ALOP Autumn School
High performance solution of linear optimization problems

Numerical linear algebra

1. The matrix A in the file `bcsstk08.mat` is square of dimension $n = 1074$. The time required to solve $A\boldsymbol{x} = \boldsymbol{b}$ by computing A^{-1} as `Ainv` and then forming `Ainv*b` is not always the same, but it is generally at least an order of magnitude longer than the time required to solve $A\boldsymbol{x} = \boldsymbol{b}$ by using `A\b`. The residual norm $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ is larger when forming `Ainv*b` than when solving $A\boldsymbol{x} = \boldsymbol{b}$ by using `A\b`, but the norm of the solution error $\|\boldsymbol{x} - \boldsymbol{x}^*\|_2$ is similar. Similar behaviour is shown for `nasa1824.mat`. However, for `pds02.mat` the computed solution is exactly correct using forming `Ainv*b`, and close to machine precision with `A\b`.

2. Note that the times below relate to a particular run on a particular PC. Averages of repeat runs should, perhaps, be used and the results will vary from one class of processor to another.

   The results in the first table slow show that, despite the arrowhead matrix being sparse, its inverse is full and, if sparsity is not exploited, the LU factors are also completely full. This reflects the behaviour of Gaussian elimination see in Question 2 of Workshop 2. The time taken to compute the decomposition $LU = PA$ is 2-3 times less than the time required to compute A^{-1}, reflecting the computational cost of $\frac{2}{3}n^3$ for $LU = PA$ and $\frac{8}{3}n^3$ for A^{-1} obtained in Assignment 1.

   However, when using the sparsity-exploiting LU decomposition `[L U P Q]=lu(SparseA)` there is no fill-in (as indicated by the fill factor of 1). Again, this reflects the behaviour of Gaussian elimination in Question 2 of Workshop 2 when the rows and columns were re-ordered to force a more efficient pivot sequence. Unsurprisingly, the time taken to compute the sparsity-exploiting LU decomposition is substantially (30–100 times) less than the time when sparsity is not exploited.

Problem	Method	Nonzeros in result	Fill factor	Density	Time
For `Arrowhead`	`inv(A)`	$\tau(A^{-1}) = 10000$	33.56	100	0.0147
$n = 100$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 10000$	33.56	100	0.0214
$\tau(A) = 298$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 298$	1.00	3	0.0003
For `Arrowhead`	`inv(A)`	$\tau(A^{-1}) = 40000$	66.89	100	0.0043
$n = 200$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 40000$	66.89	100	0.0237
$\tau(A) = 598$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 598$	1.00	1	0.0011
For `Arrowhead`	`inv(A)`	$\tau(A^{-1}) = 160000$	133.56	100	0.0131
$n = 400$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 160000$	133.56	100	0.0871
$\tau(A) = 1198$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 1198$	1.00	1	0.0008
For `Arrowhead`	`inv(A)`	$\tau(A^{-1}) = 640000$	266.89	100	0.0366
$n = 800$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 640000$	266.89	100	0.0154
$\tau(A) = 2398$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 2398$	1.00	0	0.0019
For `Arrowhead`	`inv(A)`	$\tau(A^{-1}) = 2560000$	533.56	100	0.1385
$n = 1600$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 2560000$	533.56	100	0.0457
$\tau(A) = 4798$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 4798$	1.00	0	0.0060
For `Arrowhead`	`inv(A)`	$\tau(A^{-1}) = 10240000$	1066.89	100	0.7762
$n = 3200$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 10240000$	1066.89	100	0.3646
$\tau(A) = 9598$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 9598$	1.00	0	0.0193

   For the three test matrices `bcsstk08.mat`, `nasa2910.mat` and `c-36.mat`, in each case A^{-1} is (essentially) full. If sparsity is not exploited, the LU factors are also far from full and, in the case of `nasa2910.mat`, the fill-factor is little more than for the sparsity LU factors. However, in all cases it is clearly far more efficient to form LU factors by exploiting sparsity.

Problem	Method	Nonzeros in result	Fill factor	Density	Ti
For `bcsstk08.mat`	`inv(A)`	$\tau(A^{-1}) = 1147044$	88.51	99	0.1
$n = 1074$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 622964$	48.07	54	0.1
$\tau(A) = 12960$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 61224$	4.72	5	0.0
For `nasa2910.mat`	`inv(A)`	$\tau(A^{-1}) = 8468100$	48.58	100	0.8
$n = 2910$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 1262776$	7.25	15	0.2
$\tau(A) = 174296$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 401665$	2.30	5	0.0
For `c-36.mat`	`inv(A)`	$\tau(A^{-1}) = 55935441$	848.26	100	29.7
$n = 7479$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 21331716$	323.50	38	29.5
$\tau(A) = 65941$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 167062$	2.53	0	0.0
For `pds-02.mat`	`inv(A)`	$\tau(A^{-1}) = 45601$	7.49	1	0.3
$n = 2953$	`[L U P]=lu(A)`	$\tau(L) + \tau(U) = 8438$	1.39	0	0.0
$\tau(A) = 6088$	`[L U P Q]=lu(SparseA)`	$\tau(L) + \tau(U) = 6115$	1.00	0	0.0

Note that it is not always the case that the inverse of a (sparse) matrix is full, as is demonstrated by the results for `pds02.mat`, a matrix which, due to particular structural properties is said to be *hyper-sparse*. This is a phenomenon which was identified and exploited in the case of linear programming problems by Hall and McKinnon.

3. (a) $A = \lambda I$ of dimension $n = 1000$ has one eigenvalue λ of multiplicity 1000. CG terminates in one iteration. The problem is perfectly conditioned.

   (b) The matrix A from `662_bus.mat` has all real positive eigenvalues.
   CG performs all $n = 662$ iterations in 0.050 sec, achieving
   `||Ax-b||=1.87386e-006` and `||x-x^*||=7.90652e-007`
   Although the termination criterion has not been reached, the solution has some accuracy. The magnitudes of the eigenvalues lie in $[0.00504717, 4008.11]$ which (to an experienced eye) is not excessive so the problem is not too ill-conditioned for CG to retain some accuracy after n iterations.
   `A\b` achieves `||Ax-b||=5.26076e-013` and `||x-x^*||=2.53906e-012` in 0.036 sec: faster and more accurate.

   (c) The matrix A from `bcsstk08.mat` has all real positive eigenvalues.
   CG performs all $n = 1074$ iterations in 0.10 sec, but
   `||Ax-b||=80715.2` and `||x-x^*||=1.99512`
   so the solution has no accuracy. The magnitudes of the eigenvalues lie in $[2946.41, 7.65703 \times 10^{10}]$ which is rather large so the problem is too ill-conditioned for CG to retain any accuracy after n iterations.
   `A\b` achieves `||Ax-b||=2.00918e-005` and `||x-x^*||=6.28242e-012` in 0.05 sec: faster and some accuracy [although ten orders of magnitude larger than machine precision, further illustrating the ill-conditioning of A.]

   (d) The matrix $A = I + \epsilon E$ of dimension $n = 1000$, has all real positive eigenvalues for $\epsilon = 10^{-1}$.
   CG performs 2 iterations in 0.00042 sec for $\epsilon = 10^{-8}$, and more iterations for larger values of ϵ, but still only 13 in 0.0016 sec when $\epsilon = 10^{-1}$, terminating with
   `||Ax-b||=5.73638e-009` and `||x-x^*||=6.07421e-009`
   The eigenvalues are clustered in the interval $[0.66929, 1.57948]$ which means that the problem is so well conditioned that CG soon finds an accurate solution.
   `A\b` achieves `||Ax-b||=1.24284e-14` and `||x-x^*||=1.25086e-014` in 0.041 sec: much slower but more accurate.
   This example illustrates that if the system to be solved is sufficiently well conditioned, even a simple CG implementation can be much faster than MATLAB's best direct solver `A\b`.

(e) The matrix $A = I + \boldsymbol{u}\boldsymbol{u}^T$ of dimension $n = 1000$, has two distinct eigenvalues: $\lambda = 2$ (of multiplicity 1) and $\lambda = 1$ (of multiplicity $n - 1$).

CG performs 2 iterations in 0.0041 sec, terminating with
`||Ax-b||=4.02057e-014` and `||x-x^*||=4.02281e-014`
The two distinct eigenvalues means that the problem is solved exactly in two CG iterations. It is possible to show this on paper, but the algebraic manipulation is very involved.

`A\b` achieves `||Ax-b||=3.97961e-014` and `||x-x^*||=3.95951e-014` in 0.035 sec: much slower and no more accurate.

This example illustrates the provable property that if the matrix A has m distinct eigenvalues then CG (and any other Krylov subspace method) terminates in (at most) m iterations. Part (2a) illustrates the case $m = 1$ when A has a single eigenvalue of multiplicity n.