Matrix computations: Going beyond libraries

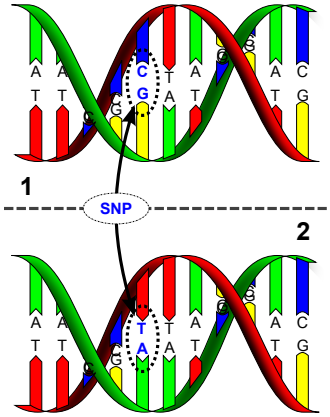
Paolo Bientinesi
Umeå Universitet
pauldj@cs.umu.se

October 5, 2022
Swedish e-Science Academy

10-second lecture

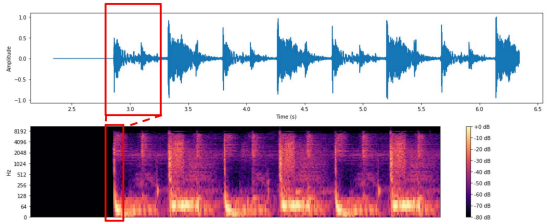
- Matrix computations are ubiquitous in science and engineering
- Many excellent linear algebra libraries exist
- Are they used (properly)? Not so much

Matrix computations everywhere



Genome-wide Association Studies

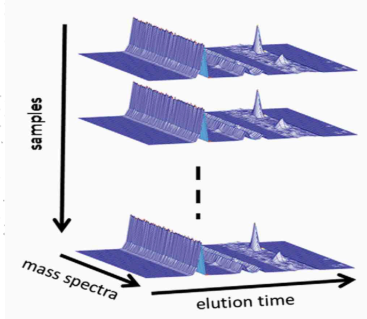
Estimation of Earth's gravitational field



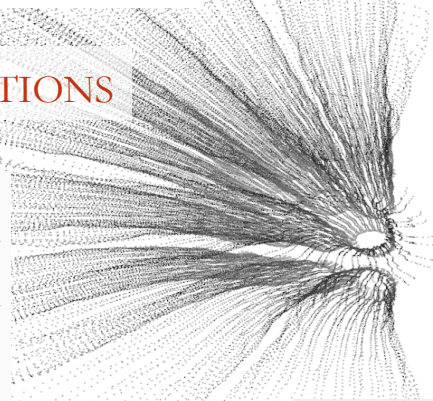
Music Information Retrieval

Digital signal processing

Matrix Computations



Chemometrics



Simulation of carbon nanotubes

Libraries since the '70s!

- 1971: NAG Fortran Library – scientific computing
- 1972: EISPACK – eigenproblems
- 1973: BLAS 1 – basic vector ops
- 1975: LINPACK – linear systems
- ...
- ...

Hierarchy of Libraries: 1980s

BLAS-2: Mat-vec ops, ACM TOMS 1988

BLAS-3: mat-mat ops, ACM TOMS 1990

BLAS-1, BLAS-2, BLAS-3

Hierarchy of Libraries: 1990s

Solvers & eigensolvers, 1992

LAPACK

BLAS-1, BLAS-2, BLAS-3

Hierarchy of Libraries: 1990s

Distributed memory, 1995, 1997

ScaLAPACK, PLAPACK, ...

LAPACK

BLAS-1, BLAS-2, BLAS-3

Hierarchy of Libraries: 1990s

Dense & sparse, 1997

PETSc, ...

ScaLAPACK, PLAPACK, ...

LAPACK

BLAS-1, BLAS-2, BLAS-3

Many libraries! ... (and many more iterative ones)

PETSc, Trilinos, ...

ScaLAPACK, PLAPACK, Elemental, ...

LAPACK, Plasma, SuperMatrix, Magma, ...

BLAS-1, BLAS-2, BLAS-3, ATLAS, BTO-BLAS, BLIS, ...

ACM Turing Award

JACK DONGARRA

United States – 2021

CITATION

For his pioneering contributions to numerical algorithms and libraries that enabled high performance computational software to keep pace with exponential hardware improvements for over four decades

RESEARCH SUBJECTS

But. . . Issue #1: expression vs. code

But... Issue #1: expression vs. code

$$C := C + A * B^T + B * A^T$$

Naive translation to code:

```
CALL DGEMM( 'N', 'Y', N-K-KB+1, N-K-KB+1, KB, ONE, A( K+KB, K+KB ), LDA,
      $     B( K, K+KB ), LDB, ONE, C( K+KB, K+KB ), LDC )
CALL DGEMM( 'N', 'Y', N-K-KB+1, N-K-KB+1, KB, ONE, B( K+KB, K+KB ), LDA,
      $     A( K, K+KB ), LDB, ONE, C( K+KB, K+KB ), LDC )
```

The "right" way:

```
CALL DSYR2K( UPLO, 'Transpose', N-K-KB+1, KB, -ONE, A( K, K+KB ), LDA,
      $      B( K, K+KB ), LDB, ONE, A( K+KB, K+KB ), LDA )
```

But... Issue #1: expression vs. code

$$C := C + A * B^T + B * A^T$$

Naive translation to code:

```
CALL DGEMM( 'N', 'Y', N-K-KB+1, N-K-KB+1, KB, ONE, A( K+KB, K+KB ), LDA,
      $     B( K, K+KB ), LDB, ONE, C( K+KB, K+KB ), LDC )
CALL DGEMM( 'N', 'Y', N-K-KB+1, N-K-KB+1, KB, ONE, B( K+KB, K+KB ), LDA,
      $     A( K, K+KB ), LDB, ONE, C( K+KB, K+KB ), LDC )
```

The "right" way:

```
CALL DSYR2K( UPLO, 'Transpose', N-K-KB+1, KB, -ONE, A( K, K+KB ), LDA,
      $      B( K, K+KB ), LDB, ONE, A( K+KB, K+KB ), LDA )
```

Not exactly user friendly. Certainly not users' preference (anymore)

Users preference(s)

$$C := C + A * B^T + B * A^T$$

```
C = A * B'+ B * A' + C;                          // Matlab, Octave

C = A * transpose(B) + B * transpose(A) + C      // Julia

C = A * trans(B) + B * trans(A) + C;             // Armadillo

C = A * B.transpose() + B * A.transpose() + C;   // Eigen

ct = at @ bt.T + bt @ at.T + ct                  // NumPy

ct <- at %*% t(bt) + bt %*% t(at) + ct           // R

C = A@tf.transpose(B) + B@tf.transpose(A) + C    // TensorFlow

C = A@torch.t(B) + B@torch.t(A) + C              // PyTorch
```

Issue #2: building blocks vs. applications

- BLAS: **Basic** Linear Algebra Subprograms

  $\boldsymbol{w} := \boldsymbol{x}^T\boldsymbol{y}$ (DOT), $\boldsymbol{y} := \alpha\boldsymbol{x} + \boldsymbol{y}$ (AXPY), $\eta := (\boldsymbol{x}^T\boldsymbol{x})^{1/2}$ (NORM)

  $\boldsymbol{y} := \boldsymbol{A}\boldsymbol{x}$ (GEMV), $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ (TRSV)

  $\boldsymbol{C} := \boldsymbol{A}\boldsymbol{B}$ (GEMM), $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$ (TRSM)

- LAPACK: Linear Algebra Package

$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$	eigenproblems
$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$	linear systems, least squares problems
$\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{A}$	matrix factorizations, ...

Applications

Signal Processing	$x := \left(A^{-T}B^{T}BA^{-1} + R^{T}LR\right)^{-1} A^{-T}B^{T}BA^{-1}y$	$R \in \mathbb{R}^{n-1\times n}$, UT; $L \in \mathbb{R}^{n-1\times n-1}$, DI
Kalman Filter	$K_k := P_k^b H^T (HP_k^b H^T + R)^{-1};\ x_k^a := x_k^b + K_k(z_k - Hx_k^b);\ P_k^a := (I - K_K H)\, P_k^b$	
Ensemble Kalman Filter	$X^a := X^b + \left(B^{-1} + H^T R^{-1} H\right)^{-1} \left(Y - HX^b\right)$	$B \in \mathbb{R}^{N\times N}$ SSPD; $R \in \mathbb{R}^{m\times m}$, SSPD
Ensemble Kalman Filter	$\delta X := \left(B^{-1} + H^T R^{-1} H\right)^{-1} H^T R^{-1} \left(Y - HX^b\right)$	
Ensemble Kalman Filter	$\delta X := XV^T \left(R + HX(HX)^T\right)^{-1} \left(Y - HX^b\right)$	
Image Restoration	$x_k := (H^T H + \lambda\sigma^2 I_n)^{-1}(H^T y + \lambda\sigma^2(v_{k-1} - u_{k-1}))$	
Image Restoration	$H^\dagger := H^T(HH^T)^{-1};\ y_k := H^\dagger y + (I_n - H^\dagger H)x_k$	
Rand. Matrix Inversion	$X_{k+1} := S(S^TAS)^{-1}S^T + (I_n - S(S^TAS)^{-1}S^TA)X_k(I_n - AS(S^TAS)^{-1}S^T)$	
Rand. Matrix Inversion	$X_{k+1} := X_k + WA^TS(S^TAWA^TS)^{-1}S^T(I_n - AX_k)$	$W \in \mathbb{R}^{n\times n}$, SPD
Rand. Matrix Inversion	$X_{k+1} := X_k + (I_n - X_kA^T)S(S^TA^TWAS)^{-1}S^TA^TW$	
Rand. Matrix Inversion	$\Lambda := S(S^TAWAS)^{-1}S^T;\ \Theta := \Lambda AW;\ M_k := X_kA - I$ $X_{k+1} := X_k - M_k\Theta - (M_k\Theta)^T + \Theta^T(AX_kA - A)\Theta$	

Generalized Least Squares	$b := (X^T M^{-1} X)^{-1} X^T M^{-1} y$	$n > m$; $M \in \mathbb{R}^{n \times n}$, SPD; $X \in \mathbb{R}^{n \times m}$; $y \in \mathbb{R}^{n \times 1}$
Stochastic Newton	$B_k := \frac{k}{k-1} B_{k-1}(I_n - A^T W_k((k-1)I_l + W_k^T A B_{k-1} A^T W_k)^{-1} W_k^T A B_{k-1})$	
Optimization	$x_f := WA^T(AWA^T)^{-1}(b - Ax)$; $x_o := W(A^T(AWA^T)^{-1}Ax - c)$	
Optimization	$x := W(A^T(AWA^T)^{-1}b - c)$	
Triangular Matrix Inv.	$X_{10} := L_{10}L_{00}^{-1}$; $X_{20} := L_{20} + L_{22}^{-1}L_{21}L_{11}^{-1}L_{10}$; $X_{11} := L_{11}^{-1}$; $X_{21} := -L_{22}^{-1}L_{21}$	
Tikhonov Regularization	$x := (A^T A + \Gamma^T \Gamma)^{-1} A^T b$	$A \in \mathbb{R}^{n \times m}$; $\Gamma \in \mathbb{R}^{m \times m}$; $b \in \mathbb{R}^{n \times 1}$
Tikhonov Regularization	$x := (A^T A + \alpha^2 I)^{-1} A^T b$	
Gen. Tikhonov Reg.	$x := (A^T PA + Q)^{-1}(A^T Pb + Qx_0)$	$P \in \mathbb{R}^{n \times n}$, SSPD; $Q \in \mathbb{R}^{m \times m}$, SSPD; $x_0 \in \mathbb{R}^{m \times 1}$
Gen. Tikhonov reg.	$x := x_0 + (A^T PA + Q)^{-1}(A^T P(b - Ax_0))$	
LMMSE estimator	$K_{t+1} := C_t A^T (AC_t A^T + C_z)^{-1}$; $x_{t+1} := x_t + K_{t+1}(y - Ax_t)$; $C_{t+1} := (I - K_{t+1}A)C_t$	
LMMSE estimator	$x_{\text{out}} = C_X A^T (AC_X A^T + C_Z)^{-1}(y - Ax) + x$	
LMMSE estimator	$x_{\text{out}} := (A^T C_Z^{-1} A + C_X^{-1})^{-1} A^T C_Z^{-1}(y - Ax) + x$	

$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k(z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$\Lambda := S(S^T A W A S)^{-1} S^T; \ \Theta := \Lambda A W; \ M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A)\Theta$$

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y \quad \ldots \quad E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$$

$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k(z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$\Lambda := S(S^T A W A S)^{-1} S^T; \; \Theta := \Lambda A W; \; M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A)\Theta$$

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y \quad \ldots \quad E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$$

$$\boxed{y := \alpha x + y} \quad \boxed{LU = A} \quad \cdots \quad \boxed{C := \alpha AB + \beta C}$$

$$\boxed{X := A^{-1} B} \quad \boxed{C := AB^T + BA^T + C} \quad \boxed{X := L^{-1} M L^{-T}} \quad \boxed{QR = A}$$

$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k(z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$\Lambda := S(S^T A W A S)^{-1} S^T; \; \Theta := \Lambda A W; \; M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A)\Theta$$

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y \quad \ldots \quad E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$$

?

$$y := \alpha x + y \qquad LU = A \qquad \cdots \qquad C := \alpha AB + \beta C$$

$$X := A^{-1} B \qquad C := AB^T + BA^T + C \qquad X := L^{-1} M L^{-T} \qquad QR = A$$

$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k(z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$\Lambda := S(S^T AWAS)^{-1} S^T; \; \Theta := \Lambda A W; \; M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A)\Theta$$

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y \quad \ldots \quad E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$$

LINEAR ALGEBRA MAPPING PROBLEM

C. Psarras, H. Barthels, P. Bientinesi, [arXiv:1911.09421]
"The Linear Algebra Mapping Problem. Current state of linear algebra languages and libraries", ACM TOMS, 2022

A. Sankaran, N.A. Alashti, C. Psarras, P. Bientinesi, [arXiv:2202.09888]
"Benchmarking the Linear Algebra Awareness of TensorFlow and PyTorch", iWAPT-22

H. Barthels, C. Psarras, P. Bientinesi, [arXiv:1912.12924]
"Linnea: Automatic Generation of Efficient Linear Algebra Programs", ACM TOMS, 2021

Linear Algebra Mapping Problem (“LAMP”)

Linear Algebra Mapping Problem ("LAMP")

- $\mathcal{E}$: a sequence of explicit assignments — $\boldsymbol{var_i := EXP_i}$
- $\mathcal{K}$: a set of available computational building blocks — e.g., BLAS, LAPACK, ...
- $\mathcal{M}$: a cost function defined over $\mathcal{K}^+$ — #FLOPs, exec. time, #mem.ops, stability

Linear Algebra Mapping Problem ("LAMP")

- $\mathcal{E}$: a sequence of explicit assignments — $\boldsymbol{var_i := EXP_i}$
- $\mathcal{K}$: a set of available computational building blocks — e.g., BLAS, LAPACK, ...
- $\mathcal{M}$: a cost function defined over $\mathcal{K}^+$ — #FLOPs, exec. time, #mem.ops, stability

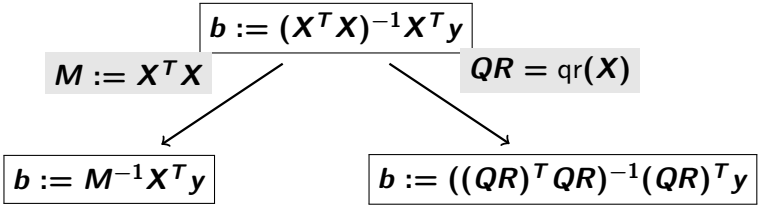
LAMP:
Find a sequence of calls to building blocks in $\mathcal{K}$, optimal according to $\mathcal{M}$, that computes all the assignments in $\mathcal{E}$.

- Suboptimal solution $\longrightarrow$ easy
- Optimality $\longrightarrow$ NP complete $\longleftarrow$ reduction from Ensemble Computation

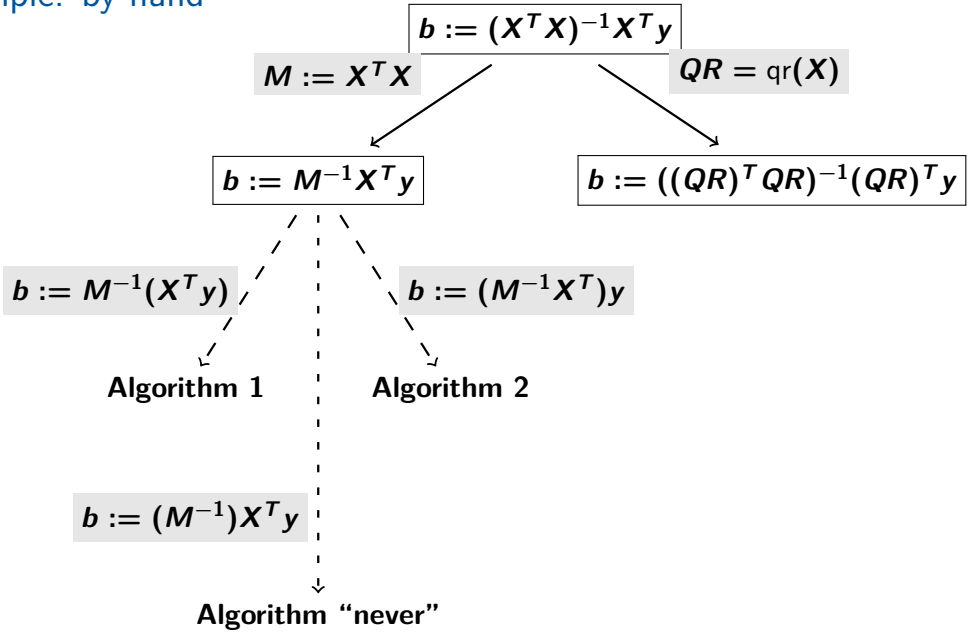
Example:

$$\boxed{\boldsymbol{b} := (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$$

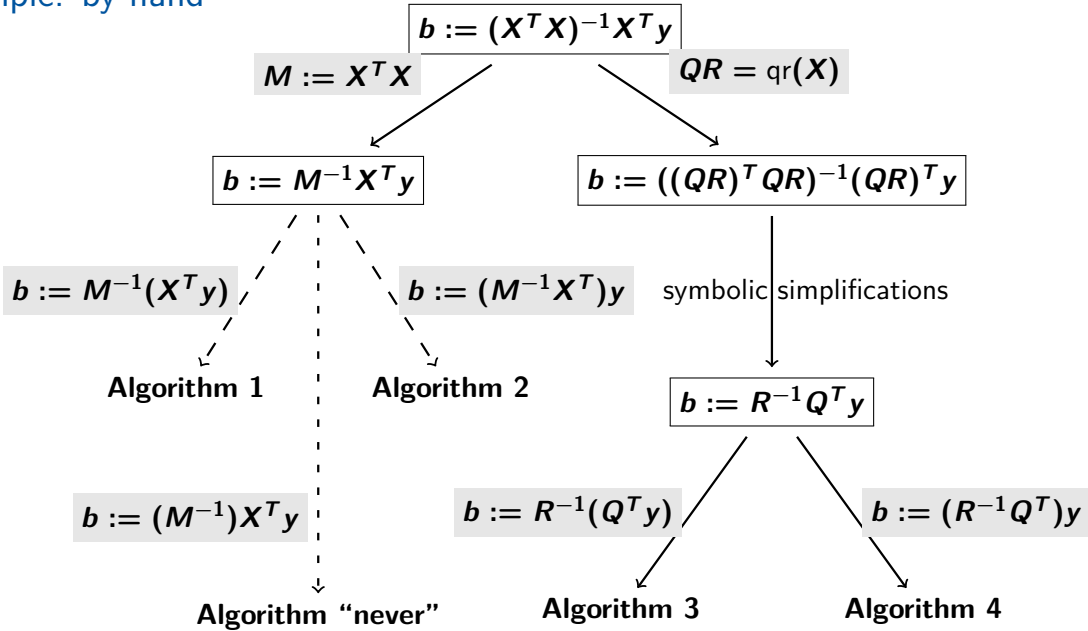
Example: by hand



Example: by hand



Example: by hand



Example: high-level language

$$\boxed{\boldsymbol{b} := (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$$

```
b = inv(X'*X)*X'*y          // Matlab (naive)

b = (X'*X)\X'*y             // Matlab (recommended)
```

Example: $\boldsymbol{b := (X^T X)^{-1} X^T y}$

1) **By hand**: C/Fortran Required: expertize, time, patience

computer efficiency!  . . . but human productivity 

2) **High-level languages**: Matlab, Julia, R, Armadillo (C++), NumPy, TensorFlow, . . .

human productivity!  . . . computer efficiency?

Investigation: How well do high-level languages solve LAMPs?

High-level notation

- Matlab
- Octave
- Julia
- C++ with Armadillo
- C++ with Eigen
- NumPy
- R
- (TensorFlow)
- (PyTorch)

Q1: Do they map? matrix products

`input`	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B								

Q1: Do they map? matrix products

`input`	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27

Q1: Do they map? matrix products

input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	

Q1: Do they map? matrix products

input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	

C = C+A*A'

Q1: Do they map? matrix products

input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	
C = C+A*A'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14

Q1: Do they map? matrix products

input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	
C = C+A*A'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14
SYRK	✓	✓	✓	×	×	✓	✓	

Q1: Do they map? matrix products

input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	
C = C+A*A'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14
SYRK	✓	✓	✓	×	×	✓	✓	
C = C+AB'+BA'								

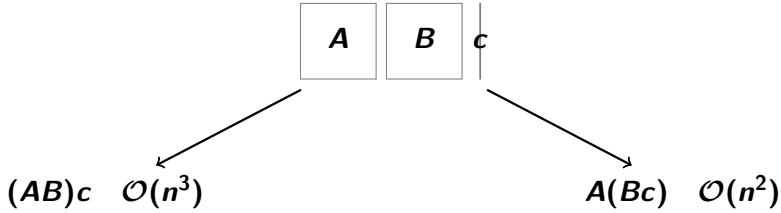
Q1: Do they map? matrix products

input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	
C = C+A*A'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14
SYRK	✓	✓	✓	×	×	✓	✓	
C = C+AB'+BA'	0.57	0.59	0.69	0.59	0.58	0.57	0.58	0.28

Q1: Do they map? matrix products

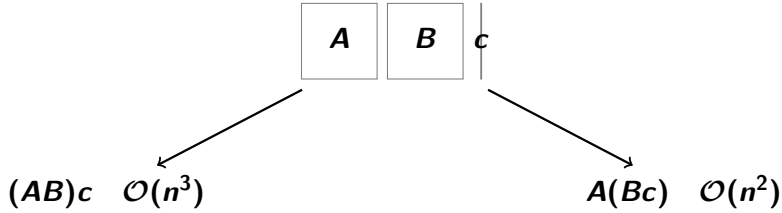
input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	
C = C+A*A'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14
SYRK	✓	✓	✓	×	×	✓	✓	
C = C+AB'+BA'	0.57	0.59	0.69	0.59	0.58	0.57	0.58	0.28
SYR2K	×	×	×	×	×	×	×	

Optimal parenthesisation



Matrix product is associative, but its cost is not

Optimal parenthesisation



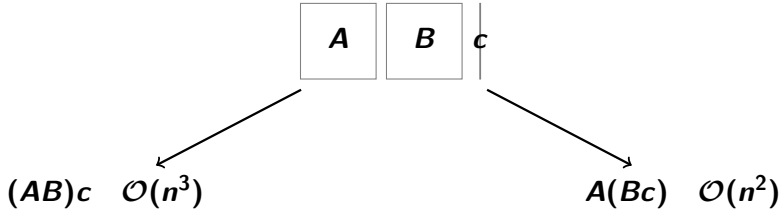
Matrix product is associative, but its cost is not

Matrix Chain(k)

Which is the best one?

Catalan$(k-1)$ parenthesiszations
if parallelism, then $k!$ parenthesiszations

Optimal parenthesisation



Matrix product is associative, but its cost is not

Matrix Chain(k)

Catalan$(k-1)$ parenthesiszations
if parallelism, then $k!$ parenthesiszations

Which is the best one?

Matrix Chain Algorithm(k)

$O(k \log k)$ Hu & Shing 1982
$O(k^3)$ dynamic programming

Q2: Matrix Chain?

Chain	**Optimal Evaluation**
1) “left-to-right”	((A B) C)
2) “right-to-left”	(A (B C))
3) “mixed”	((A B) (C D))

Q2: Matrix Chain?

Chain	**Optimal Evaluation**
1) "left-to-right"	((A B) C)
2) "right-to-left"	(A (B C))
3) "mixed"	((A B) (C D))

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055

Q2: Matrix Chain?

Chain	Optimal Evaluation
1) "left-to-right"	((A B) C)
2) "right-to-left"	(A (B C))
3) "mixed"	((A B) (C D))

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	A*B*C	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056

Q2: Matrix Chain?

Chain	Optimal Evaluation
1) "left-to-right"	((A B) C)
2) "right-to-left"	(A (B C))
3) "mixed"	((A B) (C D))

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	A*B*C	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	A*B*C*D	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22

Q2: Matrix Chain?

Chain	Optimal Evaluation
1) "left-to-right"	((A B) C)
2) "right-to-left"	(A (B C))
3) "mixed"	((A B) (C D))

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	A*B*C	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	A*B*C*D	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	Matrix chains	×	×	×	×	×	≈	×

Wait! They read our paper!

Chain	Optimal Evaluation
1) "left-to-right"	((A B) C)
2) "right-to-left"	(A (B C))
3) "mixed"	((A B) (C D))

		Matlab	Octave	**Julia**	R	Eigen	Armad.	NumPy
1)	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	A*B*C	0.42	0.43	**0.054**	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	A*B*C*D	0.32	0.33	**0.22**	0.33	0.35	0.31	0.33
	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	Matrix chains	×	×	✓	×	×	≈	×

Challenge: Generalized Matrix Chain

[arXiv:1804.04021]

Matrix chains in practice

- Unary operators: transposition, inversion ($\boldsymbol{X} := \boldsymbol{A}\boldsymbol{B}^T\boldsymbol{C}^{-T}\boldsymbol{D} + \ldots$)
- Overlapping kernels (e.g., $\boldsymbol{L} \leftarrow \boldsymbol{L}^{-1}$, $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$)
- Decompositions (e.g., $\boldsymbol{A} \rightarrow \boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}$, $\boldsymbol{A} \rightarrow \boldsymbol{L}\boldsymbol{U}$)
- Properties & specialized kernels (`GEMM`, `TRMM`, `SYMM`, ...)

Challenge: Not all flops were created equal

[arXiv:1706.01341]

#FLOPs vs. execution time (vs. numerical stability)

$$\operatorname*{\mathbf{argmin}}_{\mathcal{A}} \big(\text{FLOPs}(\mathcal{A}) \big) \neq \operatorname*{\mathbf{argmin}}_{\mathcal{A}} \big(\text{time}(\mathcal{A}) \big)$$

$\Rightarrow$ **Performance prediction**: efficiency

Q3: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61

Q3: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.73	0.62	0.69	N/A	0.62	0.65	0.46

Q3: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.73	0.62	0.69	N/A	0.62	0.65	0.46
	SPD	**0.41**	**0.40**	0.60	0.63	N/A	**0.34**	0.62	0.31

Q3: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.73	0.62	0.69	N/A	0.62	0.65	0.46
	SPD	**0.41**	**0.40**	0.60	0.63	N/A	**0.34**	0.62	0.31
	Triangular	**0.03**	**0.04**	**0.03**	0.63	N/A	0.62	0.65	0.03

Q3: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.73	0.62	0.69	N/A	0.62	0.65	0.46
	SPD	**0.41**	**0.40**	0.60	0.63	N/A	**0.34**	0.62	0.31
	Triangular	**0.03**	**0.04**	**0.03**	0.63	N/A	0.62	0.65	0.03
	Diagonal	0.03	0.05	**0.01**	0.63	N/A	**0.03**	0.62	0.001
		$\approx$	$\approx$	$\approx$	$\times$	$\times$	$\approx$	$\times$	

Q3: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.73	0.62	0.69	N/A	0.62	0.65	0.46
	SPD	**0.41**	**0.40**	0.60	0.63	N/A	**0.34**	0.62	0.31
	Triangular	**0.03**	**0.04**	**0.03**	0.63	N/A	0.62	0.65	0.03
	Diagonal	0.03	0.05	**0.01**	0.63	N/A	**0.03**	0.62	0.001
		≈	≈	≈	×	×	≈	×	
C = A*B	-	1.44	1.48	1.47	1.47	1.45	1.44	1.44	1.46
	Triangular	1.44	1.48	**0.75**	1.47	1.45	1.44	1.44	0.74
	Diagonal	1.44	1.48	**0.03**	1.47	1.45	1.42	1.44	0.06
		×	×	✓	×	×	×	×	

Q4: Common Subexpressions?

$$\begin{cases} \boldsymbol{X} := \boldsymbol{AB} \\ \boldsymbol{Y} := \boldsymbol{AB} \end{cases} \quad \rightarrow \quad \begin{cases} \boldsymbol{X} := \boldsymbol{AB} \\ \boldsymbol{Y} := \boldsymbol{X} \end{cases}$$

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
copy	0.27	0.31	0.36	0.30	0.30	0.26	0.30
direct	0.54	0.6	0.61	0.56	0.58	0.52	0.55
	×	×	×	×	×	×	×

Challenge: Common Subexpressions

[arXiv:1710.06915]

$$\begin{cases} X := AB^{-T}C \\ Y := B^{-1}A^{T}D \end{cases} \quad \rightarrow \quad \begin{cases} Z := AB^{-T} \\ X := ZC \\ Y := Z^{T}D \end{cases}$$

BUT

$$X := ABABv \quad \not\rightarrow \quad \begin{cases} Z := AB \\ X := ZZv \end{cases}$$

Q6: Other features?

- Code motion

```
for i = 1:n,
    C = A*B;
    d[i] = C[i,i];
end
```

$\rightarrow$?

```
C = A*B;
for i = 1:n,
    d[i] = C[i,i];
end
```

$\times$

- Blocked operands

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M_1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M_2} \end{bmatrix} \quad \rightarrow ? \quad \begin{cases} \boldsymbol{M_1 x_T = y_T} \\ \boldsymbol{M_2 x_B = y_B} \end{cases}, \quad \begin{cases} \boldsymbol{y_T := M_1 x_T} \\ \boldsymbol{y_B := M_2 x_B} \end{cases} \quad \times, \times$$

- $\mathrm{diag}(\boldsymbol{A} + \boldsymbol{B})$ vs. $\mathrm{diag}(\boldsymbol{A}) + \mathrm{diag}(\boldsymbol{B})$ $\rightarrow$ Armadillo

  $\mathrm{diag}(\boldsymbol{AB})$ vs. ... $\rightarrow$ $\times$

Summary

- Note: This evaluation was **NOT** a ranking of languages [arXiv:1911.09421]
- LAMPs are challenging — the optimal solution requires deep & interdisciplinary expertise
- To users:
  Beware! Compilers & languages are great with scalars, not so much with matrices (yet?)
- To language developers:
  Please pay attention to the optimizations we exposed. They arise frequently in the solution of LAMPs.
- Our approach: (a different talk) "**Linnea**: A linear algebra compiler"
  [arXiv:1912.12924] https://linnea.cs.umu.se/