Robust and Accurate Matrix Factorizations

Takeshi Ogita

Department of Mathematical Sciences
Tokyo Woman's Christian University

Dagstuhl Seminar, Nov. 2009

Outline

$\boxed{\text{Purpose}}$ Let A be a real $m \times n$ matrix. We propose algorithms for robust and accurate matrix factorizations of A.

$\boxed{\text{Applications}}$ Many applications (linear systems, determinant, eigenvalues, singular values, etc.) Until now, not sparse case.

The proposed methods allow condition number of A to be extremely large.

$\Longrightarrow$ For example, if using IEEE 754 double precision, we can treat condition number beyond 10^{100}, extremely large case!

Notation

$\mathbb{R}$: set of real numbers
$\mathbb{F}$: set of floating-point numbers
$\mathbf{u}$: unit round-off (IEEE 754 double: $\mathbf{u} = 2^{-53} \approx 10^{-16}$)

For $A = (a_{ij}), B = (b_{ij}) \in \mathbb{R}^{m \times n}$,

- $|A| := (|a_{ij}|) \in \mathbb{R}^{m \times n}$
- $A \le B \quad \Longleftrightarrow \quad a_{ij} \le b_{ij}$ for all (i, j)

Matrix factorizations

Using our concept, we can construct robust algorithms for

- LU and QR factorizations ($A = LU$, $A = QR$ and $A = R^T R$)
- Cholesky factorization ($A = R^T R$)
- Singular value decomposition ($A = U \Sigma V^T$)
- Eigenvalue decomposition ($A = VDV^T$)
- and more (maybe)

Condition number

For a real square matrix A, the condition number of A is defined by

$$\kappa(A) := \|A\| \cdot \|A^{-1}\|.$$

$\Longrightarrow$ It is well-known that $\kappa(A)$ indicates the difficulty of the problem.
$\Longrightarrow$ For example, consider solving a linear system $Ax = b$. Let $x^* := A^{-1}b$. For $Ay = b + \delta b$ and $\Delta x = y - x$,

$$\frac{\|\Delta x\|}{\|x^*\|} \leq \kappa(A) \frac{\|\Delta b\|}{\|b\|}.$$

IEEE standard 754 for double precision floating-point arithmetic

$\Longrightarrow$ Relative precision (unit round-off): $\mathbf{u} \approx 1.11 \times 10^{-16}$

$\Longrightarrow$ If condition number is greater than 10^{16} ($\kappa(A) > \mathbf{u}^{-1}$), then the computed result may have no correct digit.

$\Longrightarrow$ Limit of working precision (e.g. working prec. = double prec.)

$\Longrightarrow$ We call such a case "ill-conditioned problem".

Accuracy of matrix factors

Difficult to know how accurate and useful the matrix factors are.

$\boxed{\text{What about residual?}}$ Suppose $\|A\| \approx 1$ for simplicity.

LU, QR : $\|A - LU\| = \mathcal{O}(\mathbf{u}), \quad \|A - QR\| = \mathcal{O}(\mathbf{u})$
SVD : $\|A - U\Sigma V^T\| \approx \|AV - U\Sigma\| = \mathcal{O}(\mathbf{u})$
Eig : $\|A - VDV^T\| \approx \|AV - VD\| = \mathcal{O}(\mathbf{u})$

They almost always hold independent of $\kappa(A)$!

$\Longrightarrow$ We need some information about A^{-1}.

Suppose $\kappa(A) \gg \mathbf{u}^{-1}$ and $\widehat{X} := A^{-1}$.
Let $R_1 := fl(A^{-1})$. [best possible approx. in working precision]

$$\Longrightarrow \quad R_1 = A^{-1} + \Delta \quad \text{with} \quad |\Delta_{ij}| = \mathcal{O}(\mathbf{u})|A^{-1}_{ij}|.$$

However, $\|R_1 A - I\| < 1$ does not necessarily hold:

$$\begin{aligned} \|R_1 A - I\| &= \|(A^{-1} + \Delta)A - I\| = \|\Delta \cdot A\| \\ &\approx \|\Delta\|\|A\| = \mathcal{O}(\mathbf{u})\kappa(A) > 1. \end{aligned}$$

$\Longrightarrow$ Higher precision for representing $R \approx A^{-1}$ is necessary.

Principle of this work

To treat ill-conditioned problems, some higher precision arithmetic is necessary.

$\Longrightarrow$ It is not efficient in terms of computational cost that multiple precision arithmetic is applied to all computations.

$\Longrightarrow$ Restricted to a specific computation such as dot product or matrix multiplication, it is possible to use relatively fast algorithms of high precision arithmetic.

$\Longrightarrow$ We develop a fast verification method which uses pure floating-point arithmetic and approximations where possible.

Requirements for the algorithms

We only need

- Standard (backward stable) numerical algorithms for matrix factorizations (benefit from BLAS and LAPACK)
- Accurate matrix multiplication (arbitrary high-precision)

Accurate matrix multiplication

We assume that arbitrarily high precision dot product can be computed (with its error bound):

$\mathbb{F}$: set of floating-point numbers
$\mathbf{u}$: unit round-off ($\mathbf{u} \approx 10^{-16}$ in IEEE 754 double precision)

We want to use pure floating-point numbers/arithmetic where possible.

Let $A := \sum_{i=1}^{p} A_i$ and $B := \sum_{i=1}^{q} B_i$ with $A_i, B_i \in \mathbb{F}^{n\times n}$. As a general function of calculating a matrix product, we define

$$C = \{A \cdot B\}_k^{\ell}, \quad C := \sum_{i=1}^{\ell} C_i,\ C_i \in \mathbb{F}^{n\times n}$$

which computes $A \cdot B$ in k-fold working precision and stores it into ℓ-fold working precision (meaningful cases: $k \geq \ell$), i.e. it holds that

$$|C - A \cdot B| \leq c_1 \mathbf{u}^{\ell} |A \cdot B| + c_2 \mathbf{u}^{k} |A||B| =: E.$$

$\Longrightarrow \quad C - E \leq A \cdot B \leq C + E$

$\Longrightarrow \quad$ We use midpoint-radius form: $A \cdot B \in \langle C, E \rangle$

Brief structure of algorithms (Type I)

1: Put $X_0 = I$ and $k = 1$. (I: indentity matrix)
2: $B_k \leftarrow \{X_{k-1} \cdot A\}_k^1$. [higher prec. / working prec.]
3: If $\kappa(B_k) \approx 1$, then stop.
4: Matrix factorization $B_k \approx G_k H_k$ with $\kappa(B_k) \approx \kappa(G_k)$.
5: $T_k \approx G_k^{-1}$.
6: $X_k \leftarrow \{T_k \cdot X_{k-1}\}_k^k$. [higher prec. / higher prec.]
7: Update $k \leftarrow k + 1$ and return to 2.

[Computed in high prec. / Stored in high prec. or working prec.]

This includes Rump's method for inversion, inverse LU and QR factorizations.

Property of Type I

The following hold at the k-th iteration:

$$\kappa(X_k \cdot A) = \max\{\mathcal{O}(\mathbf{u}^k) \cdot \kappa(A),\ 1\}$$

$$\kappa(X_k) = \min\{\mathcal{O}(\mathbf{u}^{-k}),\ \kappa(A^{-1})\}$$

To give a rigorous proof is not possible.
(Some probabilistic analysis and expectation values are needed.)

For Rump's method for matrix inversion, Oishi et al. (JCAM, 2007) and Rump (JJIAM, to appear).

Algorithm for inverse Cholesky (Type II)

1: Put $X_0 = I$ and $k = 1$. $(\ell := \lceil k/2 \rceil)$
2: $\langle B_k, E_B \rangle \leftarrow \{A \cdot X_{k-1}\}_k^{\ell+1}$. [$k$-fold / $(\ell+1)$-fold]
3: $\langle C_k, E_C \rangle \leftarrow \{X_{k-1}^T \cdot B_k\}_{\ell+1}$. [$(\ell+1)$-fold / working prec.]
4: $\langle G_k, E_G \rangle \leftarrow \frac{1}{2}(\langle C_k, E_C \rangle + \langle C_k^T, E_C^T \rangle)$
5: Compute $\delta_k \geq cn\mathbf{u} \cdot \mathrm{tr}(G_k) + (\||X^T|E_B\| + \|E_G\|)$.
6: Cholesky factorization: $G_k + \widetilde{\delta}_k I \approx R_k^T R_k$.
7: If Step 6 fails, then stop. ($\Longrightarrow$ A is indefinite)
8: $T_k \approx R_k^{-1}$.
9: $X_k \leftarrow [X_{k-1} \cdot T_k]_{\ell+1}^{\ell}$. [$(\ell+1)$-fold / ℓ-fold]
10: Update $k \leftarrow k+1$ and return to 2.

Property of Type II

Even if A is ill-conditioned such as $\kappa(A) \gg \mathbf{u}^{-1}$, $A+\delta I$ is regularized such that $\kappa(A+\delta I) \approx \frac{1}{\delta} \approx (n\mathbf{u})^{-1}$.

In our algorithm, $\kappa(G_k + \widetilde{\delta}_k I) \approx (n\mathbf{u})^{-1}$ until the convergence. At the k-th iteration, we observe that

$$\kappa(X_k^T A X_k) \approx \max\{(n\mathbf{u})^k \kappa(A), 1\}.$$

Moreover, we observe $\kappa(X_k) \approx (n\mathbf{u})^{-\frac{k}{2}}$.

$\boxed{\text{Positive definiteness}}$ If $\|X^T AX - I\| < 1$ for any nonsingular X, then A is proved to be positive definite.

Algorithm for SVD (Type III)

1: Put $X_0 = V_0 = I$ and $k = 1$.
2: $T \leftarrow \{X_{k-1}^T \cdot A\}_k^1$. [higher prec., working prec.]
3: $B_k \leftarrow T \cdot V_{k-1}$.
4: $\widetilde{\sigma}_i = (B_k)_{ii}$, $g_i = \sum_{j \neq i} |B_k|_{ij}$ for all i.
5: If $\varepsilon_{\rm tol} \cdot \widetilde{\sigma}_i \geq g_i$ for all i, then stop.
6: SVD of B_k: $B_k \approx U_k \Sigma_k V_k^T$.
7: $X_k \leftarrow \{X_{k-1} \cdot U_k\}_k^k$. [higher prec., higher prec.]
8: Update $k \leftarrow k + 1$ and return to 2.

Property of Type III

At the k-th iteration

$$\frac{|\sigma_i - \widetilde{\sigma}_i|}{\sigma_n} \lesssim \mathbf{u} + \mathcal{O}(\mathbf{u}^k) \cdot \kappa(A)$$

(σ_n: the smallest singular value)

This can also be used for symmetic eigenvalue problems with small modifications. (Thanks to Prof. S. M. Rump.)

Features of the algorithms

- The proposed algorithms can increase the computational precision iteratively adapting to difficulty of the problem.

- Higher precision arithmetic is used only for matrix product, i.e. dot product. Other procedure is done by pure floating-point arithmetic.

- Therefore, the algorithm is expected to be fast in terms of measured computing time, if accurate dot product algorithms are available.

Numerical results for inverse LU / QR

We present an example of numerical experiments showing the behavior of inverse LU and inverse QR.

- double precision arithmetic as working precision ($\mathbf{u} = 2^{-53} \approx 1.1 \cdot 10^{-16}$)
- Test matrix: Rump's matrix (`randmat(n,cnd)` in INTLAB)
- $\mathtt{n} = 100$ and $\mathtt{cnd} = 10^{100}$ ($A \in \mathbb{F}^{100\times100}$ with $\kappa(A) \approx 1.75 \cdot 10^{107}$)

Table 1: Results for a Rump's matrix with $n = 100$ and $\kappa(A) \approx 1.75 \cdot 10^{107}$ by accurate inverse LU factorization

k	$\kappa(U_k)$	$\kappa(T_k)$	$\kappa(AX_k)$	$\mathbf{u}^k\kappa(A)$
0	–	–	$1.75 \cdot 10^{107}$	$1.75 \cdot 10^{107}$
1	$3.50 \cdot 10^{18}$	$3.50 \cdot 10^{18}$	$2.37 \cdot 10^{93}$	$1.94 \cdot 10^{91}$
2	$5.28 \cdot 10^{18}$	$5.28 \cdot 10^{18}$	$2.18 \cdot 10^{78}$	$2.16 \cdot 10^{75}$
3	$4.01 \cdot 10^{18}$	$4.01 \cdot 10^{18}$	$1.79 \cdot 10^{64}$	$2.39 \cdot 10^{59}$
4	$4.85 \cdot 10^{18}$	$4.85 \cdot 10^{18}$	$3.45 \cdot 10^{48}$	$2.66 \cdot 10^{43}$
5	$1.99 \cdot 10^{18}$	$1.99 \cdot 10^{18}$	$6.77 \cdot 10^{33}$	$2.95 \cdot 10^{27}$
6	$1.16 \cdot 10^{18}$	$1.16 \cdot 10^{18}$	$1.30 \cdot 10^{18}$	$3.27 \cdot 10^{11}$
7	$2.73 \cdot 10^{17}$	$2.73 \cdot 10^{17}$	$3.96 \cdot 10^{2}$	< 1
8	$1.91 \cdot 10^{2}$	$1.91 \cdot 10^{2}$	$8.68 \cdot 10^{1}$	< 1

Table 2: Results for a Rump's matrix with $n = 100$ and $\kappa(A) \approx 1.75 \cdot 10^{107}$ by accurate inverse QR factorization

k	$\kappa(R_k)$	$\kappa(T_k)$	$\kappa(AX_k)$	$\mathbf{u}^k\kappa(A)$
0	–	–	$1.75 \cdot 10^{107}$	$1.75 \cdot 10^{107}$
1	$3.27 \cdot 10^{19}$	$3.27 \cdot 10^{19}$	$2.00 \cdot 10^{93}$	$1.94 \cdot 10^{91}$
2	$1.86 \cdot 10^{19}$	$1.86 \cdot 10^{19}$	$1.06 \cdot 10^{77}$	$2.16 \cdot 10^{75}$
3	$7.97 \cdot 10^{17}$	$7.97 \cdot 10^{17}$	$1.61 \cdot 10^{62}$	$2.39 \cdot 10^{59}$
4	$2.20 \cdot 10^{17}$	$2.20 \cdot 10^{17}$	$4.23 \cdot 10^{46}$	$2.66 \cdot 10^{43}$
5	$2.31 \cdot 10^{17}$	$2.31 \cdot 10^{17}$	$2.00 \cdot 10^{32}$	$2.95 \cdot 10^{27}$
6	$4.04 \cdot 10^{17}$	$4.04 \cdot 10^{17}$	$6.69 \cdot 10^{16}$	$3.27 \cdot 10^{11}$
7	$2.18 \cdot 10^{18}$	$2.18 \cdot 10^{18}$	$1.39 \cdot 10^{3}$	< 1
8	$1.39 \cdot 10^{3}$	$1.39 \cdot 10^{3}$	$1.00 \cdot 10^{0}$	< 1

Application (1): Solutions of linear systems

We apply our algorithm to solutions of linear systems.

1. Compute an accurate LU factors of A^T (if Doolittle version is used)
$$PA^T X_U \approx L \quad \Leftrightarrow \quad X_U^T AP \approx L^T$$
2. Compute $\widetilde{y} = [X_U^T \cdot b]_m^1$ for $X_U = X_{1:m}$
3. Solve $L^T z = \widetilde{y}$ and obtain its approximate solution $\widetilde{z}$
4. Compute $\widetilde{x} = P\widetilde{z}$

Numerical result (1): (scaled) Hilbert matrix H_n

H_n is an integer matrix (exactly representable for $n \le 21$)

- $n = 15$ ($\kappa(H_{15}) \approx 6.12 \times 10^{20}$)
- Right-hand side: $b = H_{15}e \in \mathbb{F}^{15}$, $e := (1, \ldots, 1)^T = H_{15}^{-1}b$

(Matlab demo)

Numerical result (2): Rump's matrix

We evaluate normwise relative errors of approximate solutions of linear systems using our algorithm for several condition numbers.

- Rump's matrix `randmat`
- $n = 200$ and anticipated condition numbers from 10^{10} to 10^{100}
- $b = (1, \ldots, 1)^T$
- Comparison to GMP-based GEPP (multiple precision)

```
function [p,rel_err] = test_gmp_lin(A,b,xt,tol)
%  xt: given exact solution of Ax = b
% tol: tolerance for relative error

d = 53; norm_xt = norm(double(xt)); rel_err = 1;
while 1
  xv = gmp_lin(A,b,d);  % solve Ax=b using GMP
  % normwise relative error
  rel_err = norm(double(xt-xv))/norm_xt;
  if rel_err < tol, break, end
  d = 2*d;  % d = 53, 106, 212, ...
end
```

Table 3: Results for Rump's matrices with $n = 200$

	Proposed algorithm			GMP-based GEPP		
$\kappa(A)$	ε_1	t_1	m	ε_2	t_2	$d/53$
2.00^{21}	$5.1 \cdot 10^{-12}$	0.67	2	$5.7 \cdot 10^{-14}$	12.01	2
1.14^{34}	$4.8 \cdot 10^{-15}$	1.49	3	$5.2 \cdot 10^{-17}$	18.99	4
1.28^{44}	$1.8 \cdot 10^{-15}$	2.68	4	$5.0 \cdot 10^{-17}$	19.65	4
2.88^{54}	$1.9 \cdot 10^{-9}$	2.63	4	$2.8 \cdot 10^{-12}$	19.74	4
5.69^{61}	$1.2 \cdot 10^{-15}$	4.11	5	$3.9 \cdot 10^{-17}$	30.89	8
2.05^{74}	$2.7 \cdot 10^{-15}$	6.02	6	$4.0 \cdot 10^{-17}$	30.13	8
1.06^{84}	$4.7 \cdot 10^{-9}$	6.05	6	$3.8 \cdot 10^{-17}$	30.22	8
6.59^{93}	$4.8 \cdot 10^{-13}$	8.12	7	$4.5 \cdot 10^{-17}$	29.99	8
2.11^{102}	$1.4 \cdot 10^{-15}$	11.88	8	$4.8 \cdot 10^{-17}$	31.30	8

Application (2): Verified computation of determinant

(Matlab demo)

Thanks!