J. Appl. Math. & Informatics Vol. **27***(2009), No. 3 - 4, pp. 621 - 638*
Website: http://www.kcam.biz

ILL- VERSUS WELL-POSED SINGULAR LINEAR SYSTEMS: SCOPE OF RANDOMIZED ALGORITHMS

S.K. SEN*, RAVI P. AGARWAL, AND GHOLAM ALI SHAYKHIAN

ABSTRACT. The linear system $Ax = b$ will have (i) no solution, (ii) only one non-trivial (trivial) solution, or (iii) infinity of solutions. Our focus will be on cases (ii) and (iii). The mathematical models of many real-world problems give rise to (a) ill-conditioned linear systems, (b) singular linear systems (A is singular with all its linearly independent rows are sufficiently linearly independent), or (c) ill-conditioned singular linear systems (A is singular with some or all of its strictly linearly independent rows are near-linearly dependent). This article highlights the scope and need of a randomized algorithm for ill-conditioned/singular systems when a reasonably narrow domain of a solution vector is specified. Further, it stresses that with the increasing computing power, the importance of randomized algorithms is also increasing. It also points out that, for many optimization linear/nonlinear problems, randomized algorithms are increasingly dominating the deterministic approaches and, for some problems such as the traveling salesman problem, randomized algorithms are the only alternatives.

AMS Mathematics Subject Classification: 15A06, 15A09, 65F05, 65F22
Keywords and phrases : Ill-conditioned singular linear system, near-linearly dependent rows, pseudo-inverse, randomized algorithm, specified domain.

1. Introduction

Let the consistent linear system be $Ax = b$, where A is an mxn numerically known matrix and b is an $mx1$ numerically known vector. The system will be considered ill-conditioned if two or more rows of A are nearly linearly dependent. If all those rows of A which are sufficiently linearly independent while its other rows (one or more) are perfectly linearly dependent, then A will be considered well-conditioned but singular. This second case does not pose numerical instability while the first one may indeed induce considerable numerical

Received April 11, 2008. Accepted October 26, 2008. *Corresponding author.

error in computation. We will demonstrate this fact while considering numerical examples in section 2. Several physically concise algorithms have been recently developed [1-3]. All these algorithms are deterministic. However, some are iterative while others are non-iterative. Instead of these algorithms, we will demonstrate through the popular Matlab command pinv that an exactly singular linear system is not ill-conditioned as long as the system does not have near-linearly dependent rows. In fact, if a perfectly singular system has all its linearly independent rows sufficiently linearly independent then the singular system is well-posed/well-conditioned. No numerical instability is faced while solving the system. There are many possible randomized algorithms and random sequence generators mainly meant for optimization [4-39]. Most of these algorithms can be employed to solve the much simpler (rather relaxed) non-optimization linear problems. In section 2, we will discuss/demonstrate, for a specified precision, the important difference of well-posed singular systems and ill-posed singular systems with respect to the pseudo (Moore-Penrose or, equivalently, minimum norm least-squares) inverse with the help of simple numerical examples. In section 3, we will present a straightforward randomized algorithm for solving a singular well-conditioned linear system, a singular ill-conditioned linear system, as well as a near-singular (and strictly non-singular) linear system when a narrow bound for the solution vector is specified. We will stress in this section that deterministic algorithms, specifically non-iterative ones, fail to produce the desired solution vector. Section 4 is an exposition of the increasing importance of randomized algorithms with increasing computing power while section 5 comprises conclusions.

2. Ill- vesrus well-posed singular systems

It is well-known that for a singular consistent system $Ax = b$, true inverse of A does not exist and hence no unique solution is possible. However, the minimum norm least-squares solution of the system $Ax = b$ does always exist so that $||x||$= minimum and also $||Ax - b||$ = minimum (=0 for consistent system but > 0 for inconsistent system). Is this ever existent solution free from numerical instability? The answer is "no" if the singular system $Ax = b$ is ill-conditioned. That is, some of the rows of A are near-linearly dependent. The term "near-linearly" is precision (word length of the computer) dependent and is viewed based on our accuracy requirement/specification. Thus in a given context, we may compare any two systems ranking one more ill-posed than the other. In fact, just computing a generalized inverse (g-inverse) A^- or the minimum norm least-squares (mt-) inverse A^+ (also called the Moore-Penrose inverse or, simply, the pseudo-inverse) which always exists for both ill-posed singular matrices as well as and well-posed singular matrices will introduce considerable error in the solution vector $x = A^-b$ or $x = A^+b$ if the singular (consistent) system is ill-posed. Otherwise, i.e., if the singular system is well-posed, such error will not be introduced in the solution vector. We like to stress the fact that not all consistent singular systems are the same. Or, equivalently, not all singular

matrices are the same with respect to the mt-inverse or a g-inverse. An ill-posed singular matrix will have more error in its mt-inverse while a well-posed singular matrix will have less or no error in its mt-inverse within the limit of the precision of the computer/software used. Such a distinction between an ill-posed and a well-posed singular matrix/system is necessary when we talk about numerical sensitivity/instability. For, example, if the exact singular matrix or, equivalently, perfectly well-posed singular matrix A is

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix},$$

then Matlab outputs its mt-inverse as

$$A^{+} = \begin{bmatrix} 0.04 & 0.08 \\ 0.08 & 0.16 \end{bmatrix}$$

which is exact. It can be easily verified that

$$AA^{+}A = A, A^{+}AA^{+} = A^{+}, (AA^{+})^{t} = AA^{+}, (A^{+}A)^{t} = A^{+}A$$

exactly. On the other hand, if we consider the almost exact (with respect to precision) singular matrix, also called ill-posed singular matrix, B , where

$$B = \begin{bmatrix} 1 & 2 \\ 2 & 4 - 10^{-5} \end{bmatrix},$$

the exact mt-inverse B^{+} will be

$$B^{+} = \begin{bmatrix} -399999 & +200000 \\ +200000 & -100000 \end{bmatrix}.$$

Observe that the exact B^{+} is the exact B^{-1} since the almost exact singular matrix B is strictly non-singular.

Matlab outputs its mt-inverse as

$$B_{m}^{+} = \begin{bmatrix} -399999.0000493783 & +200000.0000246893 \\ +200000.0000246892 & -100000.0000123447 \end{bmatrix}.$$

which is evidently in error. Similar errors will be introduced in higher order singular systems whose two or more rows are near-linearly dependent. If we now consider the exactly singular (symmetric) matrix A of order 3, where

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix},$$

then Matlab produces the minimum-norm least-squares (mt-) inverse A^{+}, where

$$A^{+} = \begin{bmatrix} 0.00510204081632653 & 0.0102040816326531 & 0.0153061224489796 \\ 0.0102040816326531 & 0.0204081632653061 & 0.03061224489799592 \\ 0.0153061224489796 & 0.0306122448979592 & 0.0459183673469388 \end{bmatrix}.$$

The consequent relation $AA^+A = A$ can be seen to be exactly satisfied in Matlab up to at least 14 significant digits implying excellent numerical stability (practically no error introduced). If the matrix B is

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4-10^{-5} & 6 \\ 3 & 6 & 9 \end{bmatrix},$$

then

$$B^+ = 10^4 \begin{bmatrix} 1.999999998217689 & -0.307690768969812 & -0.461536153426023 \\ -9.999999999677787 & 1.538461538478187 & 2.307692307573808 \\ 6.000000000379296 & -0.923072307790393 & -1.384608461599505 \end{bmatrix}.$$

The consequent relation $BB^+B = B$ can be seen to be satisfied only up to 10 decimal places implying reduced numerical stability (more error introduced). We demonstrate this sensitivity by taking the following simple numerical linear systems and considering the popular Matlab command *pinv*.

Example 1. Let the given consistent strictly nonsingular system be $Ax = b$, where

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4-10^{-p} \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 6-10^{-p} \end{bmatrix}, \tag{1}$$

p being a finite positive integer. It can be seen that the exact solution of the system is $x = [11]^t$ where t denotes the transpose. If $p = 14$, then the system may be termed near-singular when the precision used is 15 digits. The standard precision with which Matlab works is 15 significant digits. The Matlab command *inv* in the ordered set of commands

```
p=14; A=[1  2;2  4-10^-p];b=[3  6-10^-p]'; x=inv(A)*b
```

displays exact solution as $x = inv(A) * b = [1\ 1]^t$ for $1 \leq p \leq 14$.

However, for $p \geq 15$, Matlab *inv* treats the system as ill-conditioned with respect to the Matlab precision 15 digits with RCOND = 2.467162e-017. The consequent solution vector displayed is $x = [2\ \ 1]^t$ which is, as expected, unacceptably incorrect. If we now use the pseudo-inverse Matlab command *pinv* in the ordered set of commands

```
p=15,A=[1  2;2 4-10^-p];b=[3  6-10^-p]';x=pinv(A)*b,nx=norm(x)
```

and successively reducing p in the foregoing commands, we then get the minimum-norm least-squares solution x as in Table 1. In Table 1, the right-most column is the comment on how *pinv* has treated the system (1) considering correct up to 4 decimal places. The notation $||x||$ denotes the Euclidean norm of the column vector x. The foregoing system (1) has been an ill-conditioned (strictly) non-singular system. We now consider the following ill-conditioned consistent singular system $Ax = b$, where

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4-10^{-p} & 6 \\ 3 & 6 & 9 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 12-10^{-p} \\ 18 \end{bmatrix} \tag{2}$$

Table 1. Solution vector x for different values of p in the strictly non-singular system (1) using *pinv*

p	x^t	$\|x\|$	System treated as
15	[.6 1.2]	1.3416	singular
14	[6+2x10^{15} 1.2]	1.3416	singular
13	[0.9688 1.0078]	1.3979	Near-singular
12	[1.0039 0..9980]	1.4156	Near-singular
11	[1 0.99995]	1.4141	Not so near-singular
10	[1 1]	1.4142	Non-singular

The Matlab *inv* command will not work since the matrix A is singular. However, the Matlab *pinv* will always work for any matrix A. If we remove "-10^{-p}" throughout in system (2), then the matrix A is singular well-posed and is of rank 1. There are infinity of solutions. The (unique) minimum-norm least-squares solution is $x = [.4286\ .8571\ 1.2857]^t$ (correct up to 4 decimal places) and the Euclidean norm of x is $||x|| = 1.6036$ (correct up to 4 places). The solution is absolutely correct up to 4 places and thus excellent implying that the system (2) without "-10^{-p}" is singular well-posed. Now we take different positive integer values of p and demonstrate in Table 2 the ill-/well-posed behavior of the system (2). Whether the system is ill-posed or not depends on the precision used. The more ill-posed the system is, the more are the errors introduced.

Table 2. Solution vector x for different values of p in the singular system (2) using *pinv* (result retained correct up to 4 decimal places)

p	x	$\|x\|$, *rank*(A)	Singular system treated as
15	$[.4286\ \ .8571\ \ 1.2857]^t$	1.6036, 1	Very ill-posed as if "-10^{-p}" is absent
14	$[.4375\ \ 1\ \ 1.375]^t$	1.7556, 2	Ill-posed
13	$[.4023\ \ 1\ \ 1.2031]^t$	1.6154, 2	Ill-posed
12	$[.4023\ \ 1\ \ 1.2031]^t$	1.6114, 2	Still ill-posed
11	$[.4\ \ .9998\ \ 1.2002]^t$	1.6125, 2	Somewhat ill-posed
10	$[.4\ \ 1\ \ 1.2]^t$	1.6124, 2	Well-posed (up to 4 places)

Failure of pinv to get a solution in a specified domain A singular consistent system will have always infinity of solutions. Out of these solutions, there will be one and only one solution called the minimum-norm least-squares solution which the Matlab command *pinv* computes using the command $x = pinv(A) * b$, $b \neq 0$ (null column vector). For such a system, we may want to obtain a solution in a specified domain since such a solution could be required in a

real world situation. This cannot be achieved using the *pinv* command. The alternative is using a randomized algorithm preferably in a specified narrow domain. Such an algorithm is polynomial-time unlike a deterministic method such as the n-dimensional k-section, where n is the dimension of the solution vector x . Such a deterministic method is exponential-time. If the i -th element of x is $x_i \in [\alpha_i, \beta_i]$ $i = 1, 2, ..., n$ and if the interval $[\alpha_i, \beta_i]$ is divided into k equal subintervals then we need to investigate $k^n = e^{nlogk}$ subdomains for the required solution. This implies that the solution of a moderate size linear system could be intractable even in the fastest available computer today (2007). If we take $k = 2$ (bisection), $n = 50$, then the number of domains to be considered in the computation will be $e^{50log2} = 1.12589990684263e{+}015$. If for each domain, we require one microsecond (10^{-6} second) of computation for the system then we will be requiring 35.7020518405195 years to get the solution in the specified domain. Thus a randomized algorithm is an immediate choice for such a system with specified domain for the solution.

3. Randomized algorithm for $Ax = b$ for specified bounds on x

Consider the consistent ill-posed singular system (2). Let $p = 10$. Then the following self-explanatory Matlab program will be

```
max= 1000000,p=10,n=3; A=[1 2 3;2 4-10^-p 6;3 6 9],
b=[6 12-10^-p 18]',
for j=1:max,
   xl=[.9 .9 .9]'; xh=[1.1 1.1 1.1]';
   x=xl+0.2*[rand rand rand]';
   abserr=norm(A*x-b);
   if abserr<0.5*10^-6, x, abserr, end;
end;
```

The solution that we get is

```
x=[ 0.95261928012747 1.07761579324057 0.964049704922958]'
and the absolute error is abserr=6.96832300233739e-008.
```

Observe that since the system is singular (consistent), we still have infinity of solutions in the specified domain

$$\begin{bmatrix} .9 \\ .9 \\ .9 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 1.1 \\ 1.1 \\ 1.1 \end{bmatrix}, \tag{3}$$

although we know that one of the exact solutions is $x = [1\ 1\ 1]^t$. Another exact solution is $x = [0.7\ 1\ 1.1]^t$ which is, however, just outside the specified domain. Yet another exact solution inside the specified domain is $x = [0.97\ 1\ 1.01]^t$. The randomized algorithm is always polynomial-time since we do not relate the number of random numbers generated with the order n of the matrix, specifically in any non-polynomial form. Further, the algorithm has computed an approximate solution usable in a real world environment in the specified domain. The

uniqueness of the solution in the desired domain depends on whether the system is full-rank or not. We now consider the strictly non-singular system (1) which may be called near-singular ill-posed system with respect to the inverse of A for large positive finite integral values of p for a specified finite precision. For a sufficiently large precision compared to the value of p , the system may not be considered near-singular. In system (1), if we take $p = 8$ and allow absolute error to be less than 0.510^{-6} then the following self-explanatory Matlab program

```
max= 1000000,p=8,n=2;A=[1 2;2 4-10^-p],b=[3 6-10^-p]',
for j=1:max,
        xl=[.9 .9]';
        x=xl+0.2*[rand rand]';
    abserr=norm(A*x-b);
    if abserr<0.5*10^-6, x, abserr, end;
end;
```

gives

```
x =  [ 1.02023212404495  0.989883932303654]'
and abserr = 2.52838866020121e-008
```

while the exact solution is $x = [1\ 1]^t$. This approximate solution is within the specified domain, viz.,

$$\begin{bmatrix} .9 \\ .9 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 1.1 \\ 1.1 \end{bmatrix}, \tag{4}$$

and reasonably good for a real world implementation as seen from the absolute error value. However, if one observes carefully, then one can discover that this strictly mathematically non-singular system has been treated as a singular system. This is typical of ill-posed (near-singular) systems. In a sufficiently large precision (much higher than 15 digits), the foregoing system (1) with $p = 8$ will be treated as well-posed and then the resulting numerical solution will be $x = [1\ 1]^t$ for all practical purposes. If we now take $p = 4$ and allow the $absoluteerror \leq 0.5 \times 10^{-5}$ then out of several solutions, we have the following solution

```
x =   [1.0046787831405   0.997660031115833]',
abserr = 2.37483988449008e-006,
```

which is acceptable in real-world implementation. In the later case, the randomized algorithm has treated the system as near-singular (strictly not singular). We like to stress the fact that a singular consistent linear system will have infinity of solutions, where a randomized algorithm becomes handy if a solution is desired in a specified domain. The usual non-iterative deterministic algorithms will not be straightway applicable as shown through the foregoing examples in this section.

4. Increasing importance of randomized algorithms with computing power

Gradual increase in importance of genetic/randomized algorithms over deterministic ones A numerical solution for the real world implementation is a must. An analytical (mathematical) solution is of no use to an engineer until it is converted into numbers. The most important tool for a numerical solution is mainly a digital computer. A digital computer may be used to practically any finite precision. Every 18 months the CPU (central processing unit) speed is doubling, every 12 months band width is doubling while every 9 months hard disk space is doubling. Consequently, many problems designed/developed earlier which could not be solved due to the limitation of computing power are now being solved. We present below briefly an account of past computing ages and the gradual increase in innovation and importance of randomized algorithms such as the genetic and ant system algorithms over deterministic ones such as the gradient methods. We like to stress the fact that the computing speed, the storage space, and the band width are all together needed to be improved in order to improve the computing power. That is, just increasing the speed of computing without increasing the memory and band width could result in an operational bottle-neck since a high speed CPU will be bogged down due to too many as well as too slow data retrieval and storage operations if the memory size as well as band width are not commensurable, i.e., if these are not comparatively large. As stated earlier, both CPU and memory space are progressively improving.

Pre-high speed computing age (1946-1964) This age consisting of nineteen years may be divided into three parts — early first generation (vacuum tubes) (1946-53), late first generation (1953-59) and the second generation (transistors) computers (1959-64) [40]. The executable memory cycle times were 0.04-40 ms (milliseconds) during the early first generation, 0.01-0.02 ms during the late first generation, and 0.002-0.01 ms during the second generation. An early first generation computer and a late first generation computer were capable of executing about 10^3 and 5×10^4 operations per second on an average, respectively. Hardly a negligible fraction (compared to modern computers) of practical computing existed during 1946-53. Randomized algorithms were not perceived during these years as a viable alternative to deterministic ones, nor much innovation did exist for these algorithms. Also these needed apparently large amount of computation compared to that required by a deterministic one. In reality, however, all randomized algorithms are polynomial-time, i.e., fast. Specifically genetic algorithms (global search techniques) as well as ant system approaches were practically nonexistent during 1946-64. We were more comfortable psychologically with deterministic algorithms than with nondeterministic ones. In reality we did not have much confidence about the result which remains variant at each run unlike the deterministic ones. This is because the seed to generate required random sequences for a randomized algorithm differs from one run to another

— a situation much unlike any deterministic algorithm such as the Gauss reduction method with partial pivoting. A deterministic algorithm will always produce exactly the same output on the same computer, no matter how many times the program (algorithm) is run. Thus Gauss reduction type and similar other algorithms were the only practical means to solve a linear system.

High-speed computing age (1964-1975) This age consisting of the eleven year period may be divided into early third generation (monolithic (medium scale) integrated circuits 1964-69) and late third generation (monolithic (large scale) integrated circuits 1969-75). The executable memory cycle time was $0.5 - 2\mu s$ ($1\mu s = 10^{-6}$ sec) during the early third generation while it was $0.02 - 1\mu s$ during the late third generation. An early third generation and a late third generation computer could execute 10^6 (one million) and 20×10^6 (twenty million) operations per second on an average, respectively. The enhanced speed permitted the scientists, engineers, and researchers to explore more compute bound problems which were hitherto discouraged due to processor speed and memory capacity/speed limitations. Also, they designed and developed newer algorithms suited to computations for real world problems. Randomized algorithms such as the evolutionary approaches specifically genetic algorithms started gaining acceptance and momentum increasingly and are being considered as possible candidates for practical computations along with the deterministic ones. But these were yet to become sufficiently appealing for extensive computation in lieu of deterministic ones and were yet to be widely accepted means of linear system solver.

Super-high-speed computing age (1975-1990) The term supercomputer has a time-dependant informal definition since today's supercomputer tends to become tomorrow's normal computer. Further, there is no generally accepted definition for fourth generation (very large scale monolithic integrated (VLSI) circuits and possibly with vector processors) as well as fifth generation (implementing artificial intelligence through usually a software simulation of the natural intelligence) computers. It is thus not meaningful to extend the concept of computer generation beyond the third generation. We would consider computers introduced since 1975 onwards as modern computers and refer to the third generation computers as those of the past. However, for the purpose of speed relative to that of the past computers, a modern computer was loosely termed as a supercomputer if its speed exceeds 100 million (floating-point) operations per second (one hundred megaflops). Such a technological improvement gave a significant impetus to researchers to explore much more compute bound algorithms such as the randomized ones and perceive the scope/utility of these algorithms over the deterministic algorithms such as the deterministic linear system solver.

Ultra-high-speed computing age (1990-onwards) Compared to the foregoing speeds, it would not be unreasonable to term a processing speed exceeding 1000 million (i.e., one billion) flops as an ultra-high speed. The ultra-high frequency band is generally accepted as 3000-300 megahertz. Electrical signals propagate

no faster than the speed of light. A random access memory (RAM), i.e., the executable memory used to one gigahertz (10^9 cycles per second) will deliver information at 10^{-10} sec (i.e., 0.1 nanosecond) speed if its diameter is 3 centimeters since in 10^{-10} seconds, light travels 3 centimeters [41]. It is the physics, rather than the technology and the architecture, that sets up the limits/barriers to increase the computational speed arbitrarily. The physical barriers are the (i) speed of light, (ii) the thermal efficiency, and the quantum barriers. Per mass of hydrogen atom (1.67×10^{-24} gm), maximum 2.505×10^{23} bits/sec can be theoretically processed/transmitted. Since the number of protons in the universe is estimated as 10^{73}, no more than 7.9×10^{103} bits per year can be processed, if the whole universe is dedicated to information processing [41]. The ultra-high speeds along with ultra-large memory and ultra-large band-width have allowed the researchers to encroach into the realm of hitherto unexplored NP-hard problems such as a large traveling salesman problem (TSP) of immense practical importance in a meaningful non-deterministic (randomized) way. While a deterministic algorithm for the TSP is combinatorial/exponential-time needing the computation for $(n-1)!$ paths to obtain the exact minimum cost path, a genetic (heuristic) approach which is always polynomial-time would need relatively very little computation to provide us a low cost path, which though may/may not be the exact minimum cost path, that is accepted and used by the traveling salesman. There is absolutely no polynomial-time way to verify whether the computed result (path) is truly the globally minimal path. Possibly in future a better (lower cost) path will be found by this or some other algorithm with/without increased computing power. We will still not be able to check whether the lower cost path is a globally minimal cost path. The purpose is to impress on the fact that randomized algorithms will be the only tool to explore the vast world of NP-hard problems [41, 42]. The deterministic algorithms will have no entry to this world as these could take billions of centuries to produce the required output. Even the estimated age of the universe is a numerical zero compared to this computation time. In the present context, however, we are not involved in NP-hard problem. Our problem, viz., the linear system solving, specifically ill-conditioned singular linear system solving, is much simpler and polynomial-time. We have attempted here too the distinct scope of a randomized algorithm. Even if the employed algorithm is an exponential function of n, the order of the matrix A, then it will be an exponential algorithm. So long as the given linear system is not considered large, such an exponential algorithm will produce the required acceptable solution in a tolerable time frame because of teraflops speed available to us [43]. When the algorithm is not an exponential function of n, it will be a polynomial-time algorithm and hence fast. Observe that there are numerous not-so-large ill-posed systems in a real-world environment [44, 45]. Even in such polynomial-time problems, a genetic algorithm/any evolutionary approach appears to be competitive and by nature is simple to implement rather readily as shown in the physically concise Matlab codes.

World's fastest computers as in November 2007 Supercomputers are typically used for highly compute intensive problems such as those in quantum physics, molecular modeling, weather forecasting and climate research, and physical simulation including that of nuclear tests. While the US is the leading consumer of ultra-high speed computing systems with 284 of the 500 systems, Europe follows with 149 systems and Asia has 58 systems. In Asia, Japan leads with 20 systems, Taiwan has 11, China 10 and India 9 [43]. The No. 1 position goes to the BlueGene/L System, a joint development of IBM and the US Department of Energy's (DOE) National Nuclear Security Administration (NNSA) and installed at DOE's Lawrence Livermore National Laboratory in California. Although BlueGene/L has been in the No. 1 position since November 2004, the current system is much faster at 478.2 teraflops compared to 280.6 teraflops six months ago before its upgrade. BlueGene/P system installed in Germany at the Forschungszentrum Juelich (FZJ) is in the No. 2 position with the processing speed of 167.3 teraflops while the No. 3 system is at the New Mexico Computing Applications Center (NMCAC) in Rio Rancho in New Maxico, having the speed of 126.9 teraflops. The No. 4 position goes to the Tata supercomputer called EKA (meaning "one" in Sanskrit) situated at the Computational Research Laboratory (CRL) in Pune, India. It is a Hewlett-Packard Platform 3000 BL 460c system integrated with CRL's own innovative routing technology to achieve a speed of 117.9 teraflops. CRL built the supercomputer facility using dense data center layout and novel network routing and parallel processing library technologies developed by its scientists. The second ranked supercomputer in India, rated 58th in the Top500 list, is at the Indian Institute of Science, Bangalore. India has been making steady progress in the field of supercomputing from the time it first bought two supercomputers from the US pioneer Cray Research in 1988. US strictures on the scope of its use and its demand for intrusive monitoring and compliance led India to devise its own supercomputers using clusters of multiprocessors. The foregoing information under the heading "World's fastest computers as in November 2007" follows primarily from [43].

5. Conclusions

Ill-posed polynomial root-finding versus ill-posed singular system Multiple zeros of a polynomial can be very accurately/exactly computed using a fixed-point iterative scheme such as the deflated Newton method [46]. Thus multiple polynomial root-finding problem is well-conditioned in the sense that the roots can be very accurately computed by some numerical method. However, this is not so with the polynomial root-cluster (closely spaced roots) problem. Any numerical method devised/developed so far and possibly to be developed in future for a specified precision can be easily shown to fare badly by appropriately constructing a polynomial with sufficiently closely spaced roots/zeros. Thus, for a specified precision, the polynomial root-cluster problem is ill-conditioned with respect to computing the polynomial's zeros. Let a polynomial have the repeated zeros 5, 5, 5, 5, and 5. The deflated Newton method will produce the

zeros 5, 5, 5, 5, and 5 correct up to 14 decimal places when Matlab (15 digit precision) is used. If, on the other hand, the polynomial has zero-clusters 5.01, 5.02, 5.03, 5.04, and 5.05 then any method will fare badly introducing significant errors in the zeros. The extent of such errors, however, depends on the precision employed. Consider the Matlab commands

```
>>format long g; A=[5.01  5.02  5.03  5.04  5.05]; c=poly(A)
```

to construct the polynomial $c_1x^5 + c_2x^4 + c_3x^3 + c_4x^2 + c_5x + c_6 = 0$ where $c_1 = 1$, $c_2 = -25.15$, $c_3 = 253.0085$, $c_4 = -1272.627725$, $c_5 = 3200.63975274$, $c_6 = -3219.818138712$. The Matlab command $>> r = roots(c)$ then produces the five roots 5.05007152279192, 5.03970336092616, 5.03043294325773, 5.0197197217852, and 5.01007245123899 which are significantly in error and are different from the actual root-clusters 5.01, 5.02, 5.03, 5.04, and 5.05.

Exact singular linear system problem like multiple polynomial root-finding problem is well-conditioned. Singular linear system with nearly linearly dependent rows problem, on the other hand, like the polynomial root cluster problem (which might/might not have some repeated roots) is ill-posed. In the later cases, errors will be introduced in computation.

Deterministic iterative versus non-iterative and randomized algorithms While deterministic non-iterative methods usually have no regards for the specified (n -dimensional) domain, the deterministic iterative methods do have. The different kinds of generalized inverses of the matrix A will produce different solutions for the system $Ax = b$. These inverses neither need the knowledge of the required domain nor can be readily modified to incorporate the desired domain. Most often these inverses will produce a solution outside the domain. In real world environment, such a solution could be of no use. Under these circumstances, a randomized algorithm is desirable in more than one ways. It will produce a solution inside the domain if there is a solution and if sufficiently large number of random numbers each of, say, fifteen digits are used. It is also simple and straightforward. In view of the ultra-high computing power (over one billion floating point operations per second), the randomized algorithms are more attractive now than ever. These are polynomial-time, i.e., fast and in most real-world problems, we do not face any meaningful computational complexity issue.

A typical example to focus on near-singularity as the real problem not the singularity

(a) *Non-homogeneous linear system* Consider the strictly nonsingular linear system $Ax = b$, where

$$A = \begin{bmatrix} .1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 + .5 \times 10^{-k} \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 15 \\ 24 + .5 \times 10^{-k} \end{bmatrix},$$

and $k =$ a finite positive integer. The relative errors in the solution vector x by four different methods, viz., Matlab inv, Matlab pinv, Optimal iterative scheme

(OIS-quadratic) [2], Matlab inversion-free (Gauss elimination) are given in Table 3.

Table 3 Relative error in the computed solution vector x

k	Matlab *inv*	Matlab *pinv*	OIS *pinv*	Matlab inv-free
4	1.0082×10^{-10}	3.6351×10^{-10}	1.5570×10^{-5}	5.0243×10^{-11}
5	0	1.2023×10^{-9}	3.8565×10^{-3}	5.0243×10^{-10}
6	9.6187×10^{-9}	8.6032×10^{-9}	3.2120×10^{0}	8.1645×10^{-9}
7	1.4189×10^{-7}	1.2876×10^{-7}	11.7624	5.0243×10^{-8}
8	5.5060×10^{-7}	2.0602×10^{-6}	1589.90	5.0243×10^{-7}
9	4.4048×10^{-6}	1.3214×10^{-5}	445919.5	6.2804×10^{-7}
10	7.8796×10^{-5}	1.4529×10^{-4}	29983433	9.4208×10^{-5}
11	0	1.3811×10^{-3}	1.9689×10^{-13}	5.0243×10^{-4}
12	1.0086×10^{-2}	2.0172×10^{-2}	1.9299×10^{-14}	3.7780×10^{-3}
13	8.8388×10^{-2}	1.9357×10^{-15}	3.3650×10^{-15}	6.3418×10^{-3}
14	0	2.9036×10^{-15}	1.2094×10^{-15}	5.2378×10^{-2}

Given the $m \times n$ numerical matrix A , the quadratic OIS [2] is as follows.

Step 1. Compute $A^t/tr(AA^t)$.

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 $k = 0, 1, 2, ...$, till $||X_{k+1} - X_k||/||X_{k+1} \leq 0.5 \times 10^{-10}$.

As k increases, the singularity of A is more pronounced, the matrix A is always strictly non-singular though. The exact solution vector e is $e = [1\ 1\ 1]^t$. *Observe the error phenomenon!* The relative error is defined as $||e - x||/||e||$, where $||y||$ = Euclidean norm of the vector y . The graphical representation of the relative error in the solution vector computed using Matlab (pseudoinverse) pinv and Optimal iterative scheme (OIS-quadratic) pinv [2] is shown in Fig. 1.

The graph (Fig. 1) has been obtained using the Matlab script file *errormatlabpinvoispinv* consisting of the program

```
e=[3.6351*10^-10 1.2023*10^-9 8.6032*10^-9 1.2876*10^-7
2.0602*10^-6 1.3214*10^-5 1.4529*10^-4 1.3811*10^-2
2.0172*10^-2 1.9357*10^-15 2.9036*10^-15]';

e1=[1.5570*10^-5 3.8565*10^-3 3.2120*10^0 11.7624
    1589.90 445919.5 29983433 1.9689*10^-13
    1.9299*10^-14 3.3650*10^-15 1.2094*10^-15]';
k=[4 5 6 7 8 9 10 11 12 13 14]';
plot( k,log(e), k, log(e1),'--'); xlabel('k');
    ylabel('natural log of errors');
title('Error pattern in Matlab pinv and that in
```

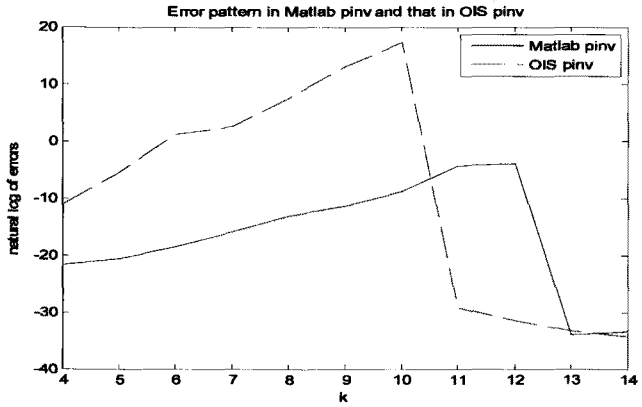

Fig. 1 Relative error pattern in the solution vector using (i) Matlab pinv and (ii) OIS pinv. As k increases the degree of singularity also increases. Both the curves are similar in nature and demonstrate that when the degree of singularity is low, i.e., when the linear system is close to a non-singular system as well as when it is close to an exactly singular system with respect to the precision, relative errors are very low or negligible. On the other hand, the relative errors are high in the middle, i.e., when the system is near-singular with respect to the precision.

```
    OIS pinv');
legend('Matlab pinv','OIS pinv');
```

The Matlab (true inverse) *inv* as well as Matlab (Gauss elimination) inv-free both produce monotonically increasing error as the singularity increases. It can be seen from Table 3 that the Matlab inv as well as Matlab inv-free perform better than Matlab pinv in the near singular region. So, for near singular system we recommend the use of Matlab inv/ inv-free rather than pinv. When the degree of singularity is too high with respect to the precision or the system is singular, i.e. two or more rows of the coefficient matrix A are linearly dependent, we recommend the use of Matlab *pinv* or OIS pinv.

In the OIS, the matrix X_{k+1} upon satisfaction of the foregoing inequality will be A^+ correct up to 10 significant digits. X_{k+1} will be a more accurate right-inverse of A rather than its left-inverse, when A is near-singular. The scheme (OIS) cannot compute only the left-inverse X such that only $XA = I$ while other relations required for the pseudo-inverse are not satisfied for a near-singular A. The relation for near-singular A , viz., $AX_{k+1} = I \neq X_{k+1}A$ within the standard precision (14 digits) is interesting. If the singularity is more pronounced, then the inequality will also be much more pronounced. Further, while the two relations $AX_{k+1}A = A$, $(AX_{k+1})^t = AX_{k+1}$, where X_{k+1} is the right-inverse of A for a near-singular A subject to the precision of computation are satisfied, the other two relations $X_{k+1}AX_{k+1} = X_{k+1}$, $(X_{k+!}A)^t = X_{k+1}A$ will not be satisfied. However, for exact singular/non-square rectangular matrices with full row-rank, the inverse X_{k+1} will reasonably (within the precision of

the computer) satisfy all the four relations in the foregoing optimal iterative scheme (OIS). Hence this X_{k+1} will be the minimum-norm least-squares inverse of non-singular/rectangular A with full row rank.

(b) *Eigensystem (Homogeneous linear system)* Consider the computation of numerical eigenvalues and eigenvectors of the matrix.

$$A = \begin{bmatrix} 5 & 3 & 2 & 4 \\ -7 & 10 & 2 & 8 \\ -12 & 8 & -1 & 14 \\ -14 & 21 & 3 & 26 \end{bmatrix}.$$

Using the Matlab command >> $eig(A)$, we get the four distinct eigenvalues as

```
31.3535186425859, 8.29665538126231,
0.349825976151775,  1.84619639813168e-014
```

If we now take the first eigenvalue $\lambda_1 = 31.3535$ (correct up to 4 places) then in the homogeneous system $(A-\lambda_1 I)x_1 = 0$, where I is the unit matrix of order 4 and x_1 is a corresponding eigenvector, the system will produce the eigenvector (non-normalized)

```
x1 = [-3.63797880709171e-012  -7.27595761418343e-012
      -7.27595761418343e-011  -2.91038304567337e-011]'
```

which is completely *wrong*. The matrix $A - \lambda_1 I$ which should be numerically singular has been treated as numerically non-singular. Its determinant is -0.417835384572988 which is significantly different from the numerical zero [41]. The foregoing eigenvector was obtained using the Matlab commands

```
>>lam1=31.3535; I=eye(4); B=A-lam1*I; z=[1 1 1 1]';
  x1=(I-pinv(B)*B)*z  %z=arbitrary col. vector
```

However, if we now take the first eigenvalue $\lambda_1 = 31.3535186425859$, then we get the numerically correct eigenvector (non-normalized)

```
x1 = [0.329995055166732  0.504562453415674
      0.640768907286834 1.47532641586924]'
```

which is very much acceptable and usable in a real-world application. The foregoing vector was obtained by the foregoing Matlab commands with only replacement of $lam1 = 31.3535$ by $lam1 = 31.3535186425859$. This is just to demonstrate that a relative error of 0.00006 per cent in the eigenvalue can be fatal even for such a small eigensystem.

The exact singularity or too high a degree of singularity (with respect to the precision of computation) is never a problem so far as the quality of the solution is concerned; that is, the problem is NOT ill-conditioned with respect to the computation of the solution vector x . On the other hand, the problem is ILL-CONDITIONED with respect to the computation of the solution vector when it is NEAR-SINGULAR or it has two or more near-linearly dependent rows (with respect to the precision of the computer). The foregoing simple small examples demonstrate precisely that. We are not proposing a best practical

method to solve a large least-squares problem or a large linear system that will excel all the methods so far existing. However, our subtle points must be taken into account for devising any method/algorithm for large sparse/dense system. Only the error-bounds will demonstrate how good the devised method is. It may be stressed that there are numerous real-world problems in Physics such as the forced oscillations and resonance [44] and in Operations Research [45], whose mathematical models are linear systems including singular and near-singular ones. The scope of computations using a randomized algorithm has been explicitly discussed in section 4 stressing the fact that the ultra-high speed ($teraflop/s = 10^12$ floating point operations per second) of current computers is widely available and over 95 per cent of this computing power remains unutilized (and hence is an waste) in the real-world environment. Further it must be realized that a randomized algorithm is polynomial-time (implying fast) and NOT exponential-time (implying slow). For, if it is made exponential-time then it is useless in practice.

References

1. S.K. Sen and S. Sen, *Linear systems: relook, concise algorithms, and Matlab programs*, Academic Studies - National Journal of Jyoti Research Academy**1**(1) (2007), 1-8.
2. S.K. Sen and S.S. Prabhu, *Optimal iterative schemes for computing Moore-Penrose matrix inverse*, Internat. J. Systems Sci. **8** (1976), 748-753.
3. E.A. Lord, V. Ch. Venkaiah, and S.K. Sen, *A concise algorithm to solve under-/over-determined linear systems*, Simulation **54** (1990), 239-240.
4. M. Haahr, *Introduction to randomness and random numbers*, http://www.random.org/essay.html 1999.
5. S. Galanti and A. Jung, *Low-discrepancy sequences: Monte Carlo simulation of option prices*, The Journal of Derivatives **5** (1997), 63-83.
6. T. Samanta, *Random Number Generators: MC Integration and TSP-solving by Simulated Annealing*, Genetic and Ant System Approaches, Ph.D. thesis, Florida Institute of Technology (Department of Mathematical Sciences), Melbourne, Florida, USA 2006.
7. A. Reese, *Quasi- Versus Pseudo-random Numbers with Applications to Nonlinear Optimization*, Ph.D. thesis, Florida Institute of Technology (Department of Mathematical Sciences), Melbourne, Florida, USA 2006.
8. S.K. Sen, T. Samanta, and A. Reese, *Quasi- versus pseudo-random generators: Discrepancy, complexity and integration-error based comparison*, International Journal of Innovative Computing, Information and Control **2** 3 (2006), 621-651.
9. J.H. Halton, *On the efficiency of certain quasi-random sequences of points in evaluating multi dimensional integrals*, Numerische Mathematik **2** (1960), 84-90.
10. I.M. Sobol, *On the distribution of points in a cube and the approximate evaluation of integrals*, U.S.S.R. Computational Mathematics and Mathematical Physics **7** (1967) 86-112.
11. H. Faure, *Discrepance de suites associees a un systeme de numeration (en dimension s)*, Acta Arithmetica, **XLZ** (1982), 337-351.
12. H. Niederreiter, *Low discrepancy and low dispersion sequences*, Journal of Number Theory **30** (1988),51-70.
13. B.L. Fox, *Algorithm 647: Implementation and relative efficiency of quasirandom sequence generators*, ACM Transactions on Mathematical Software **12** 4 (1986), 362-376.

14. P. Bratley, B.L. Fox, and H. Niederreiter, *Algorithm 738: Programs to generate Niederreiter's low-discrepancy sequences*, ACM Transactions on Mathematical Software **20** 4 (1994), 494-495.
15. H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, SIAM, Pennsylvania, 1992.
16. P.K. Sarkar and M.A. Prasad, *A comparative study of pseudo and quasi random sequences for the solution of integral equations*, Journal of Computational Physics **68** (1987), 66-88.
17. P. Shirley, *Discrepancy as a quality measure for simple distributions*, Proceedings of Eurographics **91** (1991), 183-193.
18. J. Struckmeier, *Fast generation of low discrepancy sequences*, Journal of Computational and Applied Mathematics, **61** (1995), 29-41.
19. L. Kocis and W. Whiten, *Computational investigations of low-discrepancy sequences*, ACM Transactions on Mathematical Software **23** 2 (1997), 266-294.
20. S.G. Henderson, B.A. Chiera, and R.M. Cooke, *Generating "dependent" quasi-random numbers*, Proceedings of the 32nd Conference on Winter Simulation, (2000), 527-536.
21. S. Haupt and R. Haupt, it Practical Genetic Algorithms, Wiley, New York, 1998.
22. M. Mascagni and A. Karaivanova, *Matrix computations using quasirandom numbers*, Springer Verlag Lecture Notes in Computer Science, 552-559, 2000.
23. J. Banks, J. Carson, and B. Nelson, *Discrete-event System Simulation* (2nd edition), Prentice-Hall, New Jersey, 1996.
24. T. Samanta and S.K. Sen, *Pseudo- versus quasi-random generators in heuristics for traveling salesman problem* (2008), to appear.
25. M. Junger, G. Reinelt, and G. Renaldi, *The traveling salesman problem*, Operations Research and Management Sciences **7** (1995),225-330.
26. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, anf D.B. Shmoys, *The Traveling Salesman Problem A Guided Tour of Combinatorial Optimization*, Wiley, New York, 1985.
27. G. Reinelt, it The Traveling Salesman: Computational Solutions for TSP Applications, Springer-Verlag, New York, 1994.
28. Georgia Tech TSP page, http://www.tsp.gatech.edu/.
29. W.L. Winston, *Operations Research: Applications and Algorithms* (4th Edition), Thomson, Belmont, California, 2004.
30. M. Dorigo, *Optimization, Learning, and Natural Algorithms*, Ph.D. thesis (in Italian), Politecnico di Milano, Italy, 1992.
31. M. Dorigo, V. Maniezzo, and A. Colorni, *The ant system: Optimization by a colony of cooperating agents*, IEEE Trans. On Systems, Man, and Cybernetics-Part B **26** 1 (1996), 29-41.
32. L.M. Gambardella and M. Dorigo, *Solving symmetric and asymmetric TSPs by ant colonies.* Proceedings of the IEEE Conference on Evolutionary Computation, ICEC96, Nagoya, Japan, 622-627, 1996.
33. H. Chi, M. Mascagni, and T. Warnock, *On the optimal Halton sequence*, Mathematics and Computers in Simulation **70**(2005), 9-21.
34. H. Niederreiter, *Low-discrepancy and low dispersion sequences*, Journal of Number Theory **30** (1988),51-70.
35. P. Bratley, B.L. Fox, and H. Niederreiter, *Implementation and test of low-discrepancy sequences*, ACM Transections on Modeling and Computer Simulation **2** 3 (1992), 195-213.
36. V. Lakshmikantham, S.K. Sen, and T. Samanta, *Comparing random number generators using Monte Carlo integration*, International Journal of Innovative Computing, Information, and Control **1** 2 (2005), 143-165.
37. D.E. Knuth, *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, MA, 2nd Edition, 1981.
38. S.K. Park and K.W. Miller, *Random number generators: Good ones are hard to find*, Communications of the ACM **31** (1988), 1192-1201.

39. MP-TESTDATA - The TSPLIB Symmetric Traveling Salesman Problem Instances, http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/ ,1995.
40. A. Ralston and E.D. Reilly, Jr. eds., *Encyclopedia of Computer Science and Engineering* (2nd ed.), Van Nostrand Reinhold, New York, 1983.
41. V. Lakshmikantham and S.K. Sen, *Computational Error and Complexity in Science and Engineering*, Elsevier, Amsterdam, 2005.
42. E.V. Krishnamurthy and S.K. Sen it Introductory Theory of Computer Science, Affiliated East-West Press, New Delhi, 2004.
43. Chidanand Rajghatta, The Times of India, TNN, *India hosts world's fourth fastest supercomputer*, The Times of India Daily News Paper, reported on November 13, 2007 at 2143 hours Indian Standard Time from Washington.
44. B.Nobel and J.W. Daniel, Applied Linear Algebra (3rd ed), Prentice-Hall, New Jersey, 1998.
45. W.L. Winston, *Operations Research: Applications and Algorithms* (4th ed), Thomson, Belmont, California, 2004.
46. E.V. Krishnamurthy and S.K. Sen, *Numerical Algorithms: Computations in Science and Engineering*, Affiliated East-West Press, New Delhi, 2001.

S.K. Sen Syamal Kumar Sen received his Ph.D. degree in Computational Science from IISc, Bangalore. After serving the institute for over thirty-five years, he joined Florida Tech, USA in January, 2004 as a professor. He has been earlier a professor in IISc and other universities besides holding one-year visiting professorship (1995-96) and Fulbright fellowship (1991) for senior teachers in Florida Tech. He is widely published, cited in *Marquis Whos Who* and is the recipient of 2008 IEEE Canaveral Section *Outstanding Recognition Award.*

Department of Mathematical Sciences, Florida Institute of Technology, 150 West University Boulevard, Melbourne, FL 32901-6975, United States
e-mail: sksen@fit.edu

Ravi P. Agarwal received his Ph.D. degree in Mathematics from IIT, Madras (Chennai). After serving National University of Singapore (Singapore) for nearly two decades, he joined Florida Tech, USA in January, 2002 as a professor. He has been earlier a professor in several universities besides being a recipient of Humboldt-Foundation fellowship (1980-82), Germany. He is one of the most published and most cited authors in the world and an authority in differential equations and applications.

Department of Mathematical Sciences, Florida Institute of Technology, 150 West University Boulevard, Melbourne, FL 32901-6975, United States
e-mail: agarwal@fit.edu

Gholam Ali Shaykhian received his Ph.D. degree in Operations Research from Florida Tech, USA. He is a Software Engineer with NASA at Kennedy Space Center. He has been bestowed with several awards and recognitions from NASA including Space Flight Awareness (SFA) Honoree, and Certificate of Appreciation. Ali currently serves as the *Program Chair* of ASEE, Minorities in Engineering Division and *Education Chair* of IEEE. He was a NASA Fellow, serving his fellowship at Bethune Cookman College from June 2003 to June 2005.

National Aeronautics and Space Administration (NASA), Engineering Directorate, NE-C1, Kennedy Space Center, FL 32899, United States
ali.shaykhian@nasa.gov