

# Επιστημονικός Υπολογισμός

Ε.Γαλλόπουλος

ΤΜΗΥΠ, Π. Πατρών

Διάλεξη 9: 6 Δεκεμβρίου 2017

## 1 Numerical Software for Algebra

## 2 Τεχνικές βελτίωσης αποτελεσμάτων

- Επαναληπτική εκλέπτυνση
- Μειονεκτήματα της οδήγησης

## 3 Ειδικά μητρώα: Συμμετρικά θετικά ορισμένα, Διαγώνια κυρίαρχα

## 4 Επίλυση προβλημάτων ελαχίστων τετραγώνων - Παραγοντοποίηση QR (επανάληψη)

- Ανακλαστές Householder (επανάληψη)

- LINPACK, EISPACK (1970s)
- LAPACK (1990+)
- ScaLAPACK (1990+)
- PLAPACK
- Υλοποιήσεις από κατασκευαστές (π.χ. Intel, IBM, NVIDIA)
- LAPACK και μετροπρογράμματα

# Existing Math Software - Dense LA

DIRECT SOLVERS	License	Support	Type		Language			Mode		
			Real	Complex	F77	C	C++	Shared	GPU	Dist
<a href="#">Eigen</a>	<a href="#">Mozilla</a>	yes	X	X			X	X		
<a href="#">Elemental</a>	<a href="#">BSD</a>	yes	X	X			X			M
<a href="#">FLAME</a>	<a href="#">LGPL</a>	yes	X	X	X	X		X		
<a href="#">FLENS</a>	<a href="#">BSD</a>	yes	X	X			X	X		
<a href="#">LAPACK</a>	<a href="#">BSD</a>	yes	X	X	X	X		X		
<a href="#">LAPACK95</a>	<a href="#">BSD</a>	yes	X	X	F95			X		
<a href="#">MAGMA</a>	<a href="#">BSD</a>	yes	X	X	X	X		X	C/O/X	
<a href="#">NAPACK</a>	<a href="#">BSD</a>	yes	X		X			X		
<a href="#">PLAPACK</a>	?	no	X	X	X	X				M
<a href="#">PLASMA</a>	<a href="#">BSD</a>	yes	X	X	X	X		X		
<a href="#">PRISM</a>	?	no	X		X			X		M
<a href="#">rejtrix</a>	<a href="#">by-nc-sa</a>	yes	X				X	X		
<a href="#">ScaLAPACK</a>	<a href="#">BSD</a>	yes	X	X	X	X				M/P
<a href="#">Trilinos/Pliris</a>	<a href="#">BSD</a>	yes	X	X		X	X			M
<a href="#">ViennaCL</a>	<a href="#">MIT</a>	yes	X				X	X	C/O/X	

Από παρουσίαση του Jack Dongarra

## Looking back at dense linear algebra software

Piotr Luszczek<sup>a</sup>, Jakub Kurzak<sup>a</sup>, Jack Dongarra<sup>a,b,c,\*</sup>

<sup>a</sup> University of Tennessee Knoxville, United States

<sup>b</sup> Oak Ridge National Laboratory, United States

<sup>c</sup> University of Manchester, United Kingdom



### HIGHLIGHTS

- Growing gap of processor–memory communication
- CPU/GPU hybridization creates challenging design co
- Loop restructuring methods include vectorization, ch
- Scheduling techniques cover bulk-synchronous, SPM
- The effects of increased parallelism on numerical pro

### ARTICLE INFO

Article history:  
Received 17 July 2013  
Accepted 28 October 2013  
Available online 4 December 2013

### ABSTRACT

Over th  
demanc  
Most of

## Numerical linear algebra on emerging architectures: the PLASMA and MAGMA projects

Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, Stanimire Tomov<sup>1</sup>

Department of Electrical Engineering and Computer Science, University of Tennessee, USA

E-mail: eagullo,dongarra,hadri,kurzak,ltaief,luszczek,tomov@eecs.utk.edu

E-mail: demmel@cs.berkeley.edu

E-mail: julien.langou@ucdenver.edu

### Elemental: A New Framework for Distributed Memory Dense Matrix Computations

JACK POULSON, BRYAN MARKER, and ROBERT A. VAN DE GEIJN,

University of Texas at Austin

JEFF R. HAMMOND and NICHOLS A. ROMERO,

Argonne Leadership Computing Facility

Parallelizing dense matrix computations to distributed memory architectures is a well-studied subject and generally considered to be among the best understood domains of parallel computing. Two packages, developed in the mid 1990s, still enjoy regular use: ScaLAPACK and PLAPACK. With the advent of many-core architectures, which may very well take the shape of distributed memory architectures within a single processor, these packages must be revisited since the traditional MPI-based approaches will likely need to be extended. Thus, this is a good time to review lessons learned since the introduction of these two packages and to propose a simple yet effective alternative. Preliminary performance results show the new solution achieves competitive, if not superior, performance on large clusters.

Continuing use of multi-core architectures and graphics processing units require sometimes even a redesign of the established algorithms in order to allism. Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) and Multicore Architectures (MAGMA) are two projects that aim to provide a library across a wide range of multi-core architectures and hybrid systems. This paper presents a comparative study of PLASMA's performance against established libraries and preliminary results of MAGMA on hybrid multi-core and GPU systems.

*Η πιο σημαντική βιβλιοθήκη με κώδικες αιχμής για την επίλυση των θεμελιωδών προβλημάτων γραμμικής άλγεβρας που χρησιμοποιείται ευρύτατα στον επιστημονικό υπολογισμό είναι η LAPACK. Πολλά περιβάλλοντα του ΕΥ στηρίζονται σ' αυτήν (MATLAB, Octave, Scilab) ενώ μερικές ρουτίνες της χρησιμοποιούνται ως μετροπρόγραμμα για την αξιολόγηση επίδοσης υπολογιστικών συστημάτων. Το εγχειρίδιο αποτελεί βασική πηγή του Επιστημονικού Υπολογισμού.*

✓ Με την LAPACK λύνονται συστήματα γραμμικών εξισώσεων (ΑΓΑ.1), γραμμιά προβλήματα ελαχίστων τετραγώνων, προβλήματα ιδιοτιμών, και προβλήματα ιδιαζουσών τιμών. Η LAPACK εκτελεί πολλούς συναφείς υπολογισμούς, όπως παραγοντοποιήσεις μητρώων και εκτίμηση δεικτών κατάστασης.

✓ Παρέχονται ρουτίνες για υπολογισμούς με πυκνά μητρώα και μητρώα ζώνης αλλά όχι για γενικά αραιά μητρώα. Επίσης υπάρχει η δυνατότητα για υπολογισμούς με πραγματική ή μιγαδική α.κ.υ.

✓ Τα αρχεία αριθμούν περισσότερες από 800,000 γραμμές σχολίων και κώδικα Fortran.

- ✓ Η πιο πρόσφατη έκδοσή LAPACK 3.6.0 (11/2015).
- ✓ Αποτελείται από **ρουτίνες οδηγούς** για την επίλυση τυπικών προβλημάτων, **υπολογιστικές ρουτίνες** με πιο εξειδικευμένο στόχο, και **βοηθητικές ρουτίνες** για υπολογισμούς χαμηλότερου επιπέδου. Οι οδηγοί καλούν μια σειρά από υπολογιστικές ρουτίνες. Οι υπολογιστικές ρουτίνες φέρουν σε πέρας ένα ευρύτερο φάσμα υπολογισμών από αυτό που αναφέρεται στους οδηγούς. Οι βοηθητικές ρουτίνες είναι χρήσιμες για πολλές άλλες εφαρμογές.
- ✓ Το αρχείο διανομής περιέχει τον κώδικα πηγής (Fortran), προγράμματα ελέγχου, και προγράμματα για την εκτίμηση της απόδοσής τους στο υπολογιστικό σύστημα που χρησιμοποιείται.
- ✓ Το αρχείο διανομής περιέχει κώδικες αναφοράς για τα BLAS επιπέδων 1, 2 και 3. Πρέπει όμως να σημειωθεί ότι οι κώδικες αυτοί πρέπει να χρησιμοποιούνται μόνον αν δεν υπάρχει εναλλακτική, πιο αποδοτική, υλοποίηση για το υπολογιστικό σας σύστημα (πράγμα σπάνιο)! Μην ξεχνάτε ότι η απόδοση της LAPACK καθορίζεται σε μεγάλο βαθμό από την απόδοση των BLAS, επομένως είναι κρίσιμο να βρείτε και να εγκαταστήσετε αποδοτικούς κώδικες για τα BLAS αντί για τους κώδικες αναφοράς. **Τα BLAS δεν είναι μέρος της LAPACK.**
- ✓ Η LAPACK στη MATLAB, στην Intel MKL, στην IBM SSL, στην NVIDIA.

Όπως και με τα BLAS τα ονόματα έχουν τη μορφή  $SYYZZZ$  όπου  $S$  δηλώνει τον αριθμητικό τύπο των δεδομένων και της αριθμητικής που θα ακολουθηθεί,  $YY$  το είδος του μητρώου και τον τρόπο αποθήκευσής του,  $ZZZ$  την πράξη γραμμικής άλγεβρας που εκτελείται.

$X$	τύπος στοιχείων	$ZZZ$	υπολογισμός
S	REAL	TRF	παραγοντοποίηση
D	DOUBLE PRECISION	TRS	επίλυση από παραγοντοποίηση
C	COMPLEX	COND	εκτίμηση δείκτη κατάστασης
Z	COMPLEX*16	RFS	εκλέπνωση λύσης
		TRI	αντιστροφή μέσω παραγοντοποίησης
		EQU	κλιμάκωση



YY	είδος μητρώου / δομή πίνακα	YY	είδος μητρώου / δομή πίνακα
GE	γενικός	PO	συμ. (ερμιτιανός) θετ. ορισμ.
TR	τριγωνικός	PP	στιβαγμένος συμ. (ερμιτιανός) θετ. ορισμ.
TB	τριγωνικός ζώνης	PB	συμ. (ερμιτιανός) θετ. ορισμ. ζώνης
TP	στιβαγμένος τριγωνικός	PT	συμ. (ερμιτιανός) θετ. ορισμ. τριδιαγώνιος
GB	γενικός ζώνης	SY	συμμ. (μιγ.) αόριστος
GT	γενικός τριδιαγώνιος	SP	στιβαγμένος συμμ. (μιγ.) αόριστος
HE	ερμιτιανός αόριστος		

## Μεγάλη σημασία δόθηκε:

- Στην ακρίβεια: επιστρέφονται δείκτες που πληροφορούν το χρήστη για την αξιοπιστία των αποτελεσμάτων (π.χ. δείκτες κατάστασης, εμπρός σφάλμα, πίσω σφάλμα, κ.ά.)
- Στην αποτελεσματικότητα και ταχύτητα: Χρήση BLAS-3, αξιοποίηση τύπου μητρώου για φθηνότερη αποθήκευση, κ.ά.

## Ειδικότερα σχετικά με το τελευταίο:

Ανάλογα με το είδος του μητρώου, μπορεί να χρησιμοποιηθεί και δομή αποθήκευσης που αξιοποιεί τα συγκεκριμένα χαρακτηριστικά. Ειδικότερα:

## Για τετραγωνικό μητρώο $n \times n$ :

**γενική δομή** συνηθισμένη αποθήκευση σε διδιάστατο πίνακα  $n \times n$

**συμμετρική, ερμιτιανή ή τριγωνική δομή** σπιβαγμένη αποθήκευση ανά στήλη

**δομή ζώνης** αποθήκευσης ζώνης

**τριδιαγώνια ή διδιαγώνια δομή** αποθήκευση σε 2 ή 3 *μονοδιάστατους* πίνακες (διανύσματα)

Για παράδειγμα, η ανά στήλη σπιραγμένη αποθήκευση ενός κάτω τριγωνικού μητρώου  $A$  είναι όπως στο  $AP$ .

$$A = \begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix}$$
$$AP = [\alpha_{11}, \alpha_{21}, \alpha_{31}, \alpha_{41}, \alpha_{22}, \alpha_{32}, \alpha_{42}, \alpha_{33}, \alpha_{43}, \alpha_{44}]$$

Για περισσότερες πληροφορίες δείτε

<http://www.netlib.org/lapack/lug/node121.html>

Παρέχονται δύο τύποι οδηγών για γραμμικά συστήματα :

- 1 απλός οδηγός (επίθεμα ονόματος  $-SV$ ), που λύνει το  $AX = B$  μέσω παραγοντοποίησης του  $A$  και εγγραφής της λύσης  $X$  επί του  $B$ .
- 2 έμπειρος οδηγός (επίθεμα ονόματος  $-SVX$ ), που επιπλέον υπολογίζει τα εξής (μερικά προαιρετικά): επίλυση  $A^T X = B$  ή  $A^* X = B$  (εκτός αν το  $A$  είναι συμμετρικό ή ερμιτιανό); εκτίμηση δείκτη κατάστασης του  $A$ , έλεγχος αν το μητρώο είναι σχεδόν μη αντιστρέψιμο (μοναδικό), έλεγχος μεγέθυνσης του παράγοντα οδήγησης; εκλέπτυνση της λύσης και υπολογισμός φραγμάτων για το εμπρός και πίσω σφάλμα; ζυγοστάθμιση του  $A$  αν έχει διαπιστωθεί κακή κλιμάκωση.

Οι έμπειροι οδηγοί χρειάζονται περί το διπλάσιο χώρο αποθήκευσης και επιπλέον χρόνο. Αμφότεροι οδηγοί μπορούν να λύσουν για πολλά δεξιά μέλη (δηλ.  $B$  πολλών στηλών). Οι οδηγοί διαφέρουν ανάλογα με το είδος του τύπου (αποθήκευσης) του  $A$ .

**DGESV** driver για την επίλυση γενικού πραγματικού συστήματος σε αριθμητική διπλής ακρίβειας.

**DGESVX** expert driver για το παραπάνω

**DGECON** εκτίμηση του αντιστρόφου του δείκτη κατάστασης γενικού πραγματικού μητρώου ως προς την νόρμα 1 ή τη νόρμα μεγίστου χρησιμοποιώντας την *LU* παραγοντοποίηση που προκύπτει από τη ρουτίνα **DGETRF**.

Η αντιστροφή μητρώων επιτελείται μέσω των υπολογιστικών ρουτινών (και όχι των ρουτινών οδήγησης).

It is seldom necessary to compute the inverse of a matrix explicitly, and it is certainly not recommended as a means of solving linear systems.

## Πως μπορούμε να μειώσουμε το σφάλμα;

Κλιμάκωση/ισοστάθμιση ( scaling/equilibration) Κατασκευάζουμε διαγώνια  $P_1, P_2$  ώστε το  $x$  από τη λύση του  $P_1 A P_2 P_2^{-1} x = P_1 b$  να έχει μικρότερο σφάλμα από εκείνο που κανοποιεί  $Ax = b$ ; π.χ. επιλέγοντας  $P_1, P_2$  ώστε  $\kappa(P_1 A P_2) < \kappa(A)$ .

Επαναληπτική εκλέπτυνση (= iterative improvement/refinement): Η ιδέα προέρχεται από την εφαρμογή Newton στο  $f(x) \equiv Ax - b = 0$ .

1.  $x_0 = A^{-1}b$
2.  $k = 1$
3. **repeat**
4.  $r_k = b - Ax_{k-1}$
5.  $z_k = A^{-1}r_k$
6.  $x_k = x_{k-1} + z_k$
7.  $k = k + 1$
8. **until** convergence

- Η ιδέα οφείλεται στον Wilkinson και υπεισέρχεται (και μεταμφιεσμένη) σε πάμπολλες εφαρμογές.

- Αν λύναμε  $A^{-1}r_k$  **ακριβώς**, τότε  $x^{(j)}$  θα ήταν η ακριβής λύση!
- Για ταχύτητα, πρέπει να κάνουμε χρήση της παραγοντοποίησης  $LU$  για τον  $A$ .
- Συνήθως: Επιζητούμε τον υπολογισμό του βήματος (4) με υψηλότερη ακρίβεια από το  $x_0$ .
- Μερικά βήματα μπορεί να επιφέρουν μεγάλη βελτίωση στην ακρίβεια.
- Πρόσφατα προτάθηκε για χρήση σε συστήματα όπως GPUs, FPGAs ως εξής: Να τρέξουν ειδικά τα βήματα (1, 5) σε πολύ γρήγορη μονή ακρίβεια και τα υπόλοιπα σε διπλή.
- Κάτω από ορισμένες συνθήκες μπορεί να έχουμε εμπρός σφάλμα φραγμένο φραγμένο (περίπου) από μικρό πολλαπλάσιο του  $\mathbf{u}_{32}$  είτε από  $\kappa(A)\mathbf{u}_{64}$ .

# Επαναληπτική εκλέπτυνση iterative refinement

- Στο επίκεντρο του ενδιαφέροντος λόγω της νέας γενιάς επεξεργαστών<sup>1</sup> ( Cell, GPUs)
- Ένας πυρήνας STI Cell BE (συνεργασία Sony, Toshiba, IBM, Sony) έχει μέγιστη επίδοση 25 Gflop/s για μονή ακρίβεια αλλά 1.8 Gflop/s για διπλή
- Στόχος να συνδυάσουμε την επίδοση των 32-bits με την ακρίβεια των 64ων στη λύση γραμμικών συστημάτων.
- Η θεωρία ανάλυσης σφάλματος δείχνει πως κάτι τέτοιο είναι εφικτό, κάτω από ορισμένες συνθήκες.

1. (s)  $x_0 = A^{-1}b$  / \* αποθηκεύονται οι παράγοντες  $L, U$
2.  $k = 1$
3. **repeat**
4.  $r_k = b - Ax_{k-1}$
5. (s)  $z_k = A^{-1}r_k$  / \* χρησιμοποιούνται οι παράγοντες  $L, U$
6.  $x_k = x_{k-1} + z_k$
7.  $k = k + 1$
8. **until** convergence

<sup>1</sup> [www.netlib.org/lapack/lawnspdf/lawn175.pdf](http://www.netlib.org/lapack/lawnspdf/lawn175.pdf)



Χρησιμοποιούμε για  $A$  το `inglefpp single (g(20))`, τυχαίο διάνυσμα  $x$  και  $b = Ax$ . Θέτουμε το υπολογισμένο  $\tilde{x} = A^{-1}b$ , και

$$\tilde{x}^{(d)} := \tilde{x} + A^{-1}(b - A\tilde{x})$$

Αποτελέσματα χωρίς και με εκλέπτυνση:

0 βήμ. εκλ.  $\|x - \tilde{x}\| = 3.3462e - 03$

1 βήμ. εκλ.  $\|x - \tilde{x}^{(1)}\| = 3.4757e - 07$

2 βήμ. εκλ.  $\|x - \tilde{x}^{(2)}\| = 9.2238e - 11$

3 βήμ. εκλ.  $\|x - \tilde{x}^{(3)}\| = 1.3467e - 15$

- Κάτω από ορισμένες συνθήκες μπορεί να έχουμε εμπρός σφάλμα φραγμένο (περίπου) από μικρό πολλαπλάσιο του  $\mathbf{u}_{32}$  είτε από  $\kappa(A)\mathbf{u}_{64}$ .

ΠΑΡΑΤΗΡΟΥΜΕ Η εκλέπτυνση αύξησε την ακρίβεια - στη θεωρία, η βελτίωση αναδεικνύεται μέσω της μείωσης του πίσω σφάλματος.

## Θεώρημα (Demmel)

Έστω ότι σε κάθε βήμα της επαναληπτικής εκλέπτυνσης υπολογίζουμε το  $r$  με διπλή ακρίβεια από εκείνη με την οποία εκτελούνται οι υπόλοιποι υπολογισμοί και ότι  $g$  είναι ο συντελεστής αύξησης για το  $A \in \mathbb{R}^{n \times n}$ . Έστω επίσης ότι  $\kappa(A) \epsilon < 1/(3n^3g + 1) < 1$ . Τότε η επαναληπτική εκτέλεση της εκλέπτυνσης συγκλίνει σε διάνυσμα, έστω  $x^{(j)}$ , για το οποίο

$$\frac{\|x^{(j)} - A^{-1}b\|_{\infty}}{\|A^{-1}b\|_{\infty}} = O(\mathbf{u})$$

Αν ο δ.κ.  $\kappa(A)$  δεν είναι μεγάλος, το αποτέλεσμα του υπολογισμού είναι πρακτικά σωστό, δηλ.

$$\frac{\|x - \hat{x}^{(j)}\|_{\infty}}{\|x\|_{\infty}} \approx \mathbf{u}$$

Αν χρησιμοποιήσουμε αριθμητική **ενός μόνον είδους ακρίβειας** (π.χ. αποκλειστικά double precision), μπορεί να βελτιωθεί σημαντικά το πίσω σφάλμα ανά στοιχείο (componentwise backward error) (Skeel'80)

- με ένα βήμα εκλέπτυνσης η υπολογισθείσα λύση  $\hat{x}^{(1)}$  είναι τέτοια ώστε

$$(A + \Delta A)\hat{x}^{(1)} = b + \delta b$$

όπου  $|\Delta a_{ij}| = O(u)|a_{ij}|, |\delta b_i| = O(u)|b_i|.$

- με αρκετά βήματα εκλέπτυνσης

$$\frac{\|x - \hat{x}\|_{\infty}}{\|x\|_{\infty}} \leq 2nu \frac{\|A^{-1}\| \|A\| \|x\|_{\infty}}{\|x\|_{\infty}}$$

Εκτίμηση δείκτη κατάστασης με ειδικούς αλγόριθμους (Hager, Higham) κόστους μικρότερου από το σχηματισμό των  $A^{-1}$ ,  $\|A\|$ ,  $\|A^{-1}\|$ .

Εκτίμηση πίσω σφάλματος

$$\omega_{A,b}(\hat{x}) = \max_i \frac{|r|_i}{(|A||\hat{x}| + |b|)_i}$$

Εκτίμηση forward σφάλματος

$$\omega_{A,b}(\hat{x}) = \omega_{A,b} \frac{\| |A^{-1}| (|\hat{r}| + \gamma_{n+1} (|A||\hat{x}| + |b|)) \|_{\infty}}{\|\hat{x}\|_{\infty}}$$



40th Anniversary Issue

## Accelerating scientific computations with mixed precision algorithms<sup>\*</sup>

Marc Baboulin<sup>a</sup>, Alfredo Buttari<sup>b</sup>, Jack Dongarra<sup>c,d,e</sup>, Jakub Kurzak<sup>c,e</sup>, Julie Langou<sup>c</sup>, Julien Langou<sup>f</sup>, Piotr Luszczek<sup>g</sup>, Stanimire Tomov<sup>h</sup>

On modern architectures, the performance of 32-bit operations is often at least twice as fast as the performance of 64-bit operations. By using a combination of 32-bit and 64-bit floating point arithmetic, the performance of many dense and sparse linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. The approach presented here can apply not only to conventional processors but also to other technologies such as Field Programmable Gate Arrays (FPGA), Graphical Processing Units (GPU), and the STI Cell BE processor. Results on modern processor architectures and the STI Cell BE are presented.

### ACCELERATING THE SOLUTION OF LINEAR SYSTEMS BY ITERATIVE REFINEMENT IN THREE PRECISIONS<sup>\*</sup>

ERIN CARSON<sup>†</sup> AND NICHOLAS J. HIGHAM<sup>‡</sup>

**Abstract.** We propose a general algorithm for solving a  $n \times n$  nonsingular linear system  $Ax = b$  based on iterative refinement with three precisions. The working precision is combined with possibly different precisions for solving for the correction term and for computing the residuals. Via rounding error analysis of the algorithm we derive sufficient conditions for convergence and bounds for the attainable normwise forward error and normwise and componentwise backward errors. Our results generalize and unify many existing rounding error analyses for iterative refinement. With single precision as the working precision, we show that by using LU factorization in IEEE half precision as the solver and calculating the residuals in double precision it is possible to solve  $Ax = b$  to full single precision accuracy for condition numbers  $\kappa_2(A) \leq 10^4$ , with the  $O(n^3)$  part of the computations carried out entirely in half precision. We show further that by solving the correction equations by GMRES preconditioned by the LU factors the restriction on the condition number can be weakened to  $\kappa_2(A) \leq 10^8$ , although in general there is no guarantee that GMRES will converge quickly. Taking for comparison a standard  $Ax = b$  solver that uses LU factorization in single precision, these results suggest that on architectures for which half precision is efficiently implemented it will be possible to solve certain linear systems  $Ax = b$  up to twice as fast and to greater accuracy. Analogous results

## MIXED PRECISION ITERATIVE REFINEMENT TECHNIQUES FOR THE SOLUTION OF DENSE LINEAR SYSTEMS

Alfredo Buttari<sup>1</sup>  
Jack Dongarra<sup>1,2</sup>  
Julie Langou<sup>1</sup>  
Julien Langou<sup>3</sup>  
Piotr Luszczek<sup>1</sup>  
Jakub Kurzak<sup>1</sup>

### Abstract

By using a combination of 32-bit and 64-bit floating point arithmetic, the performance of many dense and sparse linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. The approach presented here can apply not only to conventional processors but also to exotic technologies such as Field Programmable Gate Arrays (FPGA), Graphical Processing Units (GPU), and the Cell BE processor. Results on modern processor architectures and the Cell BE are presented.

## Exploiting Mixed Precision Floating Point Hardware in Scientific Computations

Download

Alfredo BUTTARI<sup>a</sup> Jack DONGARRA<sup>a,b,d</sup> Jakub KURZAK<sup>a</sup> Julie LANGOU<sup>a</sup>  
Julien LANGOU<sup>a</sup> Piotr LUSZCZEK<sup>a</sup> Stanimire TOMOV<sup>a</sup>

<sup>a</sup> Department of Computer Science, University of Tennessee Knoxville

<sup>b</sup> Oak Ridge National Laboratory

<sup>c</sup> University of Colorado at Denver and Health Sciences Center

<sup>d</sup> University of Manchester

**Abstract.** By using a combination of 32-bit and 64-bit floating point arithmetic, the performance of many dense and sparse linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. The approach presented here can apply not only to conventional processors but also to exotic technologies such as Field Programmable Gate Arrays (FPGA), Graphical Processing Units (GPU), and the Cell BE processor. Results on modern processor architectures and the Cell BE are presented.



40th Anniversary Issue

## Accelerating scientific computations with mixed precision algorithms <sup>☆</sup>

Marc Baboulin <sup>a</sup>, Alfredo Buttari <sup>b</sup>, Jack Dongarra <sup>c,d,e</sup>, Jakub Kurzak <sup>c,g</sup>, Julie Langou <sup>c</sup>, Julien Langou <sup>f</sup>, Piotr Luszczek <sup>g</sup>, Stanimire Tomov <sup>h</sup>

On modern architectures, the performance of 64-bit operations is significantly enhanced while maintaining the 64-bit accuracy only to conventional processors (FPGA), Graphical Processing Units (GPU), and the STI Cell architecture and the Cell BE processor.

### ACCELERATING ITERATIVE

**Abstract.** We present a new approach for accelerating iterative refinement based on iterative refinement with different precisions for the forward and backward error analysis of the attainable normwise forward error and normwise and componentwise backward errors. Our results generalize and unify many existing rounding error analyses for iterative refinement. With single precision as the working precision, we show that by using LU factorization in IEEE half precision as the solver and calculating the residuals in double precision it is possible to solve  $Ax = b$  to full single precision accuracy for condition numbers  $\kappa_2(A) \leq 10^4$ , with the  $O(n^3)$  part of the computations carried out entirely in half precision. We show further that by solving the correction equations by GMRES preconditioned by the LU factors the restriction on the condition number can be weakened to  $\kappa_2(A) \leq 10^6$ , although in general there is no guarantee that GMRES will converge quickly. Taking for comparison a standard  $Ax = b$  solver that uses LU factorization in single precision, these results suggest that on architectures for which half precision is efficiently implemented it will be possible to solve certain linear systems  $Ax = b$  up to twice as fast and to greater accuracy. Analogous results are given with double precision as the working precision.

## MIXED PRECISION ITERATIVE REFINEMENT TECHNIQUES FOR THE SOLUTION OF DENSE LINEAR SYSTEMS

Alfredo Buttari <sup>1</sup>  
Jack Dongarra <sup>1,2</sup>  
Julie Langou <sup>3</sup>  
Julien Langou <sup>3</sup>  
Piotr Luszczek <sup>1</sup>  
Jakub Kurzak <sup>1</sup>

## The Rise of Multiprecision Computations

Nick Higham  
School of Mathematics  
The University of Manchester

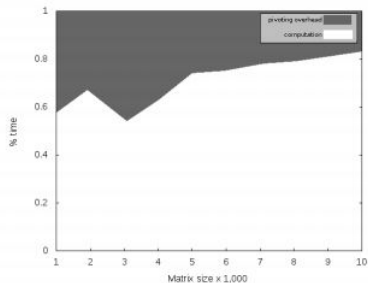
64-bit and 64-bit floating point operations on many dense and sparse matrices can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. This approach can apply not only to conventional processors but also to exotic technologies such as Field Programmable Gate Arrays (FPGA), Graphical Processing Units (GPU), and the Cell BE processor. Results on modern processor architectures and the Cell BE are presented.

64-bit Floating  
Scientific  
IS

JULIE LANGOU <sup>a</sup>, PIOTR LUSZCZEK <sup>a</sup>, STANIMIRE TOMOV <sup>a</sup>,  
JULIEN LANGOU <sup>c</sup>, PIOTR LUSZCZEK <sup>a</sup> and STANIMIRE TOMOV <sup>a</sup>  
<sup>a</sup> Department of Computer Science, University of Tennessee Knoxville  
<sup>b</sup> Oak Ridge National Laboratory  
<sup>c</sup> University of Colorado at Denver and Health Sciences Center  
<sup>d</sup> University of Manchester

**Abstract.** By using a combination of 32-bit and 64-bit floating point arithmetic, the performance of many dense and sparse linear algebra algorithms can be significantly enhanced while maintaining the 64-bit accuracy of the resulting solution. The approach presented here can apply not only to conventional processors but also to exotic technologies such as Field Programmable Gate Arrays (FPGA), Graphical Processing Units (GPU), and the Cell BE processor. Results on modern processor architectures and the Cell BE are presented.

Από την εργασία Accelerating linear system solutions using randomization techniques των Baboulin, Dongarra, et al., 2011.



**Fig. 1.** Cost of pivoting in LU factorization (CPU 1 × Quad-Core Intel Core2 Processor Q9300 @ 2.50 GHz GPU C2050 — 14 Multiprocessors ( × 32 CUDA cores) @ 1.15 GHz).

- Συνήθως, η παραγοντοποίηση  $LU$  με μερική οδήγηση (δηλ  $PA = LU$ ) συνεπάγεται μέτριο συντελεστή αύξησης και μέτριο  $\|\hat{U}\|$ , οπότε μπορεί να θεωρηθεί πίσω ευσταθής.
- Διαφορετικά πρέπει να ζητήσουμε άλλο αλγόριθμο.
- Μια επιλογή είναι η GECP ( $LU$  με πλήρη οδήγηση (ΠΟ)) στην οποία παράγονται τριγωνικά  $L$ ,  $U$  και μητρώα μετάθεσης  $P$ ,  $Q$  τέτοια ώστε  $PAQ = LU$  (το  $P$  μεταθέτει γραμμές, το  $Q$  μεταθέτει στήλες).
- Η GECP έχει 1) μικρότερο συντελεστή αύξησης από τις άλλες μορφές οδήγησης, άρα μεγαλύτερη πίσω ευστάθεια, 2) Επιβραδύνει γιατί απαιτούνται  $O(n^3)$  συγκρίσεις.
- Ευτυχώς η GEPP είναι σχεδόν πάντα πίσω ευσταθής!!!
- Δύο γενικά μειονεκτήματα κάθε μορφής οδήγησης: α) επιπλέον κόστος, β) απώλεια χρησιμων δομικών ιδιοτήτων του αρχικού μητρώου (όπως: συμμετρία, τριγωνικότητα, τριδιαγωνιότητα, κ.λπ.)
- Πώς/πότε αποφεύγεται η οδήγηση?



Η οδήγηση μπορεί να επιφέρει καταστροφή προϋπάρχουσας δομής του μητρώου (συμμετρία, τριγωνικότητα, τριδιαγωνιότητα, κ.λπ.)

- Για παράδειγμα, μπορεί  $A = A^T$  αλλά αν  $P$  είναι μη τετριμμένο μητρώο μετάθεσης (δηλ.  $P \neq I$ ) τότε  $PA \neq (PA)^T$ .
- Κλασικό παράδειγμα ((καταστροφικής)) επίδρασης της οδήγησης (και όχι μόνο):

A =

1	0	0	1
2	1	0	0
3	0	1	0
4	1	1	1

[L,U]=gep(A) ; [L,U] % gep      Higham (diagonal pivoting)

ans =

1	0	0	0	1	0	0	1
2	1	0	0	0	1	0	-2
3	0	1	0	0	0	1	-3
4	1	1	1	0	0	0	2

[L,U,p]=lu(A, 'vector') ; [L,U,p']

ans =

1.00	0	0	0	4.00	1.00	1.00	1.00	4
0.75	1.00	0	0	0	-0.75	0.25	-0.75	3
0.50	-0.67	1.00	0	0	0	-0.33	-1.00	2
0.25	0.33	1.00	1.00	0	0	0	2.00	1

A =

0.0001	0	0	1.0000
2.0000	1.0000	0	0
3.0000	0	1.0000	0
4.0000	1.0000	1.0000	1.0000

[L,U]=gep(A) ; [L,U]

ans = 1.0e+004 \*

0.0001	0	0	0	0.0000	0	0	0.0001
2.0000	0.0001	0	0	0	0.0001	0	-2.0000
3.0000	0	0.0001	0	0	0	0.0001	-3.0000
4.0000	0.0001	0.0001	0.0001	0	0	0	1.0001

[L,U,p]=lu(A, 'vector') ; [L,U,p']

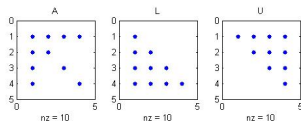
ans =

1.00	0	0	0	4.00	1.00	1.00	1.0000	4
0.75	1.00	0	0	0	-0.75	0.25	-0.7500	3
0.5000	-0.67	1.00	0	0	0	-0.33	-1.0000	2
0.0000	0.00	0.0001	1.00	0	0	0	1.0001	1

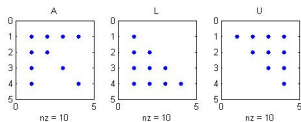
## Συμπεράσματα

Η οδηγηση είναι απαραίτητη αλλά μπορεί να καταστρέψει την προϋπάρχουσα δομή του μητρώου (π.χ. συμμετρία, αραιότητα) και να επιβαρύνει το κόστος.

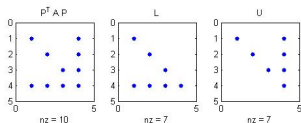
Έστω  $A$  όπως παρακάτω και ότι δεν χρειάζεται οδήνηση:



Έστω  $A$  όπως παρακάτω και ότι δεν χρειάζεται οδήνηση:



Μετά από **συμμετρικές** μεταθέσεις γραμμών και στηλών  $\tilde{A} \leftarrow PAP^T$  και υπολογίζοντας τους παράγοντες  $L$ ,  $U$  (στη συγκεκριμένη περίπτωση η μετάθεση είναι απλά  $A(4:-1:1, 4:-1:1)$ )



Διαβάστε την εργασία:

MATHEMATICS OF COMPUTATION, VOLUME 28, NUMBER 126, APRIL 1974, PAGES 537–542

## Modifying Pivot Elements in Gaussian Elimination\*

By G. W. Stewart

**Abstract.** The rounding-error analysis of Gaussian elimination shows that the method is stable only when the elements of the matrix do not grow excessively in the course of the reduction. Usually such growth is prevented by interchanging rows and columns of the matrix so that the pivot element is acceptably large. In this paper the alternative of simply altering the pivot element is examined. The alteration, which amounts to a rank one modification of the matrix, is undone at a later stage by means of the well-known formula for the inverse of a modified matrix. The technique should prove useful in applications in which the pivoting strategy has been fixed, say to preserve sparseness in the reduction.

Δείτε (GV13, Section 2.1.4) Εξαιρετικά χρήσιμος τύπος με πάρα πολλές εφαρμογές (και στο boosting):

Αν  $A \in \mathbb{R}^{n \times n}$ ,  $U, V \in \mathbb{R}^{n \times k}$  και τα  $A, A + UV^T$  είναι αντιστρέψιμα, τότε ισχύει ότι ((GV13))

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}$$

Γενική ιδέα: Αν τροποποιήσουμε ένα μητρώο κατά τάξη  $k$  το αντίστροφο του νέου μητρώου μπορεί να γραφτεί ως άθροισμα του αντιστρόφου του αρχικού και ενός μητρώου τάξης  $k$  σύμφωνα με τον παραπάνω τύπο. Προσέξτε ότι πολλοί όροι επαναλαμβάνονται!

- Πολλές εφαρμογές στον Επιστημονικό Υπολογισμό, Στατιστική, Θεωρία Ελέγχου, κ.λπ. βλ. survey(Hag89) αλλά και (Gal85)(GPS16)(Yip86)(Rie92)(BCMM03)(NSA12).

# Ειδικά μητρώα

1) Διαγώνια κυρίαρχα, 2) Συμμετρικά θετικά ορισμένα

Για ορισμένες κατηγορίες μητρώων μπορούμε να αποφύγουμε την οδήγηση. α) π.χ. επειδή το διαγώνιο στοιχείο συμβαίνει και είναι το μέγιστο της στήλης, ή β) γιατί χωρίς να χρησιμοποιήσουμε οδήγηση, ο συντελεστής αύξησης είναι μικρός (π.χ. 1 ή 2).

**Συμμετρικά θετικά ορισμένα (ΣΘΟ):** υπάρχει παραγοντοποίηση  $A = LL^T$  (Cholesky, δείτε σελ. 421, 430 στον Strang) και ο συντελεστής αύξησης είναι  $\rho_n = 1$ .  
ΠΡΟΣΟΧΗ: Το  $L$  δεν έχει κατ' ανάγκη 1 στη διαγώνιο.

**Διαγώνια κυρίαρχα (ΔΚ):** Θεώρημα (Wilkinson): Αν το  $A \in \mathbb{R}^{n \times n}$  είναι ΔΚ κατά στήλες ή γραμμές και είναι αντιστρέψιμο, τότε υπάρχει παραγοντοποίηση  $A = LU$  και ο συντελεστής αύξησης είναι  $\rho_n \leq 2$ . Επομένως, η απαλοιφή Gauss χωρίς οδήγηση είναι ευσταθής.

**Συμμετρικά ΔΚ:** υπάρχει παραγοντοποίηση  $A = LU$  αν ισχύει αυστηρή ΔΚ.

Γιατί ενδιαφερόμαστε για ΔΚ και ΣΘΟ μητρώα? Εμφανίζονται σε πάρα πολλές εφαρμογές: Στατιστική, οικονομετρία, σήματα (μητρώα συσχέτισης, μητρώα συμμεταβλητότητας), επίλυση διαφορικών εξισώσεων, κ.λπ.



- Οι επόμενες διαφάνειες αφορούν στην επίλυση προβλημάτων ελαχίστων τετραγώνων που έχουμε συζητήσει στη Γραμμική Αλγεβρα και στην Αριθμητική Ανάλυση.
- Αυτό που θα συναντήσουμε που είναι διαφορετικό σήμερα είναι την ιδέα της **οδήγησης** στην QR.
- Η βασική αναφορά για την QR θα είναι το βιβλίο του μαθήματος (Golub & van Loan)

## Θεώρημα

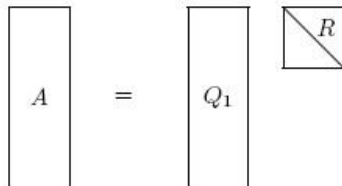
Κάθε μητρώο  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$ , μπορεί να μετασχηματιστεί σε άνω τριγωνική μορφή μέσω ορθογώνιου μετασχηματισμού. Ειδικότερα, υπάρχει παραγοντοποίηση  $A = Q \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$  όπου  $Q \in \mathbb{R}^{m \times m}$  ορθογώνιο και  $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \in \mathbb{R}^{m \times n}$  άνω τριγωνικό. Αν οι στήλες του  $A$  είναι γραμμικά ανεξάρτητες, τότε το  $R_1 \in \mathbb{R}^{n \times n}$  είναι αντιστρέψιμο. Αν επιλέξουμε τα διαγώνια στοιχεία θετικά, οι παράγοντες  $Q, R$  είναι μοναδικοί.

- Μια από τις πιο σημαντικές παραγοντοποιήσεις με πολλές εφαρμογές.
- Υπολογιστικός πυρήνας για 1) Λύση προβλήματος ελαχίστων τετραγώνων. 2) Προσέγγιση ιδιοτιμών/ιδιοδιανυσμάτων (μέθοδος QR)

## Λεπτή (ή οικονομική) παραγοντοποίηση QR (επανάληψη)

$$A = [Q_1, Q_2] \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q_1 R_1$$

Η **λεπτή QR** υπολογίζει  $A = Q_1 R_1$  όπου  $Q_1 \in \mathbb{R}^{m \times n}$  έχει ορθοκανονικές στήλες και το  $R_1$  είναι τετραγωνικό, άνω τριγωνικό.


$$A = Q_1 R_1$$

- Οι στήλες του  $Q_1$  είναι ΟΚ βάση του χώρου στηλών ( $\text{range}(A)$ ).
- Το  $R_1$  περιέχει τον άνω παράγοντα Cholesky του  $A^T A$ .

## Πώς υπολογίζεται η παραγοντοποίηση $QR$ ? (επανάληψη)

- Householder  $\rightarrow$  ανακλάσεις
- Givens  $\rightarrow$  περιστροφές
- Gram-Schmidt  $\rightarrow$  βαθμιαία ορθοκανονικοποίηση

### Υπενθύμιση:

- Αν  $P : \mathbb{R}^m \rightarrow \mathbb{R}^m$  και  $P^2 = P$ , το  $P$  λέγεται **τελεστής προβολής**.
- Αν  $P^2 = P$  και  $P^T = P$  τότε το μητρώο  $P$  λέγεται **τελεστής ορθογώνιας προβολής**.

# Υπολογισμός με ανακλαστές Householder

Οι ανακλαστές Householder είναι μητρώα που ορίζονται ως

$$H = E(u, u; 2/u^\top u) = I - \frac{2}{u^\top u} uu^\top.$$

για κατάλληλα επιλεγμένο διάνυσμα  $u$  και έχουν τις εξής ιδιότητες: 1) στοιχειώδεις μετασχηματισμοί, 2) είναι ορθογώνια μητρώα, 3) είναι συμμετρικά μητρώα.

Επίσης

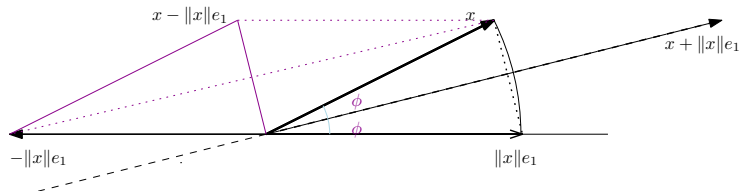
- $H = I - P_u - P_u$  όπου  $P_u$  ο τελεστής ορθ. προβολής στο  $u$ .
- $H^\top = H$  και  $H^\top H = H^2 = I$ .

Επίσης

- Ο πολλαπλασιασμός  $Hx$  **ανακλά** το  $x$  ως προς τον υπόχωρο  $\langle u \rangle^\perp$ .
- Στις 2 διαστάσεις:
- $Hx = \|x\|e_1$  αν το  $x$  ((ανακλαστεί)) ως προς τη **δικοτόμο** των διανυσμάτων  $e_1$  και  $x$
- $Hx = -\|x\|e_1$  αν ανακλαστεί ως προς τη δικοτόμο των  $-e_1$  και  $x$ .
- ... οι δικοτόμοι είναι  $u = x \pm \|x\|e_1$ .

# Οπτικοποίηση (επανάληψη)

στο επίπεδο ( $n = 2$ )



(χρησιμοποιούμε αποκλειστικά νόρμα-2)

$$x = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} \Rightarrow u = x + \|x\| e_1 = \begin{pmatrix} 1 + \sqrt{9} \\ 2 \\ 2 \end{pmatrix}$$

$$Hx = \left(I - 2 \frac{uu^T}{u^T u}\right)x = x - 2u \frac{u^T x}{u^T u} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} - 2 \begin{pmatrix} 4 \\ 2 \\ 2 \end{pmatrix} \frac{12}{24} = \begin{pmatrix} -3 \\ 0 \\ 0 \end{pmatrix} = -\|x\| e_1$$

Μηδενισμός θέσεων  $k + 1 : m$  π.χ.  $m = 4, k = 3, x =$  
$$\begin{pmatrix} 2 \\ 1 \\ -\mathbf{3} \\ \mathbf{4} \end{pmatrix}$$

Βρίσκουμε  $u$  τ.ώ.  $H(u)x =$  
$$\begin{pmatrix} \xi_1 \\ \xi_2 \\ \gamma \\ 0 \end{pmatrix}$$

$$\Rightarrow u = \begin{pmatrix} 0 \\ 0 \\ \xi_{\mathbf{3}} \\ \xi_{\mathbf{4}} \end{pmatrix} - \|x_{3:4}\| e_{\mathbf{3}} = \begin{pmatrix} 0 \\ 0 \\ -3 \\ \frac{\sqrt{25}}{4} \end{pmatrix}$$

$$Hx = \left( I - 2 \frac{uu^T}{u^T u} \right) x = x - 2u \frac{u^T x}{u^T u} = \begin{pmatrix} 2 \\ 1 \\ -3 \\ 4 \end{pmatrix} - 2 \begin{pmatrix} 0 \\ 0 \\ -8 \\ 4 \end{pmatrix} \frac{40}{80} = \begin{pmatrix} 2 \\ 1 \\ 5 \\ 0 \end{pmatrix}$$



- Το πρόσημο επιλέγεται ώστε να αποφεύγεται πιθανή καταστροφική απαλοιφή:

$$u = x + \text{sign}(\xi_1) \|x\|_2 e_1.$$

- το  $H(u)$  είναι ανεξάρτητο της κλιμάκωσης του  $u$ :

$$H = I - \frac{2}{u^\top u} uu^\top = I - \frac{2}{(\alpha u)^\top (\alpha u)} (\alpha u)(\alpha u)^\top.$$

Το  $u$  λέγεται διάνυσμα *Householder*. Καθώς  $H(u) = H(\alpha u)$ , μπορούμε να κανονικοποιήσουμε ώστε  $u(1) = 1$ .

```

function [u] = refl(x)
% computation of normalized
% Householder vector u. Then
% H(u)x = -sign(x(1))*norm(x)*eye(m,1)
[m]=length(x); mu = norm(x); u=x;
if (mu ~ 0)
    beta = x(1) + sign(x(1))*mu;
    u(2:m) = u(2:m)/beta; % normalization
end
u(1) = 1; % normalization    x=0 set u=e_1
end

```

```

function [B] = refl_row(A,u)
%    B = H*A = A-2*u*(u'*A)/(u'*u)
beta = -2/(u'*u);
w_row = beta*u'*A;
B = A + u*w_row;

```

ΠΕΡΙΓΡΑΦΗ: (1)  $v = \text{refl}(x)$ : επιστρέφει κανονικοποιημένο ( $u(1) = 1$ ) διάνυσμα Householder.  $\Omega \approx 3m$ . (2)  $B = \text{refl\_row}(A, u)$  returns  $= -\frac{2}{u^\top u} u u^\top A$ .  $\Omega \approx 4mn$ .

- Οι ανακλαστές είναι στοιχειώδη μητρώα
- ... σπάνια τα κατασκευάζουμε ....
- ... τα διαχειριζόμαστε χρησιμοποιώντας την ειδική τους μορφή (παρόμοια με τα μητρώα Gauss)
- π.χ.  $HA = A - \frac{2}{u^T u} u(A^T u)^T$ ,
- Το μητρώο  $Q$  συνήθως δεν κατασκευάζεται παρά μόνον αν χρειζόμαστε την ορθογώνια βάση ...
- αλλά αποθηκεύουμε τα (κανονικοποιημένα) διανύσματα Householder στο κάτω τριγωνικό μέρος του  $A$ .
- και όταν χρειάζεται να πολλαπλασιάσουμε με το  $Q$  το κάνουμε ((έξυπνα)) με τα διανύσματα  $h_j$
- Η κατασκευή του  $AH$  ή  $HA$  απαιτεί 1  $MV$  και 1 ανανέωση τάξης-1.

## Παραγοντοποίηση QR: μέθοδος Householder

```
function [B] = qr_house(A);  
%      :      A  
%      ,:      B      R      QR  
%      :      Householder  
[m,n]=size(A);  
B = A;  
for j=1:min(m-1,n)  
    u = refl(B(j:m,j)); %      Householder  
    B(j:m,j:n)=refl_row(B(j:m,j:n),u); %  
    B(j+1:m,j)=u(2:m-j+1); %      Householder  
end
```

- $\Omega = 2n^2(m - n/3)$  πράξεις α.κ.υ.
- Αν χρειάζεται το  $Q$ , πρέπει να πολλαπλασιάσουμε τους ανακλαστές. Αυτό επιφέρει περίπου  $2n^2(m - n/3)$  επιπλέον πράξεις, άρα το κόστος διπλασιάζεται.

- Η πληροφορία για τα κανονικοποιημένα διανύσματα  $u_k$  αποθηκεύεται (αν θέλουμε) στο κάτω τριγωνικό τμήμα του  $A$ .
- Στο παραπάνω το  $Q \in \mathbb{R}^{m \times m}$  δεν υπολογίζεται άμεσα αλλά δίδεται μέσω των παραγόντων του ανακλαστών.

$$\begin{bmatrix} \rho_{11} & \rho_{12} & \cdot & \rho_{1n} \\ \eta_2^{(1)} & \rho_{22} & \cdot & \rho_{2n} \\ \cdot & \eta_2^{(1)} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \rho_{n,n} \\ \vdots & \dots & \dots & \eta_{n+1}^{(n)} \\ \eta_m^{(1)} & \eta_m^{(2)} & \cdot & \eta_m^{(n)} \end{bmatrix}$$

# Παράδειγμα (στιγμιότυπο)

Υποθέτουμε  $n < m$

$$H_2 H_1 A = \left( \begin{array}{cc|cc|cc} x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ \hline 0 & 0 & x & x & x & x \\ 0 & 0 & \star & x & x & x \\ 0 & 0 & \star & x & x & x \\ 0 & 0 & \star & x & x & x \end{array} \right)$$

διαλέγουμε τον  $u_3$  'wste na mhdenistoún ta stoixeía tou  $A^{(2)}$  σημειωμένα ως  $(\star)$ ). Τελικά

$$H_n \dots H_1 A = R,$$

και επειδή οι ανακλαστές  $H_j$  είναι ορθογώνιοι και συμμετρικοί,

$$A = H_1 H_2 \dots H_n R = QR$$

όπου  $Q \in \mathbb{R}^{m \times m}$  ορθογώνιο ( $Q^T Q = I$ ) και το  $R \in \mathbb{R}^{m \times n}$  άνω τριγωνικό.

- Σχετικά με την οδήγηση στην QR, διαβάστε (GV13, sect. 5.4.2) (αλγόριθμος Golub-Businger).
- Για επαναληπτική εκλέπτυνση, QR και ελάχιστα τετράγωνα διαβάστε τα παρακάτω:
  - Διαβάστε (GV13, 3.5.3)
  - Διαβάστε (GV13, 3.5.3)
  - Διαβάστε (GV13, sect. 5.1 ως και 5.1.6)
  - Διαβάστε (GV13, sect. 5.2 ως και 5.2.2)
  - Διαβάστε (GV13, sect. 5.3 ως και 5.3.3)
  - Στην επόμενη διάλεξη συνεχίζουμε με block QR και την ιδέα του rank revealing QR.



R Bru, J Cerdán, J Marín, and J Mas.

Preconditioning Sparse Nonsymmetric Linear Systems with the Sherman–Morrison Formula.

*SIAM Journal on Scientific Computing*, 25(2):701–715, January 2003.



D. Bernstein and C. Van Loan.

Rational matrix functions and rank- 1 updates.

*SIAM J. Matrix Anal. Appl.*, (1):145–154.



E Dopazo and M F Martínez-Serrano.

On deriving the Drazin inverse of a modified matrix.

*Linear Alg. Appl.*, 438(4):1678–1687, February 2013.



E. Gallopoulos.

The Massively Parallel Processor for problems in computational fluid dynamics.

*Computer Physics Communications*, 37(1-3):311–316, 1985.



Efstathios Gallopoulos, Bernard Philippe, and Ahmed H. Sameh.

*Parallelism in Matrix Computations*.

Springer, 2016.



G.H. Golub and C.F. Van Loan.

*Matrix Computations*.

Johns Hopkins, 4th edition, 2013.



W. W. Hager.

Updating the inverse of a matrix.

*SIAM Rev.*, 31(2):221–239, Jun. 1989.



Elias D Niño, Adrian Sandu, and Jeffrey L Anderson.

An efficient implementation of the ensemble Kalman filter based on iterative Sherman Morrison formula.



*Procedia - Procedia Computer Science*, 9:1064–1072, 2012.



K.S. Riedel.

A Sherman-Morrison-Woodbury identity for rank augmenting matrices with application to centering.  
*SIAM J. Matrix Anal. Appl.*, 13(2):659–662, Apr. 1992.



Matthias Seeger.

Low Rank Updates for the Cholesky Decomposition.  
Technical report, EPFL, Lausanne, 2004.



G.W. Stewart.

Modifying pivot elements in Gaussian elimination.  
*Math.Comp.*, 28(126):537–542, 1974.



E L Yip.

A note on the stability of solving a rank- $p$  modification of a linear system by the Sherman-Morrison-Woodbury formula.  
*SIAM Journal on Scientific and Statistical Computing*, 7(2):507–513, 1986.