An iterative least-square method suitable for solving large sparse matrices

By I. M. Khabaza

The purpose of this paper is to report on the results of numerical experiments with an iterative least-square method suitable for solving linear systems of equations with large sparse matrices.

The method of this paper gives satisfactory results for a wide variety of matrices and is not restricted to real or symmetric or definite matrices. We consider a system of n equations:

$$Ax = b. \tag{1}$$

As a library program the method requires from the user an auxiliary sequence which, given x, produces $y = Ax$. Thus no knowledge of the elements of A is required. A need not be a matrix; it could be any linear operator.

Given an approximate solution x we calculate the residual vector r:

$$r = b - Ax. \tag{2}$$

We seek a better approximation x' of the form

$$x' = x + f(A)r \tag{3}$$

where $f(A)$ is a matrix polynomial of the form

$$f(A) = c_1 I + c_2 A + c_3 A^2 + \ldots + c_m A^{m-1}. \tag{4}$$

The true correction in (3) is $A^{-1}r$, and from the Cayley-Hamilton theorem, A^{-1} may be represented as a polynomial in A of degree $n-1$ in general. Thus $f(A)r$ is an approximation to $A^{-1}r$, but $f(A)$ is of order $m-1$ where m is a small integer, about 3, whereas n, the dimension of the vector x, may be large. The polynomial f is determined by the vector c of dimension m:

$$c = (c_1, c_2, \ldots, c_m)^T. \tag{5}$$

The superfix T denotes transposition so that c is a column vector.

We define the polynomial $g(\lambda)$ as follows:

$$g(\lambda) = 1 - \lambda f(\lambda) = 1 - c_1\lambda - c_2\lambda^2 - \ldots - c_m\lambda^m. \tag{6}$$

The coefficients $c_1, c_2, \ldots, c_m$ are determined by the method of least-squares so that $\|g(A)r\|$ is a minimum; the double strokes here denote the usual norm, i.e. the square root of the sum of the squares of the moduli of the components of a vector. We take, as an example, $m = 3$. We define $r_1 = Ar, r_2 = Ar_1, r_3 = Ar_2$. Then we determine c_1, c_2, c_3 from the system of equations

$$\begin{aligned} a_{11}c_1 + a_{12}c_2 + a_{13}c_3 &= b_1 \\ a_{21}c_1 + a_{22}c_2 + a_{23}c_3 &= b_2 \\ a_{31}c_1 + a_{32}c_2 + a_{33}c_3 &= b_3. \end{aligned} \tag{7}$$

Here $a_{ij} = a_{ji} = r_i^T r_j$, $b_i = r_i^T r$.

Let the eigensystem of the matrix A be:

$$\begin{matrix} a = \lambda_1, \lambda_2, \ldots, \lambda_n = b \\ \xi_1, \xi_2, \ldots, \xi_n \end{matrix}$$

where ξ_i is the eigenvector corresponding to the eigenvalue λ_i. In general we can express the residual vector r of equation (2) in the form

$$r = a_1\xi_1 + a_2\xi_2 + \ldots + a_n\xi_n \tag{8}$$

where a_i is the component of r along ξ_i.

If we apply the iteration (3) s times the residual becomes

$$a_1 g_1^s \xi_1 + a_2 g_2^s \xi_2 + \ldots + a_n g_n^s \xi_n \tag{9}$$

where $g_i = g(\lambda_i)$.

Thus convergence occurs only if $|g_i| < 1$ for all i. The graph of $g(\lambda)$ in the range (a, b) will indicate the rate of convergence. The iteration (3) will eliminate rapidly the components corresponding to roots where $g(\lambda) \approx 0$; it will blow up components corresponding to roots where $|g(\lambda)| > 1$. We note that $g(0) = 1$.

The program

We give here a description of the program developed for testing the method of this paper. Starting with an approximation x we calculate the residual r; if no approximation is known we can take $x = 0$. To obtain the coefficients $c_1, c_2, \ldots, c_m$ of equation (4) we require m multiplications by the matrix A and an extra multiplication to find the new residual r'; the new approximation x' is determined by using the same multiplications by A used for determining c. For each further approximation, obtained by iterating with a previously calculated set of coefficients c, we require $m - 1$ multiplications by A to calculate x' and a further multiplication to calculate r'. Let

$$v = \|r\|; \qquad v' = \|r'\|. \tag{10}$$

After each iteration we compare v with v'. Iteration with the same set c is continued as long as $v' < Cv$, where C is a convergence control parameter; otherwise a new set of coefficients c is calculated. The parameter C is given a small value such as $C=0{\cdot}2$, if we expect rapid convergence, for slow convergence we take $C = 0{\cdot}8$ or $0{\cdot}9$. If $v' > v$, as may happen if $g(\lambda)$ blows up some

components of the residual, it may be thought that it would be better to discard the latest approximation x' and its residual r' and calculate a new set c from r. In fact this turned out to be wrong. This phenomenon is similar to over-relaxation. Indeed it often paid to iterate once more although the residual is increasing. The program contains further parameters, D and F (e.g. $D = 2$, $F = 10$) which determine when to stop iterating when the residual is increasing (e.g. when $v' > Dv_0$ where v_0 is the smallest previous residual), and when to reject the latest approximation (e.g. when $v' > Fv_0$). The required solution is obtained when $v' < E$ where E is another parameter (e.g. $E = 0{\cdot}000001$). Finally, another parameter specifies the value of m which determines the dimension of c or the order of the polynomial $g(\lambda)$ and the system of equations (7). Various small values of m were tried; in most cases $m = 3$ seemed to be the most suitable value.

Numerical examples

We give the numerical results for five examples. In each of the first four examples the initial approximation x was taken to be $\mathbf{0}$; a known approximation, however rough, would of course be better. To measure the efficiency of the method we count the number of multiplications by A and this number is compared with the corresponding number required to obtain the same accuracy by existing methods. The existing methods tried were the method of Gauss-Seidel with over-relaxation (see Martin and Tee, 1961, equations 2·5 and 2·12) and the method of Conjugate Gradients (op. cit. equations 7·10 and 7·11).

In the first three examples the right-hand side b of equation (1) was taken to be $e = (1,1,..,1)^T$; $n = 20$; $A = [1, W]$, the codiagonal matrix for which $A_{ii} = 1$, $A_{i,i-1} = A_{i,i+1} = W$, otherwise $A_{ij} = 0$. We considered the values $W = -0{\cdot}25, -0{\cdot}5, -0{\cdot}6$, respectively. The roots λ_i of A are given by the formula $\lambda_i = 1+2W\cos\left(\frac{i\pi}{n+1}\right)$. This result is used to determine the range (a, b) of the roots of A and draw the graph of $g(\lambda)$ in this range.

In each example we give a table. The c-column gives the coefficients c_1, c_2, c_3. The v-column following it gives the values of the residuals for approximations obtained by repeated iteration with c.

The computation was carried out on the Ferranti Mercury Computer of the University of London Computer Unit. This is a machine with a single-length, floating-point accumulator consisting of 10 binary digits for the exponent and 30 binary digits for the argument. All calculations were carried out in single-length arithmetic.

Well-conditioned matrices

We consider the case when $W = -0{\cdot}25$.

The roots of A lie between 0·5 and 1·5. The graph of $g(\lambda)$ in Fig. 1 corresponds to the first set c of Table 1.

The final solution obtained has eight figure accuracy; this required a total of 14 multiplications by A. To obtain the same accuracy by the method of Conjugate Gradients required 16 multiplications by A, and by the method of over-relaxation required 14 multiplications by A, counting each iteration as equivalent to one multiplication, and using the optimum over-relaxation parameter $w = 1{\cdot}07$. As in practical problems the optimum w is not known; this suggests that the method of this paper may be faster.

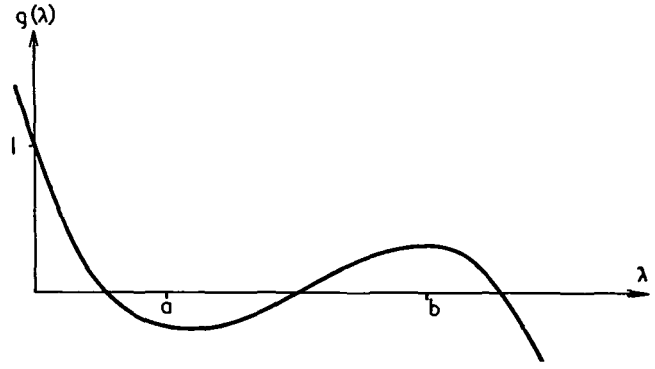

Fig. 1

Table 1

c	v	c	v
3·91	0·035	3·08	0·000009
−4·76	0·003	−2·96	0·0000002
1·81		0·90	

Ill-conditioned matrices

We consider the case $W = -0{\cdot}5$; this is the one-dimensional Laplace operator.

The roots of A lie between 0·01 and 1·99 giving a condition number of about 200. Fig. 2 gives the graph of $g(\lambda)$ corresponding to the second and third sets c of Table 2. The graph of the larger set is not drawn to scale, it oscillates between 3 and −82 in the range.

The final solution obtained has eight figure accuracy; this required a total of 48 multiplications by A. The same equation was solved by the method of over-relaxation using the optimum over-relaxation parameter $w = 1{\cdot}74$; to get the same accuracy required the equivalent of 67 multiplications by A (counting each iteration as equivalent to one multiplication). The method of conjugate gradients broke down in this case; this is because, for ill-conditioned matrices, some of the intermediate calculation requires accumulation of double-length products, which is not easy on a single-length accumulator.

We observe that the vectors c fall into two categories, small and large, which occur alternately. We could make use of this "alternate directions" phenomenon to economize on the calculations of new sets c. Thus by

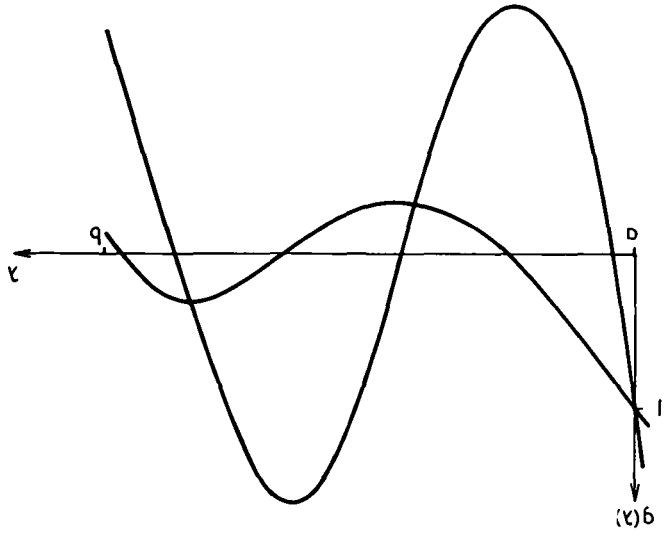

Fig. 2

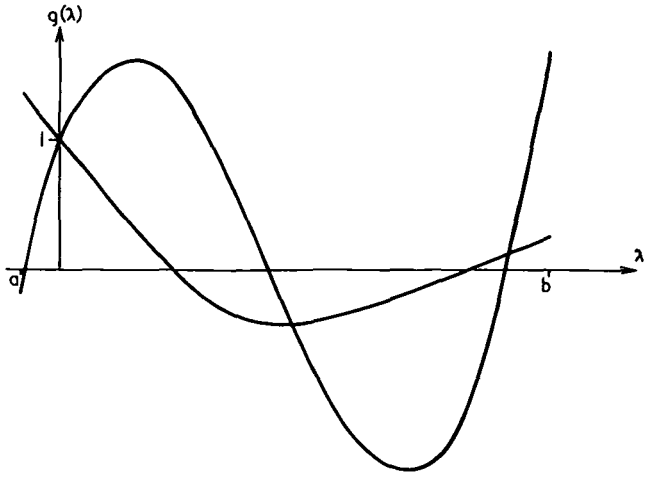

Fig. 3

Table 2

c	*v*	*c*	*v*	*c*	*v*
12	3·74	5·22	2·57	86·18	0·56
−20	3·74	−5·16	2·40	−209·93	72·28
8	9·90	1·41	2·26	117·49	
c	*v*	*c*	*v*	*c*	*v*
3·57	0·137	91·29	0·00028	4·89	0·000005
−3·63	0·132	−157·85	0·0087	−4·64	
1·08	0·126	66·88		1·23	

applying the second and third sets alternately it was possible to achieve the same accuracy, but it required a total of 57 multiplications by A.

Ill-conditioned non-definite matrices

We consider the case $W = -0{\cdot}6$.

The roots of A lie between $-0{\cdot}19$ and $2{\cdot}19$, and one of the roots is equal to $0{\cdot}0085$, giving a condition number of about 250. Fig. 3 gives the graph of $g(\lambda)$ corresponding to two consecutive sets $\boldsymbol{c}$.

The final solution has eight figure accuracy; this required a total of 98 iterations. This large number is partly due to the fact that $g(\lambda)$ cannot be numerically small throughout the range (a, b), since the origin is within the range and $g(0) = 1$; but it is probably mostly due to the ill-condition of the matrix. The method was also tried for the case $W = -10$; convergence was a little faster because the matrix is better conditioned. In this case the roots are about equally spread on both sides of the origin; in such cases there is some advantage in taking m even, e.g. $m = 4$.

Table 3

c	*v*	*c*	*v*	*c*	*v*
−3·47	1·58	5·54	1·02	−9·78	0·29
9·01	2·02	−5·32	0·91	25·27	1·18
−3·81	5·08	1·37	1·26	−11·66	
c	*v*	*c*	*v*	*c*	*v*
6·48	0·17	−2·62	0·0096	−5·87	0·004
−7·71	0·07	14·62	0·0073	9·85	0·006
2·36	0·03	−6·06	0·0318	−3·68	0·012
	0·02				
	0·04				
c	*v*	*c*	*v*	*c*	*v*
9·76	0·001	−11·04	0·0002	4·33	0·000023
−14·74	0·003	13·21	0·0007	−5·20	0·000022
4·85		−3·76		1·57	0·000038
c	*v*	*c*	*v*		
−2·63	0·000007	7·09	0·000004		
13·26	0·000015	−8·62			
−5·83		2·77			

The two-dimensional Laplace operator

In this case the components of the vector x correspond to the values $x(i, j)$ of a function of two variables which satisfies the Laplace equation in discrete form at the 81 internal points of a 10 by 10 square mesh. The internal points correspond to $i, j = 1, 2, \ldots, 9$. The boundary points correspond to $i = 0$ or 10, or $j = 0$ or 10; at these points x was specified by the formula

$$x(i, j) = i^3 - 3ij^2. \qquad (11)$$

We denote $x(i, j)$, $x(i, j-1)$, $x(i-1, j)$, $x(i, j+1)$, $x(i+1, j)$ by x_0, x_1, x_2, x_3, x_4, respectively. Then equation (1) takes the form

$$x_0-\tfrac{1}{4}(x_1+x_2+x_3+x_4)=0 \quad i,j=1,2,\ldots,9. \quad (12)$$

The operation $y=Ax$ is obtained by setting $x=0$ at the 40 boundary points and applying the substitution

$$y_0=x_0-\tfrac{1}{4}(x_1+x_2+x_3+x_4) \quad (13)$$

at the 81 internal points. The residual $r=b-Ax$ is obtained by first setting the boundary values according to (11), and using the substitution

$$r_0=\tfrac{1}{4}(x_1+x_2+x_3+x_4)-x_0 \quad (14)$$

at the 81 internal points.

Equations (12) are equivalent to the equation $Ax=b$ where A has the form $I-L-U$ (Martin & Tee, 1961, equation 1·2). To speed up convergence (A,b) are replaced by (A',b') where $A'=I-(I-L)^{-1}U$, $b'=(I-L)^{-1}b$. This is done by replacing the substitutions (13) and (14) by the following:

$$u_0=\tfrac{1}{4}(u_1+u_2+x_3+x_4) \quad y_0=x_0-u_0 \quad i,j=1,2,\ldots,9 \quad (13')$$

$$u_0=\tfrac{1}{4}(u_1+u_2+x_3+x_4) \quad r_0=u_0-x_0 \quad i,j=1,2,\ldots,9. \quad (14')$$

In (13′) as in (13) the values of u and x are set to 0 at the boundary points. In (14′) as in (14) the values of u and x are set at the boundary points according to equation (11). The values of u obtained in (14′) form the solution one would obtain after a single Gauss-Seidel iteration (op. cit., equation 2·2), and r_0 is the corresponding displacement.

The initial approximation in Table 4 was $x=0$.

Table 4

c	*v*	*c*	*v*	*c*	*v*
4·31	348	49·52	29·9	7·04	0·574
−5·89	137	−15·14	10·0	−11·36	0·254
2·58	72·2	6·60	5·78	4·46	0·097
					0·030
					0·013

c	*v*	*c*	*v*
10·64	0·0037	2·85	0·000041
−24·60	0·0026	−2·79	0·000022
13·73	0·0033	0·94	
	0·0036		

The final solution obtained has eight figure accuracy; this required a total of 55 multiplications by A'. The same equations solved by the method of over-relaxation, using the over-relaxation parameter $w=1\cdot518$, required the equivalent of 44 multiplications. In this case the method of this paper requires more multiplications than the method of over-relaxation. In practical problems, however, one has only a guess at the optimum w; thus taking $w=1\cdot4$ requires more than 70 iterations to obtain the same accuracy. It would appear that in such cases the method of this paper still has an advantage over the method of over-relaxation.

Complex matrices

The method of this paper was also tried on an 11 by 11 complex matrix of the type which arises in Power Systems analysis in Electrical Engineering (Laughton & Humphrey Davies). Iterative methods in this field are particularly suitable because a good approximation is usually known; because the matrices are very sparse and can be very large; because the solution is usually required only to about four or five significant figures; and finally because the solution itself is part of an iterative procedure which may involve altering the coefficients of the matrix after each iteration.

The matrix studied was typical: it is symmetric not Hermitian; in any row the sum of the elements is 0, or nearly 0, and the diagonal element is the largest in modulus; the smallest diagonal element is $0-7\cdot69i$ and the largest is $7\cdot960-44\cdot277i$.

The coefficients a_{ij} and b_i of equation (7) are defined as follows:

$$a_{ij}=r_i^H r_j; \qquad b_i=r_i^H r$$

where the superfix H denotes the complex conjugate transpose.

Table 5

c	*v*	*c*	*v*
0·033+0·235*i*	0·73	0·964+1·652*i*	0·05
0·0125−0·004*i*	0·46	0·396−0·596*i*	2·46
−0·00007−0·00013*i*	0·34	−0·010−0·004*i*	

c	*v*	*c*	*v*
0·025+0·121*i*	0·031	3·58+2·45*i*	0·0009
0·004−0·002*i*	0·028	0·63−2·12*i*	
−0·00002−0·00004*i*	0·0278	−0·033−0·003*i*	

The last approximation has a maximum error of one in the fifth significant figure; this accuracy required 55 multiplications by A. To obtain the same accuracy by the method of Gauss-Seidel required much more than 100 multiplications.

Concluding remarks

The method of this paper seems to be an efficient way for solving large sparse matrices or linear operators. It compares favourably with existing iterative methods. It also copes with complex or non-definite or unsymmetric matrices, whereas existing methods usually require the matrix to be symmetric definite. As for other iterative methods, its advantage over direct methods is that it requires few iterations if an approximation is already known or if only few figures accuracy is required or if the matrix is well conditioned; it requires little storage if the matrix is sparse, and indeed A may be stored not as a matrix but as a program to apply a linear operator.

The method is being tried on Atlas for matrices of order several hundreds which arise in Electrical and

Structural Engineering; this was not possible before because of the relatively small size of the Mercury fast store; I hope to report on the results in the near future.

I am indebted to the Director of the University of London Computer Unit, Dr. R. A. Buckingham, for encouragement, and to my colleague, Dr. M. J. M. Bernal, for frequent discussions.

References

MARTIN, D. W., and TEE, G. J. (1961). "Iterative methods for linear equations with symmetric positive definite matrix", *The Computer Journal*, Vol. 4, p. 242.

LAUGHTON, M. A., and HUMPHREY DAVIES, M. W. "Numerical methods for Power System Load Flow studies". To be published in the *Proceedings* of the *Institution of Electrical Engineers*.

Note on the numerical solution of linear differential equations with constant coefficients

By R. E. Scraton and J. W. Searl

The numerical solution of the differential equation

$$ay'' + by' + cy = f(x) \tag{1}$$

where a, b, c are constants and $f(x)$ is a numerically specified function, can be obtained by a variety of methods. For automatic computation the Runge–Kutta method is normally used, but this may be unstable. The procedure described below provides an alternative method which has been found satisfactory where the Runge–Kutta method has failed.

Suppose that $f(x)$ is tabulated at interval h. In the usual notation, let f_p denote $f(x_0 + ph)$ so that equation (1) may be written

$$a\frac{d^2y}{dp^2} + bh\frac{dy}{dp} + ch^2y = h^2f_p. \tag{2}$$

It is assumed that y_0 and y_0 are known, so that a procedure for determining y_1 and y_1' makes it possible to tabulate both y and y' in a step-by-step manner.

Let the sequence λ_r be defined by the equations

$$\left.\begin{aligned} \lambda_0 &= \frac{1}{a} \\ \lambda_1 &= -\frac{bh}{a^2} \\ a\lambda_{r+2} + bh\lambda_{r+1} + ch^2\lambda_r &= 0, \qquad r \geqslant 0 \end{aligned}\right\} \tag{3}$$

and let

$$U_m(p) = \frac{\lambda_0 p^m}{m!} + \frac{\lambda_1 p^{m+1}}{(m+1)!} + \frac{\lambda_2 p^{m+2}}{(m+2)!} + \ldots \tag{4}$$

It is easily verified that

$$a\frac{d^2U_m}{dp^2} + bh\frac{dU_m}{dp} + ch^2U_m = \left\{\begin{array}{ll} 0 & \text{if } m = 0, 1 \\ \dfrac{p^{m-2}}{(m-2)!} & \text{if } m \geqslant 2. \end{array}\right\} \tag{5}$$

If, therefore, f_p can be expanded in the form

$$f_p = A_0 + A_1p + A_2p^2 + A_3p^3 + \ldots \tag{6}$$

it can be shown that the solution of (2) satisfying the required initial conditions is

$$\begin{aligned} y = y_0[1 - ch^2U_2(p)] + ahy_0'U_1(p) \\ + h^2[A_0U_2(p) + A_1.1!U_3(p) \\ + A_2.2!U_4(p) + \ldots]. \end{aligned} \tag{7}$$

Thus

$$\begin{aligned} y_1 = y_0[1 - ch^2u_2] + ahy_0'u_1 \\ + h^2(A_0u_2 + A_1.1!\,u_3 + A_2.2!\,u_4 \ldots) \end{aligned} \tag{8}$$

and

$$\begin{aligned} hy_1' = -ch^2u_1y_0 + ahy_0'u_0 \\ + h^2(A_0u_1 + A_1.1!\,u_2 + A_2 2!\,u_3 \ldots) \end{aligned} \tag{9}$$

where

$$u_m = U_m(1) = \frac{\lambda_0}{m!} + \frac{\lambda_1}{(m+1)!} + \frac{\lambda_2}{(m+2)!} + \ldots$$

The coefficients A_r of equation (6) may be taken from any polynomial interpolation formula. For automatic computation, Lagrangian formulae are appropriate, and a four-point formula will be used as an illustration, viz:

$$\begin{aligned} f_p = -\frac{1}{6}(p^3 - 3p^2 + 2p)f_{-1} \\ + \frac{1}{2}(p^3 - 2p^2 - p + 2)f_0 \\ - \frac{1}{2}(p^3 - p^2 - 2p)f_1 + \frac{1}{6}(p^3 - p)f_2. \end{aligned} \tag{10}$$

Equations (8) and (9) may then be written