MANCHESTER 1824

M\cr NA
Manchester Numerical Analysis

The University of Manchester

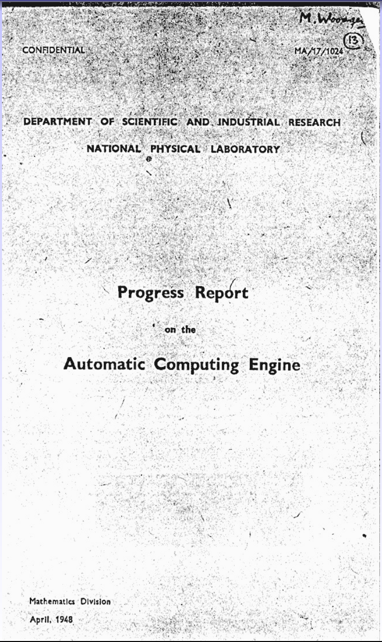
Solving Dense Linear Systems: A Brief History and Future Directions

Nick Higham
Department of Mathematics
The University of Manchester
https://nhigham.com

Slides available at https://bit.ly/dongarra70

New Directions in Numerical Linear Algebra and High Performance Computing: Celebrating the 70th Birthday of Jack Dongarra, July 7–8, 2021

Wilkinson (1948)


M. Woodger

CONFIDENTIAL MA/17/1024

DEPARTMENT OF SCIENTIFIC AND INDUSTRIAL RESEARCH

NATIONAL PHYSICAL LABORATORY

Progress Report

on the

Automatic Computing Engine

Mathematics Division

April, 1948

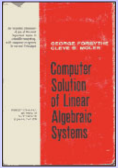
Confidential NPL report on the Automatic Computing Engine (ACE) gives program implementing LU factorization with partial pivoting and iterative refinement.

Main Developments

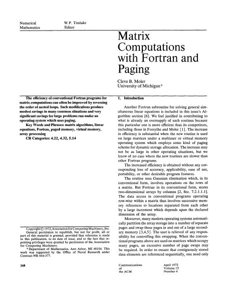
- Backward error analysis.
- Exploiting computer architecture.
- Exploiting parallelism.
- Software engineering.
- Exploiting different precisions of arithmetic.
- Exploiting structure in A.

$Ax = b$ Solver

Forsythe & Moler (1967), *Computer Solution of Algebraic Equations*: Algol, Fortran and PL/I codes for solving $Ax = b$



Moler (1972): importance of accessing arrays in column order in Fortran.
"The efficiency of … Fortran programs for matrix computations can often be improved by reversing the order of nested loops.



Numerical Mathematics
W.P. Timlake
Editor

Matrix Computations with Fortran and Paging

Cleve B. Moler
University of Michigan*

The efficiency of conventional Fortran programs for matrix computations can often be improved by reversing the order of nested loops. Such modifications produce modest savings in many common situations and very significant savings for large problems run under an operating system which uses paging.

Key Words and Phrases: matrix algorithms, linear equations, Fortran, paged memory, virtual memory, array processing
CR Categories: 4.22, 4.32, 5.14

1. Introduction

Another Fortran subroutine for solving general simultaneous linear equations is included in this issue's Algorithm section [6]. We feel justified in contributing to what is already an oversupply of such routines because this particular one is more efficient than its competitors, including those in Forsythe and Moler [1]. The increase in efficiency is substantial when the new routine is used on large matrices under a multiuser or virtual memory operating system which employs some kind of paging scheme for dynamic storage allocation. The increase may not be as large in other operating situations, but we know of no case where the new routines are slower than other Fortran programs.

The increased efficiency is obtained without any corresponding loss of accuracy, applicability, ease of use, portability, or other desirable program features.

The routine uses Gaussian elimination which, in its conventional form, involves operations on the rows of a matrix. But Fortran in its conventional form, stores two-dimensional arrays by columns [2, Sec. 7.2.1.1.1]. The data access in conventional programs operating row-wise within a matrix thus involves successive memory references to locations separated from each other by a large increment which depends upon the declared dimension of the array.

Moreover, many modern operating systems automatically partition the array storage into a number of separate pages and swap these pages in and out of a large secondary memory [3,4,5]. The user is relieved of any responsibility for controlling this swapping. When the conventional programs above are used on matrices which occupy many pages, an excessive number of page swaps may be required. In order to ensure that contiguously stored data elements are referenced sequentially, one need only

* Department of Mathematics, Ann Arbor, MI 48104. This work was supported by the Office of Naval Research under Contract NR 044-377.

programs for matrix computations can often be improved by reversing the order of nested loops.

→ LINPACK (1979) → LAPACK (1992)
→ MAGMA (2008), PLASMA (2009) → …

Basic Linear Algebra Subprograms (BLAS)

- **Level 1** Lawson, Hanson, Kincaid & Krogh (1979). *Vector operations*.
- **Level 2** Dongarra, Du Croz, Hammarling & Hanson (1988). *Matrix–vector operations*.
- **Level 3** Dongarra, Du Croz, Hammarling & Duff (1990). *Matrix–matrix operations*.
- **Batched** Abdelfattah, Costa, Dongarra, Gates, Haidar, Hammarling, H, Kurzak, Luszczek, Tomov, and Zounon (2021): Batched BLAS. *Many independent BLAS operations on small matrices.*

An interesting feature of the codes is
that they made a very intensive use of subroutines;
the addition of two vectors,
multiplication of a vector by a scalar,
inner products, etc.,
were all coded in this way

— **J. H. Wilkinson (1980)**

Unrolling Loops in FORTRAN

Dongarra & Hinds (1979)

Original

for $i = 1\colon 1\colon n$
  $y_i = y_i + \alpha x_i$
end

Unrolled loop

for $i = 1\colon 4\colon n$
  $y_i = y_i + \alpha x_i$
  $y_{i+1} = y_{i+1} + \alpha x_{i+1}$
  $y_{i+2} = y_{i+2} + \alpha x_{i+2}$
  $y_{i+3} = y_{i+3} + \alpha x_{i+3}$
end

Speedups typically about 1.5.

BLAS on a Microcomputer (**H, 1985**)

Times in seconds for solving $Ax = b$ with $n = 60$.

	Basic	Assembly BLAS	Speedup
Commodore 64	1535	298	5.2
BBC Micro	450	162	2.8

Pure Basic times dominated by subscripting!

```
| "SUBROUTINE SAXPY (N,SA,SX,SY1)"
| VECTOR = VECTOR+CONST*VECTOR: SY() : = SY()+SA*SX ()
| SYS AXPY,N,SA,SX(),SY()
SAXPY    JSR GETN
!****
          JSR GET3            ! (PTR3) -> SA
          JSR GET2            ! (PTR2) -> SX ()
          JSR GET1            ! (PTR1) -> SY ()
LOOPSAX   LDA NLOW            ! N=0?
          ORA NHIGH
          BEQ FINSAX
...
```

Growth Factor for Partial Pivoting

$$\rho_n(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \geq 1.$$

$\rho_n \leq 2^{n-1}$ but **almost always small** in practice (Wilkinson).

Growth Factor for Partial Pivoting

$$\rho_n(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \geq 1.$$

$\rho_n \leq 2^{n-1}$ but **almost always small** in practice (Wilkinson).

```
>> gf(randn(1000))
ans =
   1.5997e+01
>> gf(gallery('randsvd',1000,1e8,2,[],[],1))
ans =
   7.5329e+01
```

D. Higham, H, & Pranesh (2021): $\rho_n \gtrsim \dfrac{n}{4\log n}$.

Growth Factor for Partial Pivoting

$$\rho_n(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \geq 1.$$

$\rho_n \leq 2^{n-1}$ but **almost always small** in practice (Wilkinson).

```
>> gf(randn(1000))
ans =
   1.5997e+01
>> gf(gallery('randsvd',1000,1e8,2,[],[],1))
ans =
   7.5329e+01
```

D. Higham, H, & Pranesh (2021): $\rho_n \gtrsim \dfrac{n}{4 \log n}$.

Open problem to explain ρ_n behavior!

Iterative Refinement for $Ax = b$ (classic)

Solve $Ax_0 = b$ by LU factorization in **double precision**.

- $r = b - Ax_0$ **quad precision**
- Solve $Ad = r$ **double precision**
- $x_1 = \mathrm{fl}(x_0 + d)$ **double precision**

($x_0 \leftarrow x_1$ and iterate as necessary.)

- Programmed in **J. H. Wilkinson**, *Progress Report on the Automatic Computing Engine* (1948).
- Popular up to 1970s, exploiting cheap accumulation of inner products.

Iterative Refinement (1970s, 1980s)

Solve $Ax_0 = b$ by LU factorization.

- $r = b - Ax_0$
- Solve $Ad = r$
- $x_1 = \mathrm{fl}(x_0 + d)$

Everything in **double precision**.

- **Skeel (1980)**.
- **Jankowski & Woźniakowski (1977)** for a general solver.

Iterative Refinement (2000s)

Solve $Ax_0 = b$ by LU factorization in **single precision**.

- $r = b - Ax_0$ **double precision**
- Solve $Ad = r$ **single precision**
- $x_1 = fl(x_0 + d)$ **double precision**

- **Dongarra, Langou et al. (2006)**.
- Motivated by single precision at least twice as fast as double on Intel chips, up to 14 times faster on Sony/Toshiba/IBM Cell processor.

Iterative Refinement in Three Precisions

A, b given in **double precision**.

Solve $Ax = b$ by LU factorization in **half precision**.

- $r = b - A\widehat{x}$ **quad precision**
- Solve $Ad = r$ **half precision**
- $y = \widehat{x} + d$ **double precision**

- **Carson & H (2017, 2018)**.
- Motivated by availability of half precision on GPUs.

GMRES-Based Iterative Refinement

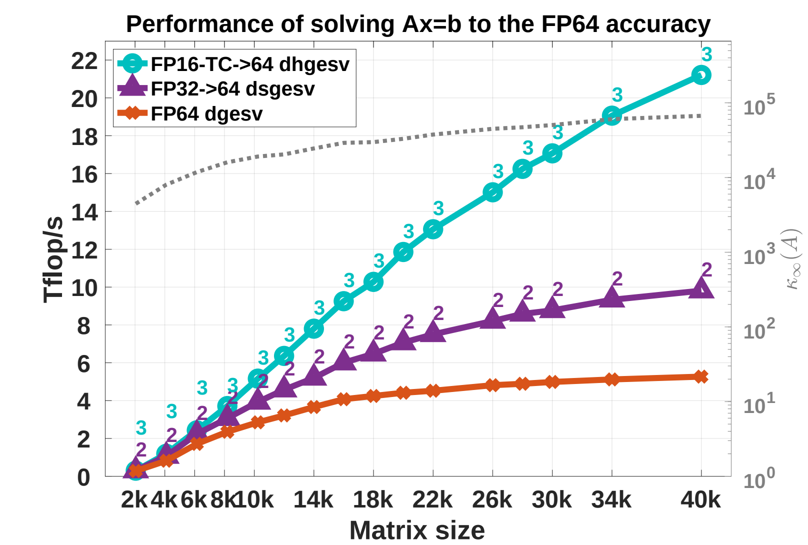
A, b given in precision u; additional precs u_f, u_p, u_g, u_r.

- Compute LU fact'n (w/pivoting) in prec u_f
- Solve $\widetilde{L}\widetilde{U}x_1 = b$ in prec u_f.
- For $i = 1, 2, \dots$
  - $r_i = b - Ax_i$ prec u_r
  - Solve $MAd_i = Mr_i$ by GMRES in prec u_g where $M = \widetilde{U}^{-1}\widetilde{L}^{-1}$ and products with MA in prec u_p.
  - $x_{i+1} = x_i + d_i$ prec u

- **Carson & H (2017/18)**: **three** precs ($u_g = u$, $u_p = u_r$).
- **Amestoy, Buttari, H, L'Excellent, Mary & Vieublé (2021)**: **five** precs.
- Implemented with $u_r = u_p = u_g = u$ in **MAGMA 2.5.0 (2019)**, **TCAIRS** in NVIDIA cuSOLVER library.

Performance on One NVIDIA GV100

Haidar et al. (2020). Factor 4 speedup over fp64.



Future Directions

- Mixed precision algorithms. LU: **Lopez & Mary (2020)**.
- Hybrid direct/iterative.
- Randomization.
- Understanding growth factor for (partial) pivoting.
- More realistic rounding error bounds: probabilistic results (**Connolly, H & Mary, 2019/20/21**, **Ipsen & Zhou, 2020**): $f(n)u \to \sqrt{f(n)}u$.
- Construction of test matrices, e.g., for HPL-AI Benchmark (**H & Fasi,** 2021[2]).
- Exploiting structure.
- Using AI?

Slides at `https://bit.ly/dongarra70`

Daily Mirror, 1952

INTO THE TITAN-SIZED WORLD OF SUPERCOMPUTING

By **Amanda Cleary Eastep**

The fastest computer in the United States fills a room the size of a basketball court and generates an electricity bill estimated at $9 million per year. Behind this titan-sized technology is the combined brainpower of a scientific team at the largest U.S. Department of Energy laboratory—Oak Ridge National Laboratory (ORNL)—which includes Jack Dongarra (M.S. CS ’73).

In his ORNL role, Dongarra helps develop methods for solving common problems that occur in scientific computing by designing algorithms and software that can solve numerical linear algebra problems for the next

Manchester, July 2, 2010

References I

Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé.
Five-precision GMRES-based iterative refinement.
MIMS EPrint 2021.5, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, April 2021.
21 pp.

Erin Carson and Nicholas J. Higham.
A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems.
SIAM J. Sci. Comput., 39(6):A2834–A2856, 2017.

References II

- Erin Carson and Nicholas J. Higham.
  Accelerating the solution of linear systems by iterative refinement in three precisions.
  SIAM J. Sci. Comput., 40(2):A817–A847, 2018.

- Michael P. Connolly, Nicholas J. Higham, and Theo Mary.
  Stochastic rounding and its probabilistic backward error analysis.
  SIAM J. Sci. Comput., 43(1):A566–A585, 2021.

- J. J. Dongarra and A. R. Hinds.
  Unrolling loops in FORTRAN.
  Software—Practice and Experience, 9:216–226, 1979.

References III

Massimiliano Fasi and Nicholas J. Higham.
Generating extreme-scale matrices with specified singular values or condition numbers.
SIAM J. Sci. Comput., 43(1):A663–A684, 2021.

Massimiliano Fasi and Nicholas J. Higham.
Matrices with tunable infinity-norm condition number and no need for pivoting in LU factorization.
SIAM J. Matrix Anal. Appl., 42(1):417–435, 2021.

References IV

Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham.
Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC18 (Dallas, TX), Piscataway, NJ, USA, 2018, pages 47:1–47:11. IEEE.

Desmond J. Higham, Nicholas J. Higham, and Srikara Pranesh.
Random matrices generating large growth in LU factorization with pivoting.
SIAM J. Matrix Anal. Appl., 42(1):185–201, 2021.

References V

Nicholas J. Higham.
Matrix computations in Basic on a microcomputer.
Numerical Analysis Report No. 101, Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK, June 1985.
62 pp.
Reissued as MIMS EPrint 2013.51, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, October 2013.

Nicholas J. Higham.
Matrix computations in Basic on a microcomputer.
IMA Bulletin, 22(1/2):13–20, 1986.

References VI

Nicholas J. Higham and Theo Mary.
A new approach to probabilistic rounding error analysis.
SIAM J. Sci. Comput., 41(5):A2815–A2835, 2019.

Nicholas J. Higham and Theo Mary.
Sharper probabilistic backward error analysis for basic linear algebra kernels with random data.
SIAM J. Sci. Comput., 42(5):A3427–A3446, 2020.

References VII

Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra.
Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems).
In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, November 2006.

Florent Lopez and Theo Mary.
Mixed precision LU factorization on GPU tensor cores: Reducing data movement and memory footprint.
MIMS EPrint 2020.20, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, September 2020.
20 pp.

References VIII

Cleve B. Moler.
Matrix computations with Fortran and paging.
Comm. ACM, 15(4):268–270, 1972.

J. H. Wilkinson.
Progress report on the Automatic Computing Engine.
Report MA/17/1024, Mathematics Division, Department of Scientific and Industrial Research, National Physical Laboratory, Teddington, UK, April 1948.
127 pp.