Academic Studies - National Journal of Jyoti Research Academy
Feb 2007, Vol. 1(1), 1-8

ASNJJRA

Research Publication and Development Cell
Jyoti Nivas College (Autonomous)

Linear Systems: Relook, Concise Algorithms, and Matlab Programs

Dr. S.K. Sen and Sagar Sen

Abstract

A linear system consists of linear equations $Ax = b$ and/or linear inequalities $Ax \leq b$, where A is an $m \times n$ known matrix and b is a known $m \times 1$ vector. A and b could be real or complex with no sign restriction on their elements. Other possible inequalities, viz., $\geq, \neq, <,$ or $>$ could also be there in each of the systems. Such systems happen to be mathematical models of numerous real-world problems and have been dealt with by numerous people over decades. Yet the search/research still continues. Presented here are some useful linear problems/systems and their solutions. Inversion-free as well as inversion-based $O(mn^2)$ procedures that include computing least-squares and minimum-norm least-squares solutions are described. The former procedure provides us the information whether the system is consistent (contradiction-free) or not as well as the rank (physical information content) of A. Also described are (i) rectifying in $O(n^2)$ operations an already computed inverse of a matrix whose elements in a column were wrongly keyed in or needed to be modified, (ii) computing the Moore-Penrose inverse ($p-$inverse) of a matrix using optimal iterative schemes, and (iii) determining the $p-$inverse of the matrix obtained when a column is removed from the $p-$inverse of the original matrix. Included are a concise mathematically direct heuristic algorithm to solve a linear program (LP) and a method for testing optimality of a given solution of an LP. Inserted are several concerned Matlab programs for quick verification of the algorithms. Discussed are the possibilities of semi-numerical (numerical and symbolic) computation where division by zero/a too small number in an intermediate step could occur.

Keywords: Heuristic algorithm for linear programs, inversion-free algorithm, linear systems, Moore-Penrose inverse, optimal iterative schemes, semi-numerical computations.

1. Introduction

An information in a real-world problem is mathematically modeled as a linear or a nonlinear equation/inequality. For example, the physical information, viz., three mangos, four oranges, and six bananas cost Rs. 13 can be modeled as $3x_1 + 4x_2 + 6x_3 = 13$, where x_1, x_2, x_3 are the costs of one mango, one orange, and one banana, respectively. A man needs at least 75 gm of protein and 90 gm of fat daily. The protein and fat contents (in gm/unit) are respectively 7, 10 for cheese, 6, 0 for fish, 0, 28 for margarine. This information gives rise to the mathematical model, that yields the proteins and fats from the foregoing food items, $7x_1 + 6x_2 + 0x_3 \geq 75$, $10x_1 + 28x_3 \geq 90$, where x_1, x_2, x_3 are the number of units of cheese, that of fish, and that of margarine, respectively. Since we have well-developed theory of equations and we are more knowledgeable about properties of linear equations than those for linear inequalities, we often convert the inequalities into equations by using additional variables and then attempt to solve, i.e., compute the values of x_1, x_2, x_3.

A system of linear equations $Ax = b$, where the numerically known matrix A is $m \times n$, the numerically unknown column vector x is $n \times 1$ (to be computed), and the numerically known column vector b is $m \times 1$. Linear equations may have infinite number of solutions[1] or just one (unique) solution or no solution. For example, the equations $3x_1+4x_2+6x_3=13$, $2x_1+3x_2+7x_3=12$ will have infinite number of solutions. Two of these solutions (in rupees) are $[x_1\ x_2\ x_3]^t = [1\ 1\ 1]^t$ and $[x_1\ x_2\ x_3]^t = [2\ 1/10\ 11/10]^t$, where t denotes the transpose. It can be seen that a linear combination of these two solutions, viz., $\alpha_1\ [1\ 1\ 1]^t + \alpha_2[2\ \ 1/10\ \ 11/10]^t = [5/3\ \ 2/5\ \ 16/15]^t$, where $\alpha_1 = 1/3$ (an arbitrary value) and $\alpha_2 = 1 - \alpha_1 = 2/3$ is also a solution implying infinity of solutions. If we now consider the linear system $3x_1 + 4x_2 + 6x_3 = 13$, $2x_1 + 3x_2 + 7x_3 = 12$, $1x_1 + 5x_2 + 8x_3 = 14$, then we have the unique solution, viz., $[x_1\ x_2\ x_3]^t = [1\ 1\ 1]^t$. If we now consider the equations $3x_1 + 4x_2 + 6x_3 = 13$, $2x_1 + 3x_2 + 7x_3 = 12$, $5x_1 + 7x_2 + 13x_3 = 25$, then we will have infinity of solutions since the last equation is a linearly dependent (redundant) equation and adds no additional information

Corresponding author. Tel. +1 321 674 7714; fax: +1 321 674 7412. E-mail addresses: sksen@fit.edu (S.K. Sen), sagar.sen@cs.mcgill.ca (Sagar Sen)
Dr. S.K. Sen, Department of Mathematical Sciences, Florida Institute of Technology, 150 West University Boulevard, Melbourne,Florida 32901-6975, USA.
Mr. Sagar Sen, Modeling, Simulation and Design Laboratory, School of Computer Science, McGill University, McConnell Engineering Building, Room 202, 3480 University Street, Montreal, Quebec, Canada H3A 2A7

to the system of equations (knowledge of facts). We may and we should as well prune/weed out the last equation without any loss of information. However, if we consider the equations $3x_1 + 4x_2 + 6x_3 = 13$, $2x_1 + 3x_2 + 7x_3 = 12$, $5x_1 + 7x_2 + 13x_3 = 26$, then we will have no solution, i.e., we will never be able to find the values of x_1, x_2, x_3 so that all the three equations in the system are satisfied (simultaneously). If the first two equations are correct then the third (last) equation is definitely wrong in the same context, i.e., the information contradicts or, equivalently, is inconsistent. We must correct the wrong equation before we attempt a solution. One may view geometrically the system involving the hyperplanes of dimension[2] up to 2 and get a better feel about what is actually happening.

In all natural processes/computations, inconsistency/ contradiction is completely unknown. It could, however, happen not due to nature's fault[3] but due to human error/ mistake as well as due to limitation in devices for measuring quantities. We have not known any measuring device - electronic, optical, or non-electronic/non-optical - that can give an accuracy more than 0.005%. If the inconsistency (contradiction) is too pronounced then we should certainly check the model against the concerned physical problem and correct the errors/mistakes[4] before we attempt to solve the problem. On the other hand, consistency does not necessarily imply that the model is without mistake. A consistent system with mistakes will represent a model which is not the intended one. Hence one needs to check against such mistakes and eliminate them.

The subject involving numerous variations of linear systems that include linear optimization and large sparse/dense linear systems is too vast. So we limit ourselves to a few physically concise algorithms which are not so commonly seen in most textbooks (as of now) and which can be easily implemented in a Matlab programming language.[5]

In Sec. 2, we present the general solution of the linear system $Ax = b$, the related generalized matrix inverses (g-inverses), and optimal iterative $O(mn^2)$ algorithms for the p-inverse (also called the pseudo-inverse or the Moore-Penrose inverse or the minimum norm least squares inverse) [1-4] along with a relative error bound. The error tells us the quality of the result while the computational complexity $O(mn^2)$ provides us the computational cost in obtaining the result. In Sec. 3, we describe an inversion free algorithm that checks the correctness (consistency) of the linear system $Ax = b$, (could prune redundant rows (equations)), obtains rank of the matrix A, and finally a (minimum norm) solution vector with a relative error bound. By allowing $A := A^t A$, $b := A^t b$, the inversion-free algorithm provides us the least-squares solution since $A^t Ax = A^t b$ is always consistent irrespective of whether $Ax = b$ is consistent or not. Sec. 4 embodies the computation of B_c^{-1} from the matrices B and B^{-1}, where $B_c = B$ with c-th column changed. Also included in this section, a procedure to compute the p-inverse A_k^+ from the matrix A_{k+1} and its p-inverse A^+_{k+1}, where $A_k = A_{k+1}$ without c-th column. Sec. 5 includes a concise mathematically direct $O(n^3)$ heuristic algorithm to solve a linear program (LP) [9] and a method for testing optimality of a given solution of an LP [14]. Also included in this section, a discussion on the possibility of using semi-numerical computation when division by 0 in an intermediate step of a method could occur although the given numerical problem is well-posed with respect to several other methods. Sec. 6 comprises conclusions.

2. General solution of $Ax = b$ and optimal iterative schemes

General solution of $Ax = b$ A g-inverse $X = A^-$ of the given $m \times n$ matrix A is any matrix that satisfies the condition $AXA = A$. If the matrix A is nonsingular (necessarily implying $m = n$) then only $X = A^- = A^{-1}$ is unique, else there are infinity of A^-'s that will satisfy the condition $AXA = A$. The elements of A can be real or complex. The general form of the solution of the consistent system $Ax = b$ is $x = A^-b + (I - A^-A)z$, where I is the unit matrix of order n and z is an arbitrary n – column vector. For singular/(non-square) rectangular A, there will be infinity of solutions, each of which will satisfy the system $Ax = b$ if it is consistent (else none of the infinity of x's will satisfy the $Ax = b$). For the computation of A^-, use Gauss reduction type method or rank-augmented LU – algorithm [4. 5] or simply use one of following iterative algorithms [4, 6] that compute the p-inverse A^+ which is one of infinite possible A^-'s. If the vector b happens to be zero (homogeneous system $Ax = 0$) then the general form of the solution is $x = (I - A^- A)z$. Observe that $Ax = 0$ is ever consistent.

The minimum norm least squares inverse, i.e., p-inverse $X = A^+$ of the given $m \times n$ matrix is the matrix A that satisfies the conditions $AXA = A$, $XAX = X$, $(AX)^t = AX$, $(XA)^t = XA$ and is unique. The p-inverse A^+ gives the minimum norm least squares solution (unique) $x = A^+b$ of the system $Ax = b$, where both the norms[6] $\|x\|$, $\|Ax - b\|$ are the smallest. Observe that $b \neq 0$ (null column vector); for if $b = 0$ then the system is ever consistent. Here the system $Ax = b$ may be consistent or inconsistent. Observe that if $Ax = b$ is consistent then $\|Ax - b\|$ will be 0 else it will never be 0, i.e., the minimum norm least squares solution will never satisfy $Ax = b$ if it is inconsistent (implying $b \neq 0$) but the solution will only satisfy the two conditions. $\|x\| =$ *smallest (smallest norn condition)*, $\|Ax - b\| =$ *smallest (least – squares condition)*. This solution is extremely important in solving numerous real world problems. If the matrix A is nonsingular then $A^+ = A^{-1}$ always. A^+ always exists for any matrix A while A^{-1} only exists for nonsingular matrix A.

Iterative schemes for the p-inverse A^+ We provide here just two schemes, viz., the quadratic and the cubic schemes. The quadratic/cubic schemes have been shown to be computationally most economical [4, 6] for the $m \times n$ matrix A. It can be seen that the best accuracy subject to the precision of computation is obtainable in these schemes unlike a mathematically non-iterative (direct) method. Let tr denote trace. The trace of the square matrix X is the sum of its diagonal elements. If

$$X = \begin{bmatrix} 2 & 7 \\ 4 & -3 \end{bmatrix}$$

then $tr\ X = 2 + (-3) = -1$. Also let I be an $\cancel{m} \times n$ unit matrix.

The quadratic scheme

Step 1. Compute $X_0 = \dfrac{A^t}{tr(AA^t)}$.

Note One may compute $tr(A^tA)$ instead of $tr(AA^t)$ as both are same. However, if $m > n$ then compute preferably $tr(A^tA)$ since the dimension of A^tA will be smaller. Observe that the two matrices AA^t, A^tA are both symmetric and diagonal elements are all nonnegative (A has real elements).

Step 2. Compute $X_{k+1} = X_k(2I - AX_k)$

for $k = 0$, 1, 2, $\cancel{A}$.... till $\dfrac{\|X_{k+1} - X_k\|}{\|X_{k+1}\|} \le 0.5 \times 10^{-4}$.

Note The matrix X_{k+1} upon satisfaction of the foregoing inequality will be A^+ correct up to 4 significant digits.

The cubic scheme is exactly the same as the quadratic scheme except that

$X_{k+1} = X_k(3I - AX_k(3I - AX_k))$

instead of $X_{k+1} = X_k(2I - AX_k)$ in Step 2.

Both the foregoing fixed-point schemes will always converge for any matrix - real or complex, square or rectangular, singular or nonsingular. For other higher order schemes as well as the linear scheme, refer [4, 6]. So the number of iterations may be kept sufficiently large, say, 1000. The accuracy desired will automatically decide/determine the number of iterations used (see the following Matlab programs).

A Matlab program for the quadratic scheme is

```
[m,n]=size(A); I=eye(m);
X=A'/trace(A*A'); for k=1:1000, X1=X*(2*I-A*X);
if norm(X1-X)/norm(X1)<=0.5*10^-4,break;else X=X1, end;
end;
'No. of iterations in quadratic scheme is', k,
'The p-inverse of the matrix A is', X1,
```

Save the program (M file) in pinverseq and then type the following command in Matlab command window.

```
A=[2 7;4 -3];pinverseq
```

if the p-inverse of

$$A = \begin{bmatrix} 2 & 7 \\ 4 & -3 \end{bmatrix}$$

is desired. Observe that since the matrix A is nonsingular, its p-inverse will be the true inverse correct up to four significant digits. The Matlab produces the number of iterations k = 7 and the p-inverse of the matrix A as

```
X1=  0.0882     0.2059
     0.1176    -0.0588
```

The default option (format short) for Matlab is printing/ outputting only four decimal places although the computation is carried out with at least 14 digits. The only other option is format long which needs to be entered at the Matlab command window before executing the program

```
A=[2 7;4 -3]; pinverseq
```

That is, if we enter

```
format long; A=[2 7;4 -3]; pinverseq
```

then Matlab will produce 14 decimal digits for each element of the p-inverse.

Matlab program for the cubic scheme is

```
[m,n]=size(A);I=eye(m);
X=A'/trace(A*A');for k=1:1000, X1=X*(3*I-A*X*(3*I-A*X));
if norm(X1-X)/norm(X1)<=0.5*10^-4,break; else X=X1, end; end;
'No. of iterations in cubic scheme is',k,
'The p-inverse of the matrix A is', X1,
```

Save the program in pinversec and then type the command

```
format long; A=[2 7;4 -3];pinversec
```

in the Matlab command window. This will produce the number of iterations k=5 and the p-inverse A^+= X1, where

```
X1 =  0.08823529411765    0.20588235294118
      0.11764705882353   -0.05882352941176
```

If the right-hand side vector $b = [9\ \ 1]^t$ then the solution of the system $Ax = b$ is given by $x = A^+b$. The Matlab commands (typed in the command window) and the output are

```
format long;
Ap= [0.08823529411765    0.20588235294118;
     0.11764705882353   -0.05882352941176];
     b=[9  1]';x=Ap*b
```

and

```
x = 1.00000000000003
    1.00000000000001
```

If $b = 0$ *(null column vector)* then the solution will be $x = 0$ *(null column vector)*.

If, in the singular consistent system $Ax = b$,

$$A = \begin{bmatrix} 2 & 7 \\ 4 & 14 \end{bmatrix}, \quad b = \begin{bmatrix} 9 \\ 18 \end{bmatrix}$$

then no. of iterations in quadratic scheme is k = 1

The p-inverse of the matrix A is A^+ = X1, where

X1 = 0.00754716981132 0.01509433962264
0.02641509433962 0.05283018867925

and the minimum norm least-squares solution vector is

x = 0.33962264150943
1.18867924528302

It can be easily verified that the equation $Ax = b$ is satisfied. There is an infinity of solutions. Another solution is $x = [1 \;\; 1]^t$ which is clearly not the minimum norm (i.e., $|x| = $ *smallest*) solution.

If, on the other hand, we have the inconsistent system $Ax = b$,

$$A = \begin{bmatrix} 2 & 7 \\ 4 & 14 \end{bmatrix}, \quad b = \begin{bmatrix} 9 \\ 17 \end{bmatrix}$$

and issue the Matlab commands A=[2 7; 4 14]; b=[9 17]'; pinverseq; x=X1*b then no. of iterations in quadratic scheme is k = 1

The p-inverse of the matrix A is the same as the foregoing X1 and the minimum-norm least-squares solution x = $[0.32452830188679 \;\; 1.13584905660377]^t$. This solution x as well as any other vector will never satisfy the foregoing inconsistent (contradictory) system. If we compute Ax, it will be $[8.60000000000000 \;\; 17.20000000000000]^t$ instead of $[9 \;\; 17]^t$.

3. Inversion-free algorithms for $Ax = b$

(a) *Mathematical C-LINSOLVER* The $O(mn^2)$ physically concise algorithm MATHEMATICAL C-LINSOLVER [7, 8, 9, 10] for consistent linear system $Ax = b$ is as follows. Let a_i^t be the i - th row of A. Then a_i is the column vector, i.e., the i – th row (of A) written as the column. In-built in the algorithm is obtained without explicitly computing the minimum-norm least-squares inverse A^+ as well as the rank r of A.

(* MATHEMATICAL C-LINSOLVER *)

P = I; x = 0; r = 0; *for* i := 1 *to* m *do* *begin* EQN (a_i, b_i, P, x); r := r + c *end*

procedure EQN(a, b, P,x); (* solves one equation $a^t x = b$ *)

begin c :=0; u := Pa; v := $\|u\|^2$; s :=b - $a^t x$;

if v ≠ 0 *then* *begin* P :=P - uu^t/v; x:=x + su/v; c :=1 *end else if* s ≠ 0 *then* Ax = b is inconsistent (contradictory) and terminate *end*;

Computational C-LINSOLVER Since the numerical zero in a floating-point arithmetic is not the same as the mathematical zero [13], replace, in MATHEMATICAL C-LINSOLVER, v ≠ 0 and s ≠ 0 by $v \ge 0.5\times10^{-4}\bar{a}$, $s \ge 0.5\times10^{-4}\bar{b}$ respectively to obtain COMPUTATIONAL C-LINSOLVER for four significant digit accuracy, where

$$\bar{a} = \left(\sum_{i=1}^{m}\sum_{j=1}^{m}|a_{ij}|\right)/(mn), \quad \bar{b} = \sum_{i=1}^{m}|b_i|/m$$

(b) *The mathematical NC-LINSOLVER* It is the concise linear system solver for near-consistent system. If the system is too inconsistent, then it is necessary to reexamine the physical model and the derived mathematical model to find out the real cause for excessive inconsistency (contradiction) and rectify the errors/mistakes. It is not advisable to proceed solving (in the least squares sense) such highly inconsistent systems since the resulting (least-squares or minimum norm least-squares) solution may convey a wrong message to the physical world. We present a mathematical version of the NC-LINSOLVER as well as its computational version with a Matlab program for ready use (just by copying and pasting).

It is a modified version of C-LINSOLVER, which provides a solution of the consistent system closest (in the sense of the minimum variation/modification of the right-hand side vector b componentwise as needed by the modified version) to the given near-consistent/inconsistent system Ax = b along with a relative error-bound of the solution for the inconsistent system. On completion of the execution of NC-LINSOLVER, we obtain a solution x for the consistent system Ax = b + Δb, the projection operator (matrix) P = I - A^+A that provides a solution Pz (where z is an arbitrary (null or non-null) vector) of the homogeneous linear system Ax = 0. Further, we obtain the rank r of A and Δb of the modification of b. $\Delta b_i \neq 0$, for, equivalently, $\Delta b_i \neq 0$ for some i tells us that the given Ax = b is inconsistent. The inconsistency index inci = $\|\Delta b\| / \|A, b\|$ as well as a relative error err = $\|b - Ax\| / \|x\|$ of the solution x are also produced. ‖ A, b ‖ denotes a norm (e.g., the Euclidean norm) of the augmented matrix (A, b).

(* MATHEMATICAL NC-LINSOLVER*)

P := I; x := 0; Δb := 0; r := 0;

(* abar = $\bar{a} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|\right)/(mn)$,

bbar = $\bar{b} = \sum_{i=1}^{m}|b_i|/m$ *)

(* abar and bbar are exactly the same as mentioned and not needed in this version.*)

for i := 1 *to* m *do*

begin

u := Pa_i; v := $\|u\|^2$; s := b_i - $a_i^t x$; c :=0; *if* v=0 and s ≠ 0

then begin print ‘Ax=b is strictly inconsistent’;

Δb_i:=-s; b_i:=b_i+Δb_i; s :=0; *end else*

if v=0 *and* s=0 *then* Δb_i:=0;

if v≠0 *then begin* x := x + us/v; P := P - uu^t/v; c := 1; Δb_i := 0 *end*; r := r + c

end;

inci:= || Δb |/ | A,b ||; err:= || b - Ax |/ || x | |

print A, b, **Δ**b, x, P, r, inci, err;

The computational NC-LINSOLVER The following modifications in the foregoing mathematical NC-LINSOLVER give us the computational version.

- Remove (* and *) from the very first comment so that abar and bbar are computed.
- Replace v=0 by v ≤ 0.5 × 10^{-4}abar, v ≠ 0 by v≥0.5 × 10^{-4}abar, s = 0 by | s | ≤ 0.5 × 10^{-4}bbar, and s ≠ 0 by | s | ≥ 0.5 × 10^{-4}bbar

Computational NC-LINSOLVER (*Matlab program*) The following program is self-explanatory.

```
function[ ] = nclinsolver1(A,b); [m, n] = size(A);
%NC-LINSOLVER: Near-consistent Linear System Solver
'The matrix A and vector b of the system Ax=b are', A,b,
P = eye(n); sd = 0; x(1:n) = 0; x = x'; delb(1:m) = 0; delb = delb'; bo = b; r = 0;
abar = 0; for i = 1:m, for j = 1:n, abar = abar+abs(A(i,j)); end; end; abar = abar/(m*n);
bbar = 0; for i = 1:m, bbar = bbar+abs(b(i)); end; bbar = bbar/m;
for i = 1:m
u = P*A(i,:)'; v = norm(u)^2; s = b(i) -A(i,:)*x; c = 0;
if v< = .00005*abar & abs(s) > = .00005*bbar, delb(i) = -s; sd = -s; b(i) = b(i) +delb(i); s = 0;
elseif v< = .00005*abar & abs(s)< = .00005*bbar; delb(i) = 0; end;
if v> = .00005*abar, x = x+u*s/v; P = P-u*u'/v; c = 1; delb(i) = 0; end; r = r+c;
end;
if abs(sd)>.00005*(abar+bbar)*0.5, 'The system Ax = b is inconsistent.', end;
inci = norm(delb)/norm([A,b]); err = norm(bo-A*x)/norm(x);
'The projection operator P = (I - A+A) is', P,
'The rank of the matrix A is', r,
'The inconsistency index is', inci,
'Modification in vector b, i.e., Db is', delb,
'Vector b of the nearest consistent system is', b,
'Solution vector of the nearest consistent system is', x,
'Error in the solution vector x is', err
```

Issuing the Matlab command

>> A = [1 2 3;4 5 6]; b = [6 15]'; nclinsolver1(A,**b)**

we obtain the following result. The matrix A and vector b of the system $Ax=b$ as well as the projection operator $P=I-A^+A$ are

$$A=\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix},\ b=\begin{bmatrix}6\\15\end{bmatrix}.$$

$$P=\begin{bmatrix}0.1667&-0.3333&0.1667\\-0.3333&0.6667&-0.3333\\0.1667&-0.3333&0.1667\end{bmatrix}$$

The rank of the matrix A is r = 2.

The inconsistency index is inci = 0.

Modification in vector *b*, i.e., Db is *delb* = $[0\ \ 0]^t$.

Vector *b* of the nearest consistent system is $b = [6\ \ 15]^t$.

Solution vector of the nearest consistent system is

$$x = [1.0000\quad 1.0000\quad 1.0000]^t.$$

Error in the solution vector *x* is err = 1.0256e-015.

If we now issue the Matlab command

>> A = [1 2 3;4 5 6; 7 8 9]; b = [6 15 25]'; nclinsolver1(A,b)

then we get the following solution.

The matrix A and vector b of the system $Ax = b$ are

$$A=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix},\ b=\begin{bmatrix}6\\15\\25\end{bmatrix}$$

The system $Ax = b$ is inconsistent.

The projection operator $P = I - A^+A$ is the same as the foregoing P.

The rank of the matrix A is r = 2

The inconsistency index is inci = 0.0299

Modification in vector *b*, i.e., Db is *delb* = $[0\ \ 0\ \ -1]^t$

Vector of the nearest consistent system is $b = [6\ \ 15\ \ 24]^t$

Solution vector of the nearest consistent system is $x = [1\ \ 1\ \ 1]^t$.

Error in the solution vector x is err = 0.5774.

To obtain a least squares solution of an inconsistent system using the NC-LINSOLVER just allow $A := A^tA$, $b := A^tb$, the inversion-free algorithm provides us the least-squares solution since $A^tAx = A^tb$ is always consistent irrespective of whether $Ax = b$ is consistent or not. However, too inconsistent (contradictory) system needs to be reexamined for possible mistakes and corrected before using the solver. We will certainly get correct least-squares solution, but such a solution may not be useful for a real world problem.

4. Inverse of a column-modified/column-omitted matrix in $O(n^2)$ operations

Column-modified matrix inverse We present here a concise $O(n^2)$ algorithm [4, 14] to compute B_c^{-1} from the given nonsingular matrix B and B^{-1}, where $B_c = B$ with c-th column changed. Let b_c = c-th column of B_c and the vector $e_i = [0 \;\; 0 \ldots 1 \ldots 0]^t$, where e_i has all the elements 0 except the i-th element which is 1.

Step 1 Compute $\alpha^c = B^{-1}b_c, \xi = \left[-\alpha_1^c/\alpha_c^c \;\; -\alpha_2^c/\alpha_c^c \ldots -\alpha_n^c/\alpha_c^c\right]^t$.

Step 2 Compute $B_c^{-1} = \left[e_1 \;\; e_2 \ldots \xi \ldots e_n\right]B^{-1}$.

The algorithm needs n^2 additions and n^2 multiplications. Having computed the inverse of the matrix B (needing $O(n^3)$ operations), if we discover that some element(s) in a column of B have been wrongly entered or need to be changed then without recomputing the inverse of the changed matrix we can compute the inverse using much less number of operations. For example, if n=1000 then we would be needing $O(10^6)$ operations instead of $O(10^9)$ operations. This is roughly 1000 times less computations.

The Matlab program The *columnmodifiedinverse* Matlab program is as follows.

```
%c = column of B that has been changed; bc = c-th column of Bc.
n = size(B); B(:, c) = bc; Bc=B; alpc = inv(B)*bc; zi = -alpc./alpc(c);
I = eye(n); I(:, c) = zi; Bcinv = I*inv(B),
```

In the Matlab command window, if we write the command

>>B = [1 2 3; 4 5 6; 7 8 8], bc = [3 5 10]', c = 3, columnmodifiedinverse

then we get

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{bmatrix}, bc = \begin{bmatrix} 3 \\ 5 \\ 10 \end{bmatrix}, \; c = 3$$

$$Bcinv = \begin{bmatrix} -1.1111 & -0.4444 & 0.5556 \\ 0.5556 & 1.2222 & -0.7778 \\ -0.3333 & 0.6667 & -0.3333 \end{bmatrix}$$

Column-omitted matrix p-inverse. Yet another concise $O(n^2)$ algorithm [4] to compute A_k^+ from known A_{k+1}, A^+_{k+1} is as follows. Here the matrix A_{k+1} is any matrix (rectangular, square, singular, or non-singular). Let $A_k = A_{k+1}$ without c-th column, A^+_{k+1-c} = A^+_{k+1} without c-th row, a_c = c-th column of A_{k+1}, and b_c^t = c-th row of A^+_{k+1}. Then

Step 1 Compute the scalar $r = 1 - b_c^t a_c$ and then

$$A_k^+ = A_{k+1-c}^+ + \frac{1}{r}(A_{k+1-c}^+ a_c)b_c^t$$

The Matlab program for the foregoing algorithm is, naming A_k as A, A_{k+1} as $A1$, A_k^+ as Ap, A^+_{k+1}, as $A1p$,

```
[m, n]=size(A1); A1p=pinv(A1);
% A1, A1p, c are assumed to have been supplied, although A1p is computed.
A=A1(:,1:c-1); A(:,c:n-1)=A1(:,c+1:n),
A1mcp(1:c-1,:)=A1p(1:c-1,:);
A1mcp(c:n-1,:)=A1p(c+1:n,:);
ac=A1(:, c), bct=A1p(c,:), r=1-bct*ac,
Ap=A1mcp+(1/r)*(A1mcp*ac)*bct,
```

Issuing the Matlab command

c=2; A1=[1 2 3 4;5 6 7 8;9 10 11 12], columnomittedpinverse

we get

$$A1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, A = \begin{bmatrix} 1 & 3 & 4 \\ 5 & 7 & 8 \\ 9 & 11 & 12 \end{bmatrix}, ac = \begin{bmatrix} 2 \\ 6 \\ 10 \end{bmatrix},$$

$$bct = [-0.1458 \;\; -0.0333 \;\; 0.0792], \; r = .7000$$

$$Ap = \begin{bmatrix} -0.4583 & -0.1190 & 0.2202 \\ 0.0417 & 0.0238 & 0.0060 \\ 0.2917 & 0.0952 & -0.1012 \end{bmatrix}$$

5. Direct heuristic algorithm for *lp* and optimality test of a given solution of *lp*

A concise mathematically direct heuristic algorithm [9] to solve the linear program (*lp*) $Min\; c^t x$ subject to $Ax = b$, $x \geq 0$ (null column vector), where A is an $m \times n$ matrix, b is an m column vector, is as follows. Let I be the unit matrix of order n and A^+ the Moore-Penrose inverse.

Step 1 Compute $H = A^+A$, $d = A^+b$, $c' = (I - H)c$,

$$s_k = \min\left\{\frac{d_i}{c'_i} : c'_i > 0\right\}.$$

Step 2 Compute $x = d - (I - H)c \times s_k$

Step 3 Remove that x_i which becomes 0, corresponding column of A, and element of c and repeat Steps 1 and 2 until no $c'_i > 0$. (Keep track of the positive elements of x using an index set).

A Matlab program named as lpsolverheuristic1 is as follows

```
%A=[-3 3 1 0;2 4 0 1], b=[6 12]', c=[4 -5 0 0]', %To be supplied
[m, n]=size(A); is=1:n; cd=1;
while sum(abs(cd))>0.5*10^-8,
I=eye(n); Ap=pinv(A); H=Ap*A, d=Ap*b, cd=(I-H)*c,
for i=1:n,
if cd(i)>0, s(i)=d(i)/cd(i);else s(i)=500;%large value for s(i)
end; end; s1=s(1:n), [sk, k]=min(s1),
if sk==500,'Soln vector x=',x, 'Index set is=', is,
'Index set specifies which elements of x have the values in x-vector',
else
x=d-(I-H)*c*sk, is0=is(1:k-1); is0(k:n-1)=is(k+1:n); is=is0,
```

```
A0=A(:,1:k-1); A0(:,k:n-1)=A(:,k+1:n); A=A0,
c0=c(1:k-1); c0(k:n-1)=c(k+1:n);c=c0, n=n-1;
end; end;
```

If we now issue the command

```
>> A = [-3 3 1 0;2 4 0 1], b = [6 12]', c = [4 -5 0 0]',
lpsolverheuristic1
```

we get the solution of the LP $Min\ c^t x$ subject to $Ax = b, x \geq 0$, where

$$A = \begin{bmatrix} -3 & 3 & 1 & 0 \\ 2 & 4 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 6 \\ 12 \end{bmatrix}, c = \begin{bmatrix} 4 & -5 & 0 & 0 \end{bmatrix}^t$$

as $x = [x_1 \quad x_2]^t = [0.6667 \quad 2.6667]^t$.

This noniterative polynomial-time heuristic algorithm is useful in many *lps* since even if it fails to produce the optimal solution, it gives one close to it and thus can be a good starting feasible solution. Practically all the interior-point as well as exterior-point methods for linear optimization start the solution procedure from a known feasible solution. The optimal objective function value produced by this heuristic solution is either close to the actual optimal value of the objective function or often the actual optimal value of the objective function.

Testing optimality of a given solution of an LP Let the linear program (*lp*) be $Min\ c^t x$ subject to $Ax = b, x \geq 0$. Let $p_j = j$ – th non-basic vector in A. Also, let B be the basis. Then [14]

Step 1 Compute $y^t = c^t_B B^{-1}$ *(row vector)*, $z_j - c_j = y^t p_j - c_j$ *(scalar)*

Step 2 If all $z_j - c_j \leq 0$ then the solution is optimal else see pp. 254-55 of [14].

It is rather easy to check whether a given solution of a linear system is correct or not. It is, however, not so obvious to check whether a given solution of an *lp* is correct or not.

Semi-numerical computation We have rarely thought about the possibility of non-numerical (symbolic) computation mixed with the numerical ones when we proceed solving a pure numerical problem. It is interesting to note that sometimes symbolic computations embedded inside numerical computations are capable of obviating the problem of division by 0 and thus could be useful. For example, we know that the numerical LU decomposition of a non-singular numerical matrix will fail when a leading/trailing minor (whose order is at least 1 less than that of the matrix) vanishes. Under these circumstances, one can use rank-augmented LU-decomposition [5] which is completely numerical. Alternatively, one may use a symbol x and carry out semi-numerical (a combination of numerical and non-numerical) computation and then allow $x \to 0$ (in the limit). In any other situation/method, similar computational procedure could be devised. However, in most programming languages such as the Fortran meant for numerical computation, such a non-numerical computation may be cumbersome (more involved in the implementation aspects) and thus may not be very attractive.

6. Conclusions

We have just provided here a few physically concise algorithms for linear systems with their computational complexities for ready use by a reader/researcher who has Matlab software. The proofs are omitted either because these are available in the cited literature or are not too hard. The NC-LINSOLVER in Sec. 3 can be easily modified to include pruning of linearly dependent rows of the matrix as these rows do not contribute to the information content of the linear system. This pruning can be just an integral part of the NC-LINSOLVER. It will reduce the size of the augmented matrix (A, b) of the system $Ax = b$ to a significant extent. Consequently we would need less storage space as well as significantly less amount of computation resulting in less error. There are many physical problems whose mathematical models are partial/ordinary differential equations. Many of these models involve large linear systems often in a specific structure such as the tri-diagonal form and a sparse form. Such a structure can be exploited by appropriately modifying the foregoing algorithms so that the storage as well as computations are highly reduced. Besides, a parallel implementation of the algorithms is not hard. Taking the structural advantage and implementing the algorithms on a parallel machine, are however, significant innovative programming as well as mathematical activities. One may pursue such activities producing excellent numerical algorithms that can solve truly large problems with competitive quality of the results and computational complexity.

End Notes

1 By 'solution' we mean the values of the variables (elements) of the unknown solution vector when substituted in all the given equations will satisfy all the equations.

2 We, the human beings, can visualize things up to three dimensions and not beyond three although to solve real-world problems we need to consider very high dimensional hyperspace (space bounded by hyperplanes). One dimensional hyperplane is a straight line starting from $-\infty$ going to ∞. Zero dimensional hyperplane is a point while two dimensional hyperplane is a plane extended from $-\infty$ to ∞, and so on. It is interesting to note that we can think of two mutually perpendicular straight lines as in Euclidean geometry. We can also imagine three mutually perpendicular lines. Can we imagine four mutually perpendicular lines? The answer is 'no' for a common human being. It appears that an ant has maximum two dimensional sense. If a sugar granule is dropped in front of the crawling ant, it would appear to it a miracle as the granule comes from above (through the third dimension). Nothing was there in front of it while all of a sudden the granule appeared!

3 Nature knows no fault. Nature follows all natural laws (known to us or not) perfectly and there exists no disorder (chaos). It is only our limitation in our capabilities/knowledge, we have developed the theory of chaos just to get some kind of forecast/solution of the extremely highly sensitive problems. Slightest change in initial conditions in these problems may cause enormous change in the forecast/solution. Consequently, the probability of forecast (for example, the forecast of the path of a cyclone/hurricane) to be correct is often very low.

4 ~~While the~~ age-old proverb, viz., *To err is human* was valid in the whole of past, is valid today, and will remain valid over the whole of future. Here *err* means mistake and *human* implies all living beings including human beings. It is impossible to find a living being who can say that he/she never commits a mistake. Another relevant proverb is *Not to err is computer*. Here *computer* implies modern computer which is nonliving.

5 Matlab is a very high level user-friendly programming language meant for scientists and engineers. Its usage needs no formal programming knowledge.

6 Out of several possible norms [4] we use/compute the Euclidean norm. The Euclidean norm of the matrix

$$A = [a_{ij}]is \ \|A\| = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a^2 ij} \; .$$

Similarly, the Euclidean norm of the vector x is $\|x\| =$.

References

1. V. Lakshmikantham, S.K. Sen, G.W. Howell, Vectors versus matrices: p-inversion, cryptographic applications, and vector implementation, Neural Parallel Sci. Comput. 4 (1996) 129–140.
2. C.R. Rao, S.K. Mitra, Generalized Inverse of Matrices and Its Application, Wiley, New York, 1971.
3. G. Golub, W. Kahan, Calculating the singular values and the pseudo-inverse of a matrix, SIAM J. Numer. Anal. B-2 (1965) 205–224.
4. E.V. Krishnamurthy, S.K. Sen, Numerical Algorithms: Computations in Science and Engineering, Affiliated East-West Press, New Delhi, 2001.
5. S.K. Sen, E.V. Krishnamurthy, Rank-augmented LU-algorithm for computing generalized matrix inverses, IEEE Trans. Comput. C-23 (1974), 199-201.
6. S.K. Sen, S.S. Prabhu, Optimal iterative schemes for computing Moore–Penrose matrix inverse, Internat. J. Systems Sci. 8 (1976) 748–753.
7. E.A. Lord, V.Ch. Venkaiah, S.K. Sen, A concise algorithm to solve under-/over-determined linear systems, Simulation 54 (1990) 239–240.
8. E.A. Lord, V.Ch. Venkaiah, S.K. Sen, A shrinking polytope method for linear programming, Neural Parallel Sci. Comput. 4 (1996) 325–340.
9. V. Lakshmikantham, S.K. Sen, M.K. Jain, A. Ramful, O(n^3) noniterative heuristic algorithm for linear programs with error-free implementation, Applied Mathematics and Computation 110(2000) 53-81.
10. S.K. Sen, H. Agarwal, Sagar Sen, Chemical equation balancing: An integer programming approach, Mathematical and Computer Modelling, 44(2006) 678-691.
11. E.H. Moore, On the reciprocal of the general algebraic matrix (abs.), Bull. Amer. Math. Soc. 26 (1920) 394–395.
12. R. Penrose, A generalized inverse for matrices, Proc. Chemb. Phil. Soc. 51 (1955) 406–413.
13. V. Lakshmikantham, S.K. Sen, Computational Error and Complexity in Science and Engineering, Elsevier, Amsterdam, 2005.
14. H.A. Taha, Operations Research: An Introduction, Macmillan, New York, 1989.