Taking advantage of Sherman's march*

Pablo Guerrero-García[1] and Matías Toril-Genovés[2]

[1]Dept. Applied Mathematics, Univ. Málaga, pablito@ctima.uma.es**
[2]Dept. Communications Engineering, Univ. Málaga, mtoril@ic.uma.es

Abstract. During the simulation of a mobile telecommunications system, a sequence of systems of linear equations must be solved. In this sequence, the coefficient matrix of the $(k+1)$th system is of order one greater than that of the kth, and the former is constructed by enlarging the latter with a new column and a new row. All matrices involved are strictly diagonally dominant, but the condition number suffers a heavy worsening as k increases. In this lecture we show that taking advantage of this diagonal dominance property is crucial to be able to obtain as much as a 30% improvement on average in the CPU time to complete the whole process in MATLAB v7.5.

Key words: LU decomposition, updating, leading principal submatrix, Sherman's march, strictly diagonally dominant.

1 Four naive approaches using MATLAB

The problem at hand can be formulated as follows. Given a strictly diagonally dominant matrix $A \in \mathbb{R}^{n\times n}$, let us denote by $A_k \triangleq A(1{:}\,k, 1{:}\,k)$ its leading principal submatrix of order $k \in 1{:}\,n$. Given a right-hand-side vector $b \in \mathbb{R}^n$, let us denote by $\mathbf{b}_k$ the subvector with the k uppermost components. The aim is to solve the systems of linear equations $A_k\mathbf{x}_k = \mathbf{b}_k$ for each $k \in 1{:}\,n$.

The very first idea would be to ignore the sequential feature of these systems and hence solve them all by direct backslashing in MATLAB:

```
for k = 1:n
  x = A(1:k,1:k)\b(1:k);
end
```

Alternatively, we can also use a factorization: in MATLAB we have both the LU and the QR decomposition available, the latter being slower but also an interesting choice when we must deal with condition numbers of $\mathcal{O}(10^7)$, say. Therefore, pivoting for numerical stability seems to be *a priori* necessary in the LU decomposition,

* Technical Report MA-10/01 (http://www.matap.uma.es/investigacion/tr.html), Dept. Applied Mathematics, Univ. Málaga, 29th January 2010. Lecture to be given at 11th ACDCA Summer Academy conference of the Technology and its Integration into Mathematics Education (TIME 2010) international symposium to be held at Málaga (Spain) in July 2010.

** Corresponding author.

```
for k = 1:n
  [Lk,Uk,pk] = lu(A(1:k,1:k),'vector'); temp = b(1:k);
  x = Uk\(Lk\temp(pk));
end
```

However, updating this pivoted factorization is no longer efficient. On the other hand, the QR decomposition can either be recomputed

```
for k = 1:n
  [Qk,Rk] = qr(A(1:k,1:k));
  x = Rk\(b(1:k)'*Qk)';
end
```

or it can easily be updated by using `qrinsert`:

```
Qk = 1; Rk = A(1,1);
for k = 1:n-1
  [Qk,Rk] = qrinsert(Qk,Rk,k+1,A(1:k,k+1),'col');
  [Qk,Rk] = qrinsert(Qk,Rk,k+1,A(k+1,1:k+1),'row');
  x = Rk\(b(1:k)'*Qk)';
end
```

Note that both codes avoid transposing the orthogonal factor to economize in memory allocation and access. To sum up, we have four choices so far:

BS. Direct backslashing with A_k.
LU. Recompute the LU decomposition with pivoting $P_kA_k = L_kU_k$.
QR. Recompute the QR decomposition $A_k = Q_kR_k$.
UP. Update the QR decomposition $A_k = Q_kR_k$ knowing $A_{k-1} = Q_{k-1}R_{k-1}$.

2 Updating the LU decomposition without pivoting

Let us show how a fifth approach can be obtained. Assume that the LU decomposition $A_k = L_kU_k$ is known. The LU decomposition of A_{k+1} can be computed by noting that:

$$A_{k+1} \triangleq \begin{bmatrix} A_k & A(1\colon k, k+1) \\ A(k+1, 1\colon k) & A(k+1, k+1) \end{bmatrix} = \begin{bmatrix} L_k & \mathrm{O} \\ \mathbf{v}_k^T & 1 \end{bmatrix} \begin{bmatrix} U_k & \mathbf{u}_k \\ \mathrm{O}^T & d_k \end{bmatrix}.$$

Equating blocks we get:

$$L_k\mathbf{u}_k = A(1\colon k, k+1), \quad \mathbf{v}_k^T U_k = A(k+1, 1\colon k), \quad \mathbf{v}_k^T\mathbf{u}_k + d_k = A(k+1, k+1),$$

and hence, using superindex $-T$ to denote the inverse of the transpose,

$$\mathbf{u}_k = L_k^{-1}A(1\colon k, k+1), \quad \mathbf{v}_k = U_k^{-T}A(k+1, 1\colon k)^T, \quad d_k = A(k+1, k+1) - \mathbf{v}_k^T\mathbf{u}_k.$$

This procedure, which was named *Sherman's march* by Stewart [4, p. 169] in reference to a procession proceeding to the southeast from Tennessee to Georgia, can be implemented efficiently in MATLAB as follows:

```
Lk = 1; Uk = A(1,1);
for k = 1:n-1
  uk = Lk\A(1:k,k+1); vkt = A(k+1,1:k)/Uk; dk = A(k+1,k+1)-vkt*uk;
  Lk = [Lk zeros(k,1); vkt 1]; Uk = [Uk uk; zeros(1,k) dk];
end
```

Note that our code avoids transposing the upper triangular factor to economize in memory allocation and access. The main drawback of this procedure is that it does not allow pivoting to be taken into account (and maybe this is the reason why it is not even described in most numerical analysis textbooks), but notice that matrix A in our case is strictly diagonally dominant by rows and hence it is well-known [2, p. 399] that Gaussian elimination can be performed without pivoting in a numerically stable way (i.e., no growth of round-off errors) for this class of matrices!

In other words, taking advantage of the diagonal dominance property has been crucial to obtain this fifth approach:

SH. Update the LU decomposition without pivoting $A_k = L_k U_k$ knowing $A_{k-1} = L_{k-1}U_{k-1}$.

However, the code seems to be quite complex when compared with that of recomputed LU or that of direct backslashing. Now a natural question arises: does it lead to a significant reduction in the CPU time, while maintaining a good numerical quality of the obtained approximate solutions?

3 Preliminary numerical experiments

To compare the five approaches developed in the previous sections, we must check both the CPU time needed to complete the whole process and the numerical quality of the approximate solutions $\hat{\mathbf{x}}_k$ obtained for each k. The latter can be accomplished with a measurement of the backward error, as the rule of thumb "forward error $\lesssim$ backward error $\times$ condition number" indicates: the forward error measures the (absolute or relative) distance between exact and computed solutions, the backward error measures the stability of the *method*, and the condition number measures the stability of the *problem*. This is the reason why a suitable scale-independent measurement is Wilkinson's relative residual [3, p. 12], which is a normwise relative backward error defined as

$$\frac{\|\mathbf{b}_k - A_k\hat{\mathbf{x}}_k\|_2}{\|A_k\|_2\,\|\hat{\mathbf{x}}_k\|_2}.$$

As it is well-known, the lesser its backward error, the better an approximate solution computed by a given method is. We have used the Frobenius norm $\|A_k\|_F \triangleq \sqrt{\sum_{i,j\in 1:k} |a_{ij}|^2}$ rather than the 2-norm $\|A_k\|_2 \triangleq \sqrt{\max_{i\in 1:k} |\lambda_i(A_k^T A_k)|}$ for our experiments not to take a long time.

The comparison was done in MATLAB v7.5 with an Intel Pentium 4, 2.8 Ghz, 768 Mb RAM under Windows XP. The right-hand-side vector $b \in \mathbb{R}^n$ has been

randomly generated from a standard normal distribution. We have considered two real-world matrices (taken from the simulator) with $n = 1020$ and $n = 1200$, and both have been checked to be strictly diagonally dominant by rows. These values of n seem to be quite representative of the real situations that our solvers are going to deal with. Instead of starting from A_1, we have started from A_{k_0} with $k_0 = 21$ in the former case and $k_0 = 201$ in the latter case, in order to perform 999 iterations for both examples. The left side of figure 1 shows the relative residuals for the $n = 1020$ example, whereas the right side shows those for the $n = 1200$ example; lower graphs in this figure limit the greatest relative residuals to $5 \cdot 10^{-19}$ and discard the first 100 iterations to show a cleaner zoom-in window for both examples.

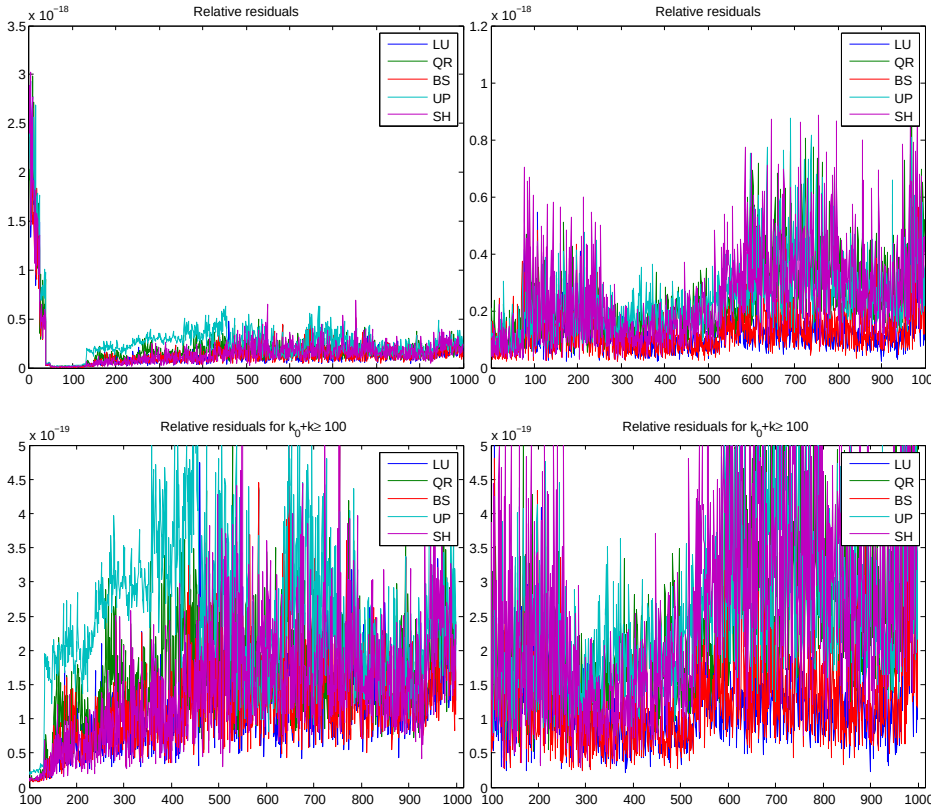


Fig. 1. Relative residuals

The backward residuals obtained in all situations were pretty small, always less than $3 \cdot 10^{-18}$ and with no significant differences among the five approaches. Hence we can conclude that the five approaches allow us to obtain approximate solutions that can be regarded as very satisfactory from a numerical analysis

point of view in spite of the progressive ill-conditioning (measured with the condition number $\|A_k\|_F\|A_k^{-1}\|_F$ using the Frobenius norm to be consistent with the relative residuals) of the leading principal submatrices displayed in the upper graphs of figure 2. As above, the left side corresponds to the $n = 1020$ example and the right side to the $n = 1200$ example.

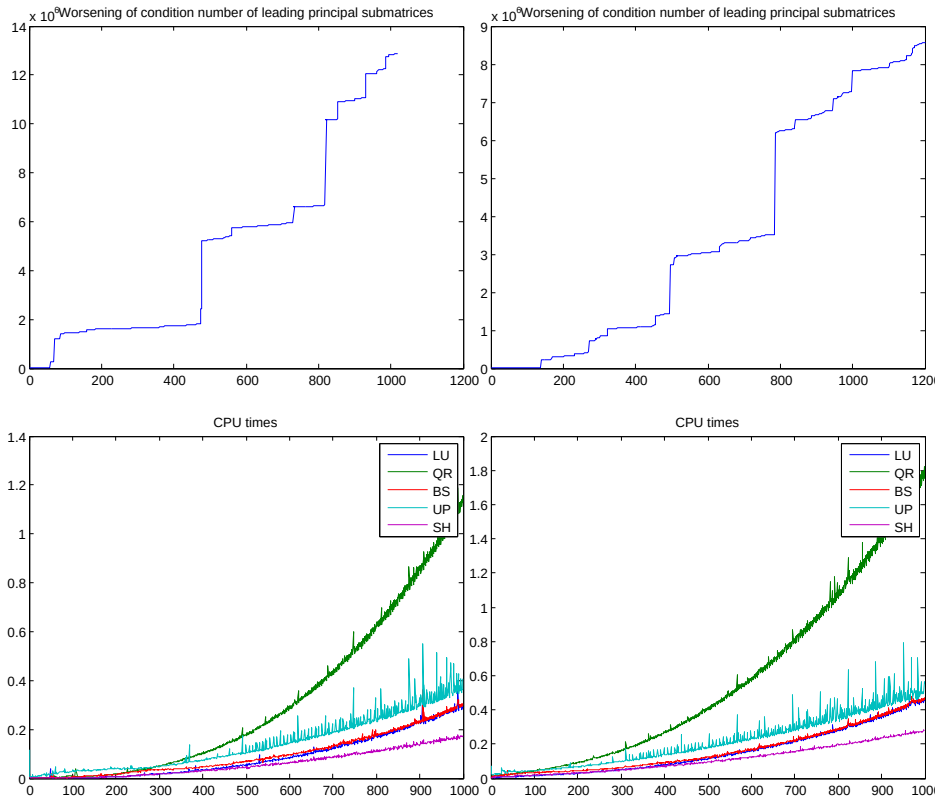


Fig. 2. Condition numbers and CPU times

Regarding the CPU times (shown in the lower graphs of figure 2), it is clear that **QR** is the worst approach and **SH** is the best one as k increases. Although **BS** and **LU** are slightly better than **UP**, these three approaches seem to get asymptotically closer as k increases (this effect is cleaner in the $n = 1200$ example). Total CPU times for the whole processes in the $n = 1020$ example, for which **SH** needed 58.71 seconds, were as follows:

- **LU** needed 86.47 seconds (x1.5 slower than **SH**, could be improved a 32%)
- **QR** needed 310.28 seconds (x5.3 slower than **SH**, could be improved a 81%)
- **BS** needed 95.69 seconds (x1.6 slower than **SH**, could be improved a 39%)

– **UP** needed 139.78 seconds (x2.4 slower than **SH**, could be improved a 58%)

and those in the $n = 1200$ example, for which **SH** needed 111.02 seconds, were as follows:

– **LU** needed 153.47 seconds (x1.4 slower than **SH**, could be improved a 28%)
– **QR** needed 571.10 seconds (x5.1 slower than **SH**, could be improved a 81%)
– **BS** needed 163.44 seconds (x1.5 slower than **SH**, could be improved a 32%)
– **UP** needed 216.35 seconds (x1.9 slower than **SH**, could be improved a 49%)

4 Conclusion

Professor Acton's celebrated history (dated sixty years ago) on being asked to invert a matrix that turned out to be an orthogonal one [1, p. 246] teach us to look carefully to the properties of the matrices being involved in every engineering problem. We have been faced with another instance of this cautionary tale in which not taking advantage of the diagonal dominance property of the matrices involved would have prevented us to obtain as much as a 30% improvement on average in the time required to complete the whole process!

Acknowledgements

This work has been supported by the Spanish Ministry of Science and Innovation (grant TEC2009-13413).

References

1. Forman S. Acton. *Numerical Methods That Work*. The Mathematical Association of America, Washington (DC, USA), 1990.
2. Richard L. Burden and J. Douglas Faires. *Numerical Analysis, 8th edition*. International Thomson Publishing, Belmont (CA, USA), 2005.
3. Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms, 2nd edition*. SIAM Publications, Philadelphia (PA, USA), 2002.
4. Gilbert W. Stewart. *Matrix Algorithms I: Basic Decompositions*. SIAM Publications, Philadelphia (PA, USA), 1998.

Taking advantage of Sherman's march⋆

11th ACDCA Summer Academy, TIME 2010 international symposium

Pablo Guerrero-García[1] and Matías Toril-Genovés[2]
{pablito@ctima,mtoril@ic}.uma.es
http://www.matap.uma.es/investigacion/tr/ma10_01.pdf

[1]Dept. Applied Mathematics, University of Málaga (Spain)
[2]Dept. Communications Engineering, University of Málaga (Spain)

Málaga, 7th July 2010

⋆*Work supported by Spanish Ministry of Science and Innovation (grant TEC2009-13413)*

Abstract:

During the simulation of a mobile telecommunications system, a sequence of systems of linear equations must be solved. In this sequence, the coefficient matrix of the $(k+1)$th system is of order one greater than that of the kth, and the former is constructed by enlarging the latter with a new column and a new row. All matrices involved are strictly diagonally dominant, but the condition number suffers a heavy worsening as k increases.

In this lecture we show that taking advantage of this diagonal dominance property is crucial to be able to obtain as much as a 30% improvement on average in CPU time to complete the whole process in MATLAB v7.5.

Key words: LU decomposition, updating, leading principal submatrix, Sherman's march, strictly diagonally dominant.

Outline: four naive approaches using MATLAB, updating the LU decomposition without pivoting, preliminary numerical experiments, conclusion.

1. Four naive approaches using MATLAB (1/3)

The problem at hand can be formulated as follows:

> Given a strictly diagonally dominant matrix $A \in \mathbb{R}^{n\times n}$, let us denote by $A_k \triangleq A(1\colon k, 1\colon k)$ its leading principal submatrix of order $k \in 1\colon n$.
>
> Given a right-hand-side vector $b \in \mathbb{R}^n$, let us denote by $\mathbf{b}_k$ the subvector with the k uppermost components.
>
> The aim is to solve the systems of linear equations $A_k\mathbf{x}_k = \mathbf{b}_k$ for each $k \in 1\colon n$.

The very first idea would be to ignore the sequential feature of these systems and hence solve them all by direct backslashing in MATLAB:

```
for k = 1:n
  x = A(1:k,1:k)\b(1:k);
end
```


1. Four naive approaches using MATLAB (2/3)

Alternatively, we can also use a factorization: in MATLAB we have both the LU and the QR decomposition available, the latter being slower but also an interesting choice when we must deal with condition numbers of $\mathcal{O}(10^7)$, say. Therefore, pivoting for numerical stability seems to be *a priori* necessary in the LU decomposition,

```
for k = 1:n
  [Lk,Uk,pk] = lu(A(1:k,1:k),'vector');
  temp = b(1:k);
  x = Uk\(Lk\temp(pk));
end
```

However, updating this pivoted factorization is no longer efficient. On the other hand, the QR decomposition can either be recomputed

```
for k = 1:n
  [Qk,Rk] = qr(A(1:k,1:k));
  x = Rk\(b(1:k)'*Qk)';
end
```

1. Four naive approaches using MATLAB (3/3)

...or it can easily be updated by using `qrinsert`:

```
Qk = 1; Rk = A(1,1);
for k = 1:n-1
  [Qk,Rk] = qrinsert(Qk,Rk,k+1,A(1:k,k+1),'col');
  [Qk,Rk] = qrinsert(Qk,Rk,k+1,A(k+1,1:k+1),'row');
  x = Rk\(b(1:k)'*Qk)';
end
```

Note that both codes avoid transposing the orthogonal factor to economize in memory allocation and access. To sum up, we have four choices so far:

BS. Direct backslashing with A_k

LU. Recompute the LU decomposition with pivoting $P_kA_k = L_kU_k$

QR. Recompute the QR decomposition $A_k = Q_kR_k$

UP. Update QR decomposition $A_k = Q_kR_k$ knowing $A_{k-1} = Q_{k-1}R_{k-1}$

2. Updating the LU decomposition without pivoting (1/3)

Let us show how a fifth approach can be obtained. Assume that the LU decomposition $A_k = L_kU_k$ is known. The LU decomposition of A_{k+1} can be computed by noting that:

$$A_{k+1} \triangleq \begin{bmatrix} A_k & A(1\colon k, k+1) \\ A(k+1, 1\colon k) & A(k+1, k+1) \end{bmatrix} = \begin{bmatrix} L_k & \mathrm{O} \\ \mathbf{v}_k^T & 1 \end{bmatrix} \begin{bmatrix} U_k & \mathbf{u}_k \\ \mathrm{O}^T & d_k \end{bmatrix}.$$

Equating blocks we get:

$$L_k\mathbf{u}_k = A(1\colon k, k+1),$$

$$\mathbf{v}_k^T U_k = A(k+1, 1\colon k),$$

$$\mathbf{v}_k^T\mathbf{u}_k + d_k = A(k+1, k+1),$$

and hence, using superindex $-T$ to denote the inverse of the transpose,

$$\mathbf{u}_k = L_k^{-1}A(1\colon k, k+1),$$

$$\mathbf{v}_k = U_k^{-T}A(k+1, 1\colon k)^T,$$

$$d_k = A(k+1, k+1) - \mathbf{v}_k^T\mathbf{u}_k.$$

2. Updating the LU decomposition without pivoting (2/3)

This procedure, which was named *Sherman's march* by Stewart [Stewar98, p. 169] in reference to a procession proceeding to the southeast from Tennessee to Georgia, can be implemented efficiently in MATLAB:

```
Lk = 1; Uk = A(1,1);
for k = 1:n-1
  uk = Lk\A(1:k,k+1);
  vkt = A(k+1,1:k)/Uk;
  dk = A(k+1,k+1)-vkt*uk;
  Lk = [Lk zeros(k,1); vkt 1];
  Uk = [Uk uk; zeros(1,k) dk];
end
```

Our code avoids transposing the upper triangular factor to economize in memory allocation and access. Main drawback: it does not allow pivoting to be taken into account—maybe the reason it's not even described in most NA textbooks! But ...

2. Updating the LU decomposition without pivoting (3/3)

...our matrix A is strictly diagonally dominant by rows and hence it's well-known [BurFai05, p. 399] that Gaussian elimination can be performed without pivoting in a numerically stable way (no growth of round-off errors) for this class of matrices!

In other words, taking advantage of the diagonal dominance property has been crucial to obtain this fifth approach:

SH. Update the LU decomposition without pivoting $A_k = L_kU_k$ knowing $A_{k-1} = L_{k-1}U_{k-1}$.

However, the code seems to be quite complex when compared with that of recomputed LU or that of direct backslashing. Now a natural question arises: does it lead to a significant reduction in the CPU time, while maintaining a good numerical quality of the obtained approximate solutions?

3. Preliminary numerical experiments (1/4)

To compare these five approaches, check both CPU time needed to complete whole process and numerical quality of approximate solutions $\hat{\mathbf{x}}_k$ for each k; latter by measuring backward error, as indicated by rule of thumb

"forward error (FE) $\lesssim$ backward error (BE) $\times$ condition number (CN)"

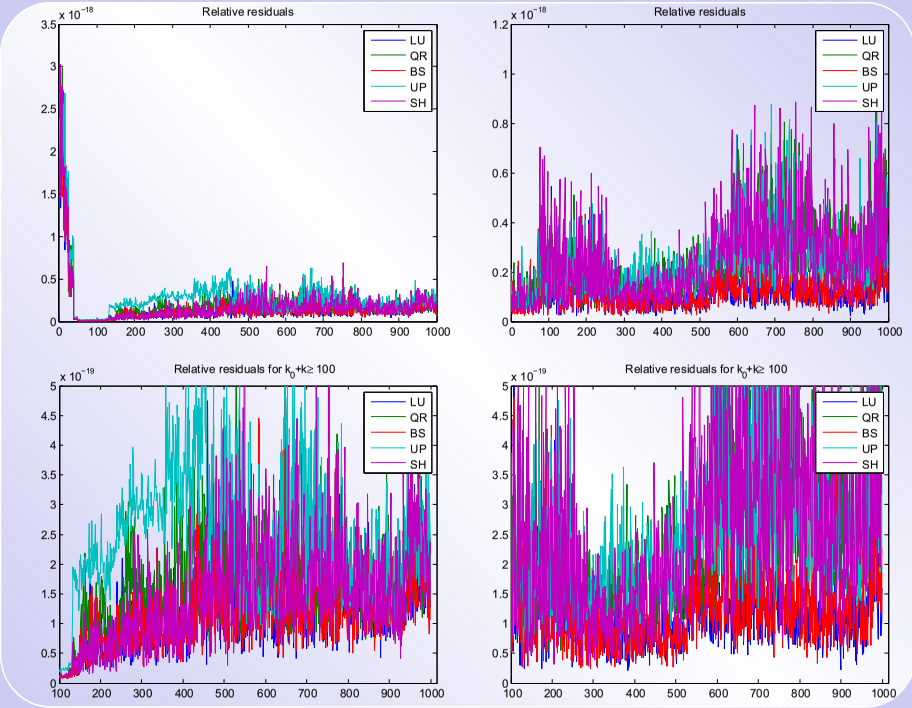
FE measures (absolute or relative) distance between exact and computed solutions, BE does *method*'s stability, and CN does *problem*'s stability.

A suitable scale-independent measurement is Wilkinson's relative residual [Higham02, p. 12]: normwise relative BE defined as

$$\frac{\|\mathbf{b}_k - A_k\hat{\mathbf{x}}_k\|_2}{\|A_k\|_2\,\|\hat{\mathbf{x}}_k\|_2}.$$

The lesser its BE, the better an approximate solution computed by a given method is. For our experiments not to take a long time, we have used Frobenius norm

$\|A_k\|_F \triangleq \sqrt{\sum_{i,j\in 1:\,k} |a_{ij}|^2}$ rather than 2-norm $\|A_k\|_2 \triangleq \sqrt{\max_{i\in 1:\,k} |\lambda_i(A_k^T A_k)|}$.



3. Preliminary numerical experiments (2/4)

The comparison was done in MATLAB v7.5 with an Intel Pentium 4, 2.8 Ghz, 768 Mb RAM under Windows XP. The right-hand-side vector $b \in \mathbb{R}^n$ has been randomly generated from a standard normal distribution. We have considered two real-world matrices (taken from the simulator) with $n = 1020$ and $n = 1200$, and both have been checked to be strictly diagonally dominant by rows. These values of n seem to be quite representative of the real situations that our solvers are going to deal with. Instead of starting from A_1, we have started from A_{k_0} with $k_0 = 21$ in the former case and $k_0 = 201$ in the latter case, in order to perform 999 iterations for both examples.

The left side of figure 1 shows the relative residuals for the $n = 1020$ example, whereas the right side shows those for the $n = 1200$ example; lower graphs in this figure limit the greatest relative residuals to $5 \cdot 10^{-19}$ and discard the first 100 iterations to show a cleaner zoom-in window for both examples.

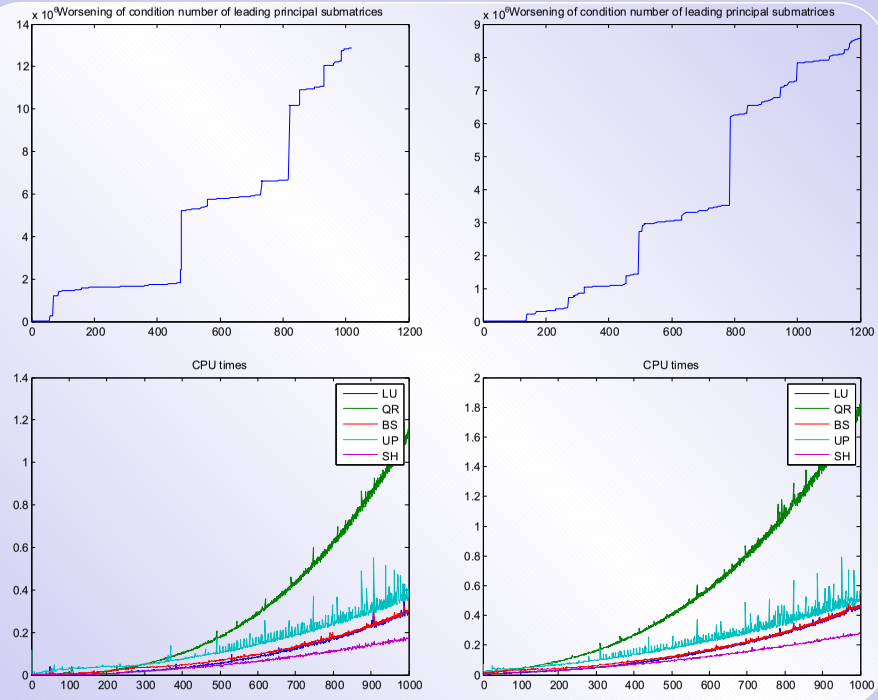
Figure 1: Relative residuals ⇒

3. Preliminary numerical experiments (3/4)

The backward residuals obtained in all situations were pretty small, always less than $3 \cdot 10^{-18}$ and with no significant differences among the five approaches. Hence we can conclude that the five approaches allow us to obtain approximate solutions that can be regarded as very satisfactory from a numerical analysis point of view in spite of the progressive ill-conditioning (measured with the condition number $\|A_k\|_F\|A_k^{-1}\|_F$ using the Frobenius norm to be consistent with the relative residuals) of the leading principal submatrices displayed in the upper graphs of figure 2. As above, the left side corresponds to the $n = 1020$ example and the right side to the $n = 1200$ example.

Regarding the CPU times (shown in the lower graphs of figure 2), it is clear that **QR** is the worst approach and **SH** is the best one as k increases. Although **BS** and **LU** are slightly better than **UP**, these three approaches seem to get asymptotically closer as k increases (this effect is cleaner in the $n = 1200$ example).

Figure 2: Condition numbers and CPU times ⇒



3. Preliminary numerical experiments (4/4)

Total CPU times for the whole processes in the $n = 1020$ example, for which **SH** needs 58.71 sec, were as follows:

- **LU** needs 86.47 sec (x1.5 slower than **SH**, could be improved a 32%)
- **QR** needs 310.28 sec (x5.3 slower than **SH**, could be improved a 81%)
- **BS** needs 95.69 sec (x1.6 slower than **SH**, could be improved a 39%)
- **UP** needs 139.78 sec (x2.4 slower than **SH**, could be improved a 58%)

and those in the $n = 1200$ example, for which **SH** needs 111.02 sec, were as follows:

- **LU** needs 153.47 sec (x1.4 slower than **SH**, could be improved a 28%)
- **QR** needs 571.10 sec (x5.1 slower than **SH**, could be improved a 81%)
- **BS** needs 163.44 sec (x1.5 slower than **SH**, could be improved a 32%)
- **UP** needs 216.35 sec (x1.9 slower than **SH**, could be improved a 49%)

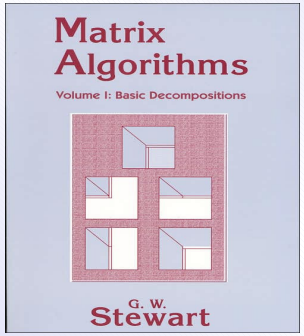
4. Conclusion

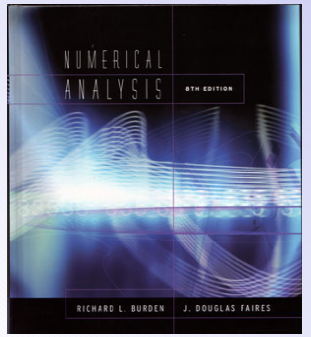
Professor Acton's celebrated history (dated sixty years ago) on being asked to invert a matrix that turned out to be an orthogonal one [Acton90, p. 246] teach us to look carefully to the properties of the matrices being involved in every engineering problem. We have been faced with another instance of this cautionary tale in which not taking advantage of diagonal dominance property of matrices involved would have prevented us to obtain as much as a 30% improvement on average in time required to complete whole process!

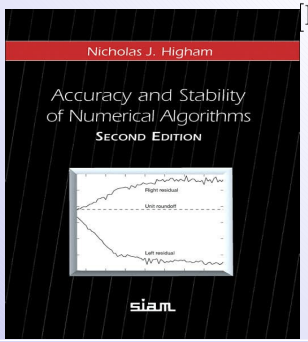
Addendum. Some final questions

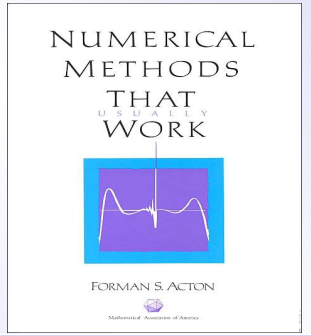
1. Can you manage the case in which a given column and row are deleted?
2. Are you still thinking that symbolic computations are nowadays powerful enough to dispense with numerical computations of any kind?
3. How large are the systems of linear equations you have ever faced with? Can they be solved with your favorite CAS/graphics calculator?

References









[Stewar98]
[BurFai05]
[Higham02]
[Acton90]