entry r at random in the list, and partition the list with those entries larger (in magnitude) than r to the left, and those smaller than r to the right. Next we check if criterion 1 is satisfied by r. If so, we do a similar procedure for the elements to the left of r, and otherwise we look to the right of r. In the former case, we retain the sum $\sum_{|r'|\leq|r|}|r'|$ for use in checking entries in the recursion. This procedure succeeds because if r_g does not satisfy the criterion (1), then neither does any $g' \leq g$, as can be shown by induction.

The expected time for this procedure satisfies the same recurrence as for QUICKMEDIAN, and hence is $O(m)$.

2.4 Choosing K and K_2

The running time of the algorithm depends critically on the rounding parameters K and K_2; as will be shown experimentally below, if K or K_2 are too small, then no improvement is obtained, but if K and K_2 are sufficiently large, then the residual is reduced in ℓ_2 norm at each step by a factor comfortably larger than 1, such as 2. That is, there are thresholds for K and K_2, and when these thresholds are exceeded, convergence is relatively rapid. These thresholds depend on A and b; they seem to depend in general only weakly on n, however, for a family of matrices arising from the same problem.

Due to the critical dependence on the values of K and K_2, the algorithm attempts to choose them so as to minimize the overall running time. In the experiments below, the value of K_2 is initially set to half the value of K, unless several successive increases in K have failed to improve the performance of the iterations; thereafter K_2 is maintained to be equal to K.

No rounding is done until the total number of entries in the factorization is at least 1.5 times the number of entries in the original matrix, and the total number of entries is also at least Kn: that is, K and K_2 are effectively n until the factors are sufficiently dense.

The algorithm tries to find an effective value for K, as follows: the history of the iterations is maintained, and for each iteration, the time taken for the iteration, the value of K, and the improvement are recorded. (Here the improvement is the reduction in the ℓ_2 norm of the residual.) One the K values that were used maximizes the ratio of improvement to iteration time. Such a choice would minimize the overall time of the algorithm. If that best K is not adequate, so that the improvement and the improvement/time ratio are each below some pre-set thresholds, then a new value of K is tried, equal to approximately $\sqrt{2}$ times the maximum K tried so far.

Thus, the algorithm searches for the best value K; as seen below, for some matrices, that search can simply yield $K = n$, and the algorithm reduces to full factorization.

2.5 Running time

The performance of the algorithm is a combination of several factors: the number of iterations needed to obtain a sufficiently small residual, the values K and K_2 needed for such iterations, and the dependence of the running time on n, K and K_2.

The observed running time for each iteration was proportional to KK_2n. It is possible to concoct a matrix and elimination order for which the rows all have two nonzero entries, and yet the time needed is $\Omega(n^2)$. On the other hand, if the elimination order is random, and the rounding procedures are done with the binary search procedure discussed in §2.2, and hashing is used to update coefficient values, then the expected running time per iteration is $O(nKK_2 \log K)$, or linear in n. We did not implement the random elimination order, however.

3 Experimental Results

Three sets of matrices were considered:

BS1 Families of matrices based on a discretization of the Laplace equation, on a regular grid;

BS2 Families based on discretizations of PDEs, on irregular grids, constructed adaptively;

MM A collection of matrices from the "Matrix Market"[Mat], choosing square matrices with real entries, and favoring asymmetric matrices, especially structurally asymmetric matrices (with an asymmetric pattern of non-zero entries).

The first two families were used by Bank and Smith in evaluating their *ILU-MG* multigraph algorithms.[BS99] (The `bs2` matrices were supplied to the author by Kent Smith.) We refer the reader to their article for a detailed description of these matrices. The `bs1` matrices also include a family of matrices related to Ising models.

In the experiments, the stopping criterion was that the residual Euclidean norm be more than 10^{-6} times the norm of b, the right hand side. The right hand side was typically chosen by making setting one entry, chosen at random, to one, with the remaining entries zero. The columns of the matrices were randomly shuffled on input, and partial pivoting was used.

Some basic experimental results for these families are tabulated below in Figures 1, 5, and 8. In these tables,

- "n" is the order of the matrix;
- "# its" is the number of iterations performed;
- $\hat{E}$ is the average number of entries per row, for the value $\hat{K}$ of K at which the largest proportion of the total gain was achieved. When K and K_2

were found to be so large that the procedure is the same as Gaussian elimination with partial pivoting, and this procedure failed, this condition is indicated by a table entry of "$**$".

- "gain" is the average gain of the iterations for which the value of K used was $\hat{K}$. Here *gain* is the logarithm (base 2) of the reduction by the iteration of the Euclidean norm of the residual. The table entry "*" indicates that the procedure failed, usually by running of out storage.

- "gain/t" is the average gain per unit CPU time of the iterations using $K = \hat{K}$, multiplied by n. As for gain, the entry "*" indicates failure.

- "E, GE" in Figure 8 indicates the average number of entries per row when Gaussian elimination using partial pivoting was applied to the matrix. Failure of the procedure is indicated by a table entry of "$**$".

- "speedup" in Figure 8 indicates the ratio of the total running time of full Gaussian elimination to that of the new algorithm. A blank entry means that both algorithms failed; an entry of "*" means that the full Gaussian elimination failed, and the new algorithm did not.

3.1 Results for the BS1 matrices

Results for the BS1 matrices are shown in Figure 1.

The matrices are loosely based on a discretization of the Laplace equation for a regular grid in two dimensions: the matrices `A-X-0.dat` have order $n = \mathtt{X}^2$, are symmetric, have $a_{ii} = 4$ for all i, and have above-diagonal non-zero entries $a_{i,i+1} = -1$ for $i = 1 \dots n-1$ and $i \neq 0 \bmod \mathtt{X}$. Also, $a_{i,i+\mathtt{X}} = -1$ for $i = 1 \dots N - \mathtt{X}$. The matrices `A-X-1.dat` have the same non-zero entries, except that they are positive.

The matrices `A-X-2.dat` have the same non-zero entries, except that the non-zero non-diagonal entries (the "1"s) are given their signs at random, equally likely to be $+1$ or -1; this is done symmetrically. Such matrices are related to Ising models. Finally, the `A-X-3.dat` matrices have similarly random entries, but asymmetric: the random choices for the off-diagonal entries are done independently.

The value of $\hat{E}$ does increase somewhat with n, and especially for the `A-X-1.dat` family, but this dependence seems relatively mild, especially for the `A-X-2.dat` family.

Figure 2 shows the behavior of the algorithm over its improvement iterations, for matrix `A-80-0`. The solid line is the gain, and the dashed line is the entries per row, E.

A somewhat-worse result shown in Figure 3 for matrix `A-80-1`.

In Figure 4, that gain and gain-per-time are shown as functions of K, with the gain again shown as a solid line, while the gain per unit time is shown as a dashed line. The low gain for $K = 20$ is apparently transient, or due to a random effect.

matrix	n	# its	$\hat{E}$	gain	gain/time
A-10-0	100	17	6.5	1.2	122.3
A-20-0	400	19	8.4	1.2	446.5
A-40-0	1600	32	7.8	0.6	678
A-80-0	6400	19	14	1.4	444.6
A-160-0	25600	21	14.1	1.1	76.7
A-320-0	102400	4	10.6	*	*
A-10-1	100	15	7.6	1.7	174.6
A-20-1	400	18	11.2	1.4	528.9
A-40-1	1600	23	19.8	1.4	688.3
A-80-1	6400	14	39.7	3.7	357.2
A-160-1	25600	13	39.9	2.6	50.3
A-320-1	102400	3	14	*	*
A-10-2	100	12	6.6	1.8	179
A-20-2	400	18	8	1.9	721.9
A-40-2	1600	16	7.8	1.3	1418.1
A-80-2	6400	17	7.9	1.2	559.4
A-160-2	25600	13	8	1.6	175.7
A-320-2	102400	11	7.4	1.9	50.7
A-10-3	100	14	15.3	4.5	453.4
A-20-3	400	11	7.2	1.9	695.4
A-40-3	1600	7	7.8	3.1	3566.1
A-80-3	6400	7	7.9	3	1312.2
A-160-3	25600	8	8	2.6	271.6
A-320-3	102400	8	8	2.7	67.6

Figure 1: Basic results for data set BS1.

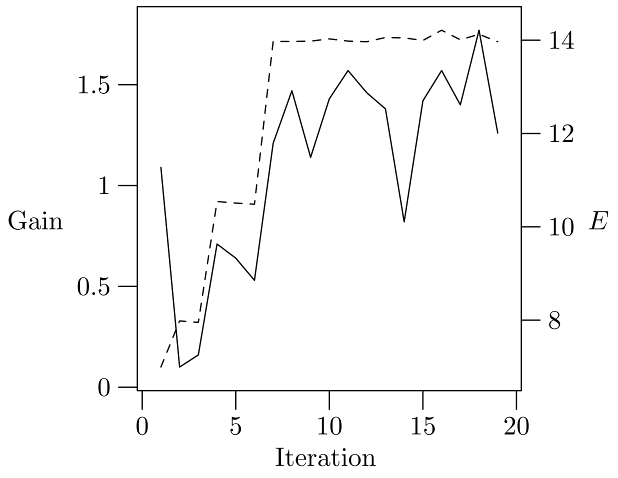

Figure 2: A-80-0, order 6400

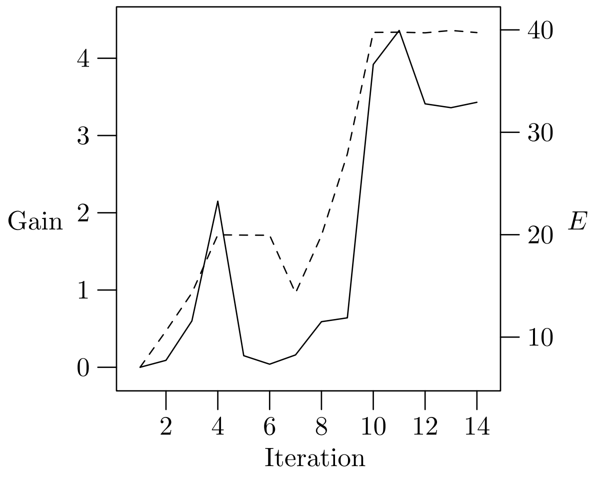

Figure 3: A-80-1, order 640

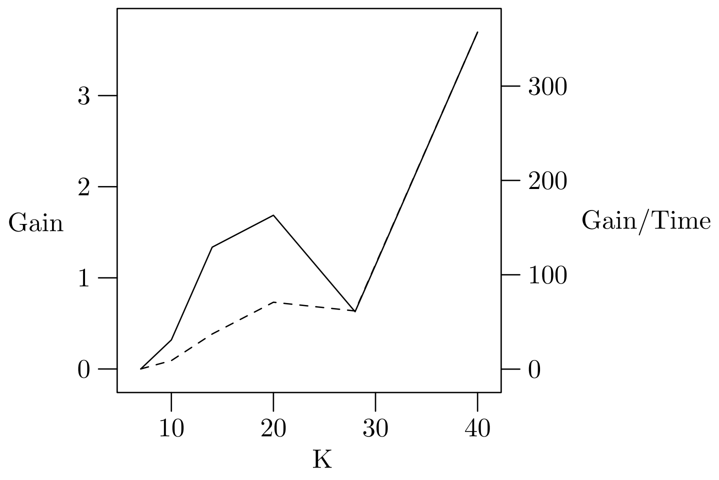

Figure 4: A-80-1

3.2 The BS2 matrices

The matrices were all derived from the discretization of partial differential equations on nonuniform adaptive grids, and have orders $n = 5000$, 20000 and 80000 for each problem domain.

3.2.1 Problem domains

`Superior.` Here the problem is the Poisson equation $\Delta u = -1$, on a domain shaped like Lake Superior.

`Hole.` A more complicated system with discontinuous, anisotropic coefficients.

`Texas.` The indefinite Helmholtz equation $-\Delta u - 2u = 1$ on a domain shaped like the state of Texas.

`UCSD.` A constant-coefficient convection-diffusion equation

$$-\nabla \cdot (\nabla + \beta u) = 1,$$

with $\beta = (0, 10^5)^T$, on a domain in the shape of the UCSD logo.

`jcn1-3 and jcn4-6.` Two variants of a current continuity equation for semiconductor device modeling. The equation has the form

$$-\nabla \cdot (\nabla + \beta u) = 0.$$

Solutions vary exponentially in magnitude across the domain. In the matrices `jcn1`, `jcn2`, and `jcn3`, the convective term is such that the device is *forward biased*, and in the remaining matrices, the sign of the convective term is reversed.

matrix	n	# its	$\hat{E}$	gain	gain/time
hole1	5000	12	96.8	4.8	56.7
hole2	20000	14	138	4.5	17.5
hole3	80000	5	35	*	*
jcn1	5000	28	10.7	0.6	121
jcn2	20000	29	10.9	0.6	27.8
jcn3	80000	24	35	1.5	8
jcn4	5000	19	19	1.2	173.3
jcn5	20000	24	25.9	1.1	36.1
jcn6	80000	18	35	1.5	7.6
sup1	5000	11	22.8	2.7	195.1
sup2	20000	11	46.9	3.9	53.6
sup3	80000	5	35	*	*
tex1	5000	14	96.2	3.2	37.9
tex2	20000	13	138.2	4.5	17.6
tex3	80000	5	35	*	*
ucsd1	5000	10	7.8	2.2	431.4
ucsd2	20000	12	8.8	1.7	79.3
ucsd3	80000	11	9.3	1.9	18.1

Figure 5: Basic results for data set BS2.

3.2.2 Results

Some basic experimental results are shown in Figure 5. Unfortunately, storage was insufficient to solve some of the larger systems, but again, a relatively mild dependence of $\hat{E}$ on n is seen.

Figures 6 and 7 show a threshold for the gain as a function of K. (Again, the axis labels on the left in these figures is for the solid line, while the axis labels on the right are for the dashed line.) The `Texas` family of matrices had the highest such threshold in this set, just as this family was the hardest for Bank and Smith's ILU-MG algorithm to solve.[BS99]

3.3 The MM matrices

As mentioned above, these matrices were taken from the "Matrix Market"[Mat], which includes the Harwell-Boeing set, and a representative matrix was chosen from nearly every set that included real square matrices of large bandwidth.

3.3.1 Results

Some basic experimental results are shown in Figure 8. In general, these matrices are smaller than would be ideal for the experiments here. Some matrices, such as `fidapm07`, `fs_760_3`, and `mbeacxc`, would appear to be singular. Other

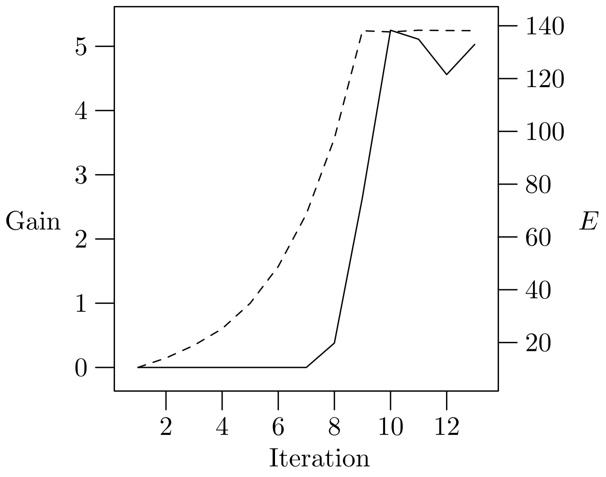

Figure 6: tex2, order 20000

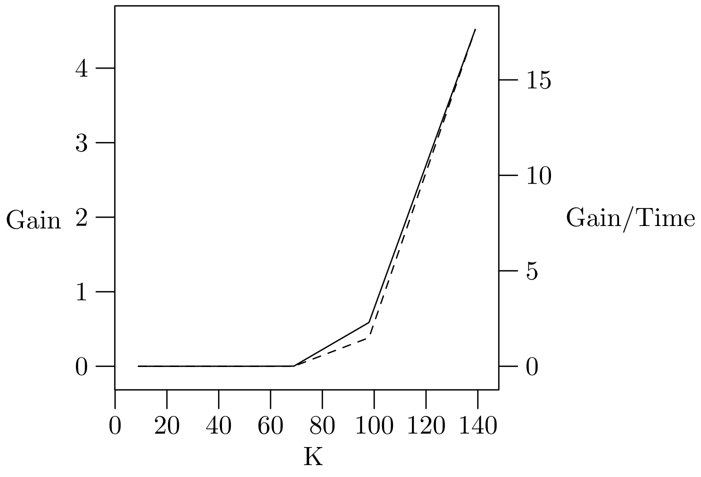

Figure 7: tex2

matrix	n	# its	$\hat{E}$	gain	gain/time	E, full GE	speedup
1138_bus	1138	19	37.2	20.3	5771.7	29	0.02
494_bus	494	8	10.7	30.8	15209.5	10	0.01
add20	2395	6	8.2	3.6	2900	392	5.75
add32	4960	6	5.5	3.7	2122.6	18	0.17
bfw782a	782	23	37.9	2.4	729.9	237	0.5
bp__1400	822	5	25.6	35.1	14446.4	31	0.25
bp__1600	822	1	7.8	Inf	Inf	23	1
bwm2000	2000	15	11	3.8	3808.5	17	0.07
ck656	656	14	10.5	2	1052.6	20	0
conf5_0_...	3072	7	422.2	*	*	*	*
dwb512	512	2	7.7	13.3	6829.6	96	1
fidapm07	2065	14	**	*	*	1008	0.12
fs_760_3	760	14	**	*	*	**	
gemat11	4929	31	139	0.9	16.5	*	*
gre_216b	216	26	**	*	*	**	
hor__131	434	12	70.1	3.1	390.6	173	0.24
impcol_e	225	7	7.2	2.9	594.3	12	0.01
jpwh_991	991	20	8.9	1	788	232	4.55
lns_3937	3937	12	372.2	*	*	*	*
lop163	163	10	42.4	21.5	3505.8	43	0.01
mahindas	1258	1	8	58.2	36616.2	45	2.98
mbeacxc	496	6	**	*	*	**	
mcca	180	8	22.1	2.8	479	49	0.01
memplus	17758	10	10	2.4	235.1	*	*
orani678	2529	17	53.2	1.3	95.2	158	0.08
orsirr_1	1030	19	10.9	1.1	942.8	348	2.66
orsreg_1	2205	16	11	1.3	1225.6	573	3.17
plat1919	1919	14	**	*	*	**	
psmigr_3	3140	6	281.2	3.4	21.9	*	*
qh882	882	14	46.9	2.6	783.4	39	0.05
rbs480b	480	7	198.7	39.5	862.1	202	0.16
rw5151	5151	13	15.7	*	*	*	*
s2rmt3m1	5489	5	62.4	*	*	*	*
saylr4	3564	18	93.3	2.9	249.9	*	*
sherman5	3312	17	22.5	2	567.6	270	0.92
shl____0	663	1	1	Inf	Inf	1	1
steam1	240	6	15.7	3.5	829.9	129	101
steam2	600	2	13.9	15.9	9518.8	92	1
tols4000	4000	1	0.6	59.9	47929.2	1	0.75
west0989	989	10	15.9	5.1	4208.4	13	0

Figure 8: Basic results for MM data set.

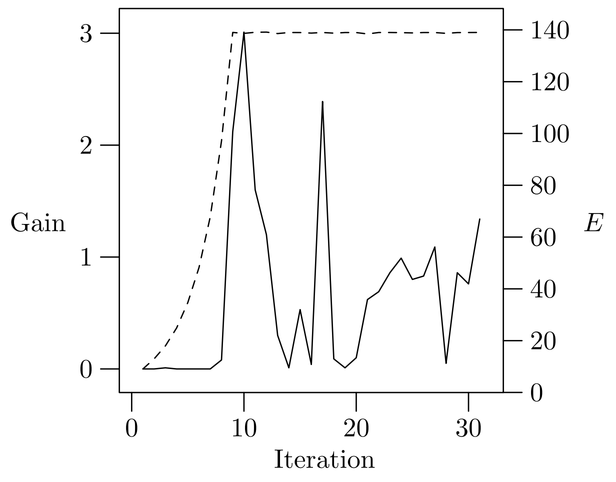

Figure 9: gemat11, order 4929

matrices, such as `shl___0` and `tols4000`, have such small fill-in that there is no real difference between the new algorithm and Gaussian elimination.

There are also some matrices, such as `1138_bus`, for which the new algorithm actually seems to be *worse* with respect to fill-in; presumably this is due to random variation, or to non-zero entries that later become zero again in full Gaussian elimination.

However, for most of the matrices tested, the new algorithm at least does no harm with respect to $\hat{E}$, and for several matrices, such as `gemat11`, `memplus`, and `rw5151`, succeeds in solving the systems where Gaussian elimination runs out of space. Moreover, the three matrices just mentioned are from diverse applications, none of which are partial differential equations.

While the main question considered here is the average number of entries needed to allow the randomized scheme to make progress, there is also the question of the overall time taken: for several matrices, especially very small ones, the "speedup" in overall time is actually less than one, even much less than one. This unfortunate behavior can be much reduced or eliminated simply by having a minimum threshold on the total number of entries in the LU factorization, say 50000, before rounding. The result is to reduce the new procedure to Gaussian elimination for many of the given matrices. However, this was not done here, so that effect of the randomized rounding on the gain could be seen.

Figures 9 and 10 show the performance of the algorithm for `gemat11`, again showing a threshold for the gain as a function of K, and Figures 11 and 12 show the performance for `memplus`.

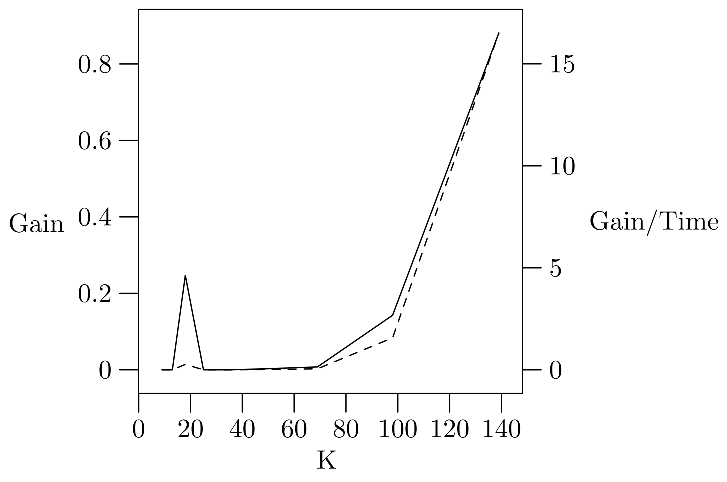

Figure 10: gemat11

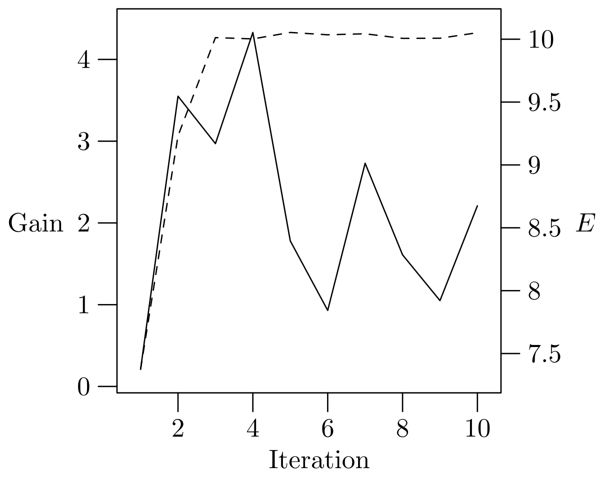

Figure 11: memplus, order 17758

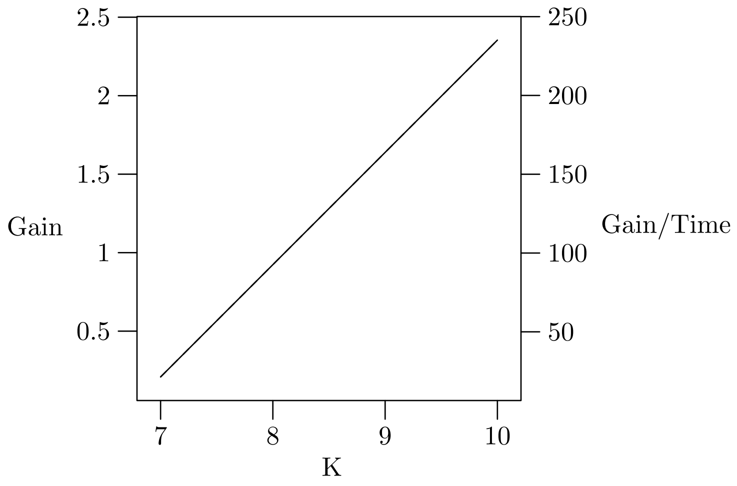

Figure 12: memplus

4 Concluding remarks

The algorithm described here is minimalist: it does not try to use a good elimination order, it has not been put within a larger iterative algorithm, it takes no advantage of symmetry, it does no acceleration using previous results, *etc.* All such options could be explored.

Other issues could be explored theoretically: why does the iterative improvement work when a new factorization is used at each step, but not otherwise? Again, any given factorization seems to be no worse as a pre-conditioner, but also no better than, other heuristics for ILU-factorization. Why is the algorithm effective for some matrices, but not others? What other matrix characteristics are related to this property?

The implementation used here was not fine-tuned for storage efficiency; in particular, as noted above, the entries in a rounded row are are all small integer multiples of a single floating point value, and no advantage of that fact was used. Such tuning, or combining this method with other techniques, should allow larger problems to be solved.

Note that the algorithm is amenable to parallelization, in two ways: each factorization can be done independently of the others (both L and U would need to be stored), and the enforced sparsity of the matrix implies that there is large set of independent vertices in the corresponding graph. In a parallel implementation of the factorization, elimination steps could be done for rows corresponding to that independent set. Next, rounding could be done in parallel for every row. Iterating such steps in the factorization removes a constant fraction of the rows per iteration, implying $O(\log n)$ iterations to complete a factorization.

Acknowledgements. I'm grateful to Lawrence Cowsar, Roland Freund, Peter Oswald, Kent Smith, and Margaret Wright for many helpful discussions, and again to Kent Smith for providing several example matrices.

References

[BS99] R. E. Bank and R. K. Smith. The incomplete factorization multigraph algorithm. *Siam J. Sci. Comp.*, 20(4):1349–1364, 1999.

[BS02] R. E. Bank and R. K. Smith. An algebraic multilevel multigraph algorithm. *Siam J. Sci. Comp.*, 23(5):1572–1592, 2002.

[Mat] The matrix market. `http://math.nist.gov/MatrixMarket`.

[Wag99] C. Wagner. Introduction to algebraic multigrid. Technical report, Universität Heidelberg, 1999.