Programming languages for matrix computations

Paolo Bientinesi, Henrik Barthels, Christos Psarras

Umeå Universitet

May 10, 2019
University of Manchester

High Performance and Automatic Computing


IBM
360

1950s & 1960s

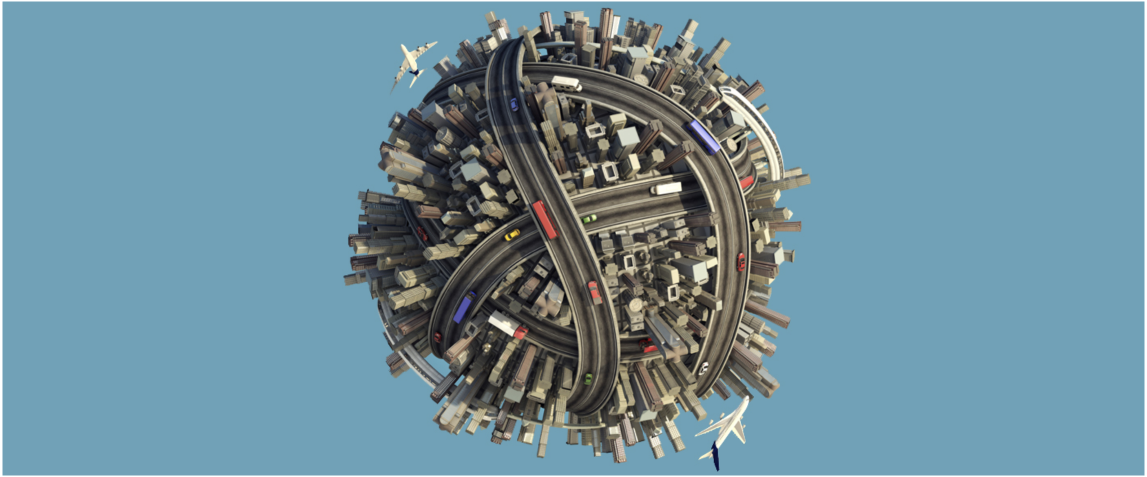
- Computers difficult to program

  BUT

- "simple" architectures: no memory hierarchy

- Cost($\mathcal{A}lg$) $\equiv$ #operations($\mathcal{A}lg$)

- Programs "easy" to optimize — smart code



Cheong Suk-Wai, *A borderless world*

Since the 1970s

- Libraries

- Identification, analysis, optimization of **building blocks**

- EISPACK, LINPACK, BLAS, LAPACK, . . .
  FFTs, numerical integration, . . .

- Convenience, portability, **separation** of concerns

Since the 1980s

- Memory hierarchies
- Increasingly complex architectures — caching, prefetching, locality, ...
- Cost($\mathcal{A}lg$) $\neq$ #operations($\mathcal{A}lg$)
- Programs difficult to optimize — complex, non-portable code
- Libraries: necessity (wrt performance)

Present: the world of scientific computing

$$\boldsymbol{y} = X\beta + Z\boldsymbol{u} + \epsilon$$

$$\min_x \|A\mathbf{x} - \mathbf{b}\|^2 + \|\Gamma\mathbf{x}\|^2$$

Linear mixed models

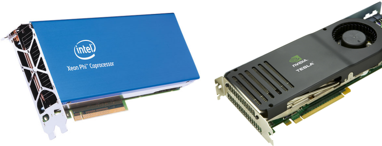
$$V_{LJ} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Lennard-Jones potential

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \left[\frac{-2\hbar}{2\mu}\nabla^2 + V(\mathbf{r},t)\right]\Psi(\mathbf{r},t)$$

Schrödinger eqn.

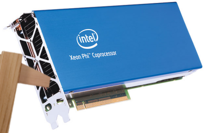
⋮



This talk

Matrix computations

Matlab, Julia, R, Eigen, Armadillo, ...



Matrix computations

Signal Processing	$x := \left(A^{-T}B^TBA^{-1} + R^TLR\right)^{-1} A^{-T}B^TBA^{-1}y$	$R \in \mathbb{R}^{n-1 \times n}$, 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	$X^a := X^b + \left(B^{-1} + H^TR^{-1}H\right)^{-1} \left(Y - HX^b\right)$	$B \in \mathbb{R}^{N \times N}$ SSPD; $R \in \mathbb{R}^{m \times m}$, SSPD
Ensemble Kalman Filter	$\delta X := \left(B^{-1} + H^TR^{-1}H\right)^{-1} H^TR^{-1} \left(Y - HX^b\right)$	
Ensemble Kalman Filter	$\delta X := XV^T \left(R + HX(HX)^T\right)^{-1} \left(Y - HX^b\right)$	
Image Restoration	$x_k := (H^TH + \lambda\sigma^2 I_n)^{-1}(H^Ty + \lambda\sigma^2(v_{k-1} - u_{k-1}))$	
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	$\Lambda := S(S^TAWAS)^{-1}S^T;\ \Theta := \Lambda AW;\ M_k := X_kA - I$ $X_{k+1} := X_k - M_k\Theta - (M_k\Theta)^T + \Theta^T(AX_kA - A)\Theta$	

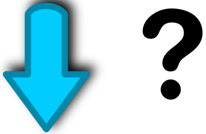
Matrix computations (2)

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.	$x := (A^T P A + Q)^{-1}(A^T P b + Q x_0)$	$P \in \mathbb{R}^{n \times n}$, SSPD; $Q \in \mathbb{R}^{m \times m}$, SSPD; $x_0 \in \mathbb{R}^{m \times 1}$
Gen. Tikhonov reg.	$x := x_0 + (A^T P A + Q)^{-1}(A^T P(b - A x_0))$	
LMMSE estimator	$K_{t+1} := C_t A^T (A C_t A^T + C_z)^{-1}$; $x_{t+1} := x_t + K_{t+1}(y - A x_t)$; $C_{t+1} := (I - K_{t+1} A) C_t$	
LMMSE estimator	$x_{\text{out}} = C_X A^T (A C_X A^T + C_Z)^{-1}(y - Ax) + x$	
LMMSE estimator	$x_{\text{out}} := (A^T C_Z^{-1} A + C_X^{-1})^{-1} A^T C_Z^{-1}(y - Ax) + x$	

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y$$

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

$$E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T \quad \ldots$$



2-step solution

MUL ADD MOV

MOVAPD

VFMADDPD ...

$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y$

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

$E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$ ...

LINEAR ALGEBRA MAPPING PROBLEM (LAMP)

$y := \alpha x + y$ $:= \alpha AB + \beta C$

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

BLAS **LAPACK** **MKL** ...

MUL ADD MOV

MOVAPD

VFMADDPD ...

Linear Algebra Mapping Problem

- $\mathcal{E}$: a sequence of explicit assignments $var_i := EXP_i$
- $\mathcal{K}$: a set of available computational building blocks BLAS, LAPACK, ...
- $\mathcal{M}$: a cost function defined over $\mathcal{K}^+$ FLOPs, data movement, stability, time

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 easy
- Optimality NP complete reduction from Ensemble Computation

LAMPs are everywhere

- Kernels

  $\mathcal{E}$: $\{C := A * B + C\}$ $\mathcal{K}$: processor's ISA $\mathcal{M}$: execution time

- Automatic code generation

  ATLAS [Whaley 2001], FLAME [Gunnels 2001], Build-To-Order BLAS [Siek 2008], LGEN [Spampinato 2014], . . .

- "Matrix Chain Problem"

  $\mathcal{E}$: $\{X := M_1 M_2 \cdots M_k\}$ $\mathcal{K}$: $\{C := A * B\}$ $\mathcal{M}$: #flops

- Applications

  $\mathcal{E}$: explicit assignments $\mathcal{K}$: libraries $\mathcal{M}$: execution time + ...

Problem acknowledged, yet overlooked

Libraries exist (a myriad of them) — How do I express my problem to use them?

- By hand. C/Fortran. Patience. Expertize.

computer efficiency! . . . but human productivity?

- High-level language. Matlab, Julia, R, Eigen, Armadillo, NumPy, . . . Quick prototyping.

human productivity! . . . but computer efficiency?

Slow solutions → "2-language problem"

2-language problem

- [arXiv:1904.12380]: Softmax function.
  Original implementation: Eigen.
  Reimplemented with explicit calls to BLAS and MKL.

- [F1000Res.2016]: The GenABEL Project for statistical genomics.
  Original implementation: R.
  Kernels reimplemented in C.

- Convex, non-smooth optimization in image processing.
  Original implementation: Python.
  Looking for alternatives.

State of the art

Many languages & environments that allow a high level of abstraction

Matlab, Julia, R, Eigen, Armadillo, NumPy

Why? Popularity, expressiveness, (performance...)

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

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
```

Do they map? matrix products

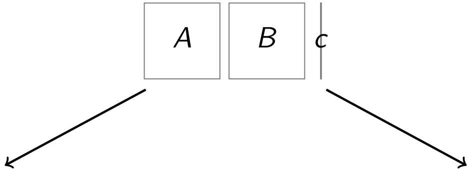
	Matlab	Julia	R	Eigen	Armad.	NumPy	C
C = AB+C	0.28	0.31	0.30	0.29	0.28	0.29	0.26
C = AB	. . .						
C = αAB							
C = αAB+βC							
`GEMM`	✓	✓	✓	✓	✓	✓	
C = C+AA'	0.17	0.22	0.31	0.29	0.17	0.18	0.13
`SYRK`	✓	✓	×	×	✓	✓	
C = C+AB'+BA'	0.56	0.70	0.61	0.57	0.56	0.58	0.27
`SYR2K`	×	×	×	×	×	×	

Do they map? $Ax = b \quad \equiv \quad x := A\backslash b \quad \not\equiv \quad \text{inv}(A) * b$

	Matlab	Julia	R	Eigen	Armad.	NumPy	C
$x := A\backslash b$	0.70	0.62	0.67	0.63	0.62	0.65	0.61
$\text{inv}(A) * b$	1.74	1.45	2.20	2.20	**0.62**	2.22	1.71
LinSolve	-	-	-	-	✓	-	-

… should they map?

Parenthesisation



$(AB)c \quad \mathcal{O}(n^3)$ $A(Bc) \quad \mathcal{O}(n^2)$

Product is associative, but cost is not

Matrix Chain Algorithm $O(k \log k)$ Hu & Shing 1982; $O(k^3)$ dynamic programming

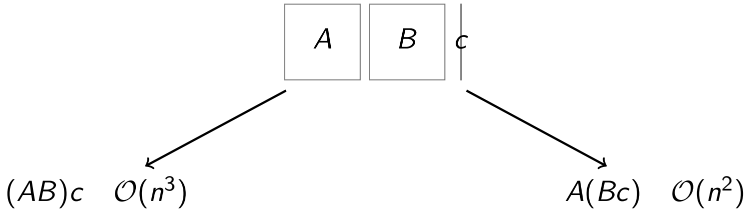
Matrix Chain?

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

	Matlab	Julia	R	Eigen	Armad.	NumPy
LtR no par.	0.056	0.055	0.061	0.058	0.056	0.055
LtR guided	0.056	0.055	0.061	0.058	0.056	0.055
RtL no par.	0.42	0.42	0.44	0.42	**0.055**	0.42
RtL guided	0.055	0.054	0.059	0.056	0.055	0.056
Mix no par.	0.32	0.33	0.33	0.35	0.31	0.33
Mix guided	0.21	0.22	0.22	0.23	0.20	0.22
Matrix chains	$\times$	$\times$	$\times$	$\times$	$\approx$	$\times$

Generalized Matrix Chain

[arXiv:1804.04021]



In practice

- Unary operators: transposition, inversion $(X := AB^TC^{-T}D + \dots)$
- Overlapping kernels (e.g., $L \leftarrow L^{-1}$, $X = A^{-1}B$)
- Decompositions (e.g., $A \rightarrow Q^TDQ$, $A \rightarrow LU$)
- Properties & specialized kernels (GEMM, TRMM, SYMM, ...)

Challenge: Not all flops were created equal [arXiv:1706.01341]

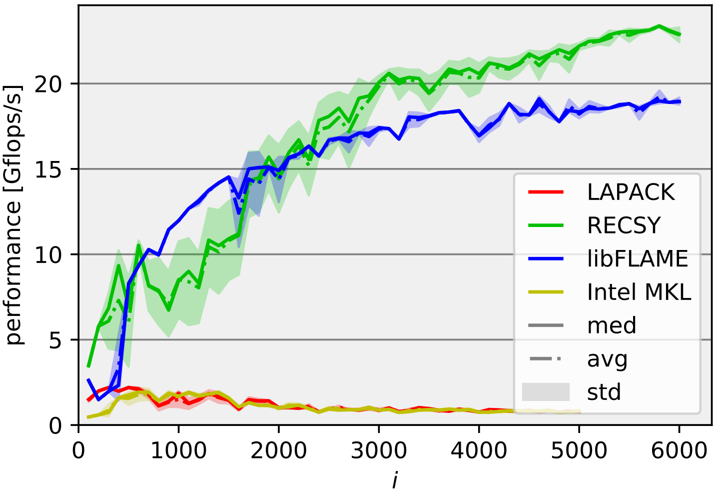
#FLOPs vs. execution time . . . vs. numerical stability

$$\underset{\mathcal{A}}{\text{argmin}}\,(\ \text{FLOPs}(\mathcal{A})\ ) \neq \underset{\mathcal{A}}{\text{argmin}}\,(\ \text{time}(\mathcal{A})\ )$$

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

Challenge: Not all flops were created equal [arXiv:1504.08035]

Triangular Sylvester equation – all libs performs the same # of flops



Challenge: Parallelism

multi-threaded libs, explicit multi-threading, runtime, hybrid, offloading

$$X := A((B^T C^{-T})D) \quad \text{vs.} \quad X := (AB^T)(C^{-T} D) \quad \text{vs.} \quad \ldots$$

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

Properties?

Operation	Property	Matlab	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	0.70	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.62	0.69	N/A	0.62	0.65	0.46
	SPD	**0.41**	0.60	0.63	N/A	**0.34**	0.62	0.31
	Triangular	**0.03**	**0.03**	0.63	N/A	0.62	0.65	0.03
	Diagonal	0.03	**0.01**	0.63	N/A	**0.03**	0.62	0.001
		$\approx$	$\approx$	$\times$	$\times$	$\approx$	$\times$	
Multiplication	Triangular	1.44	**0.75**	1.47	1.45	1.44	1.44	0.74
	Diagonal	1.44	**0.03**	1.47	1.45	1.42	1.44	0.06
		$\times$	✓	$\times$	$\times$	$\times$	$\times$	

Challenge: Inference of properties

- **easy** $E := L_1 * U^T * L_2$ triangular(E) ?
- **hard** $\lambda(L^{-T} A L^{-1})$ symmetric($L^{-T} A L^{-1}$)?
- **impossible?** $E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$ properties($I + U^T Q^{-1} U$) ?

$\Rightarrow$ **Symbolic analysis**: pattern matching

Common Subexpressions

[arXiv:1710.06915]

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

$\Rightarrow$ **Pattern matching**

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

Common Subexpressions?

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

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

Other features

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

$$M = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix} \quad \rightarrow ? \quad Mx = y, \quad y = Mx \quad \times$$

- diag$(A + B)$ vs. diag(A) + diag(B) $\rightarrow$ Armadillo

  diag(AB) vs. ... $\rightarrow$ $\times$

Summary

- LAMP is challenging — lots of expertise needed; interdisciplinary
- Compilers are great with scalars, not so much with matrices
- **Linnea**: A linear algebra compiler

Linear algebra knowledge: operators, identities, theorems

- Distributivity, commutativity, partitionings, ...
- $((QR)^T QR)^{-1}(QR)^T y \rightarrow (R^T Q^T QR)^{-1} R^T Q^T y \rightarrow R^{-1} R^{-T} R^T Q^T y \rightarrow R^{-1} Q^T y$
- $\text{SPD}(A) \rightarrow \text{SPD}(A_{BR} - A_{BL} A_{TL}^{-1} A_{BL}^T)$ Schur complement
- $\cdots$

Example

$$w := AB^{-1}c, \quad \text{SPD}(B)$$

Naive

```
w = A*inv(B)*c
```

Recommended

```
w = A*(B\c)
```

Expert

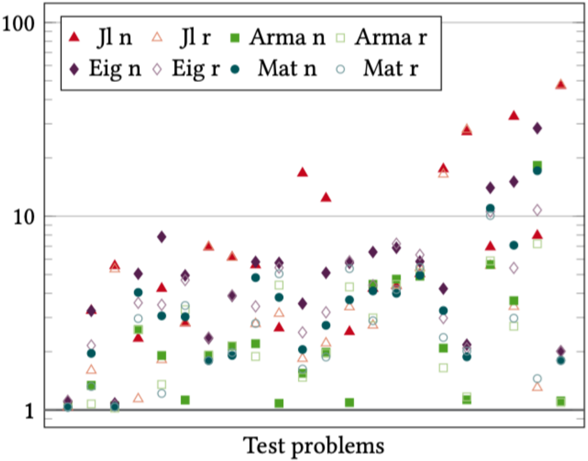
```
L = Chol(B)
w = A*(L'\(L\c))
```

Generated — "Linnea"

```
ml0 = A; ml1 = B; ml2 = c;
potrf!('L', ml1)
trsv!('L', 'N', 'N', ml1, ml2)
trsv!('L', 'T', 'N', ml1, ml2)
ml3 = Array{Float64}(10)
gemv!('N', 1.0, ml0, ml2, 0.0, ml3)
w = ml3
```

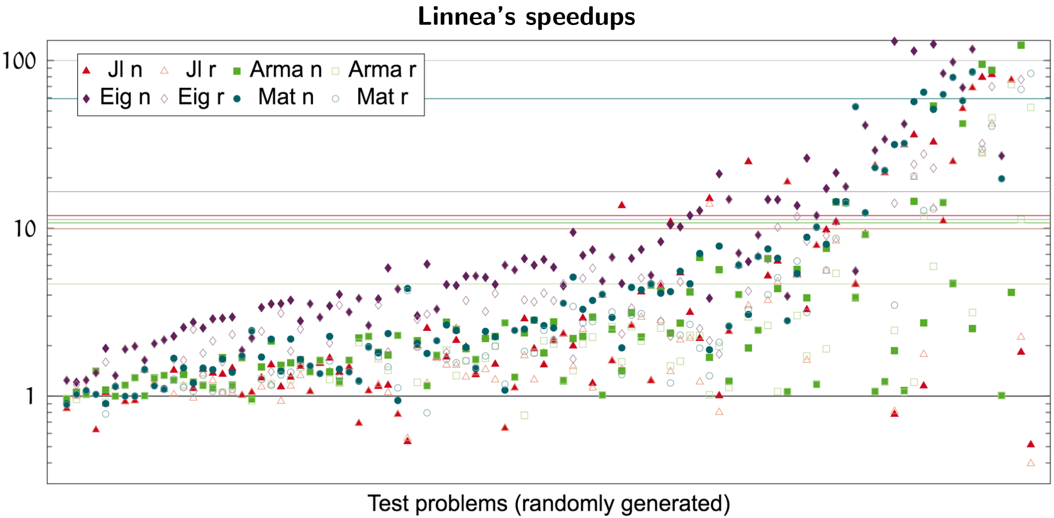
Results — applications

Linnea's speedups



Jl: Julia, **Arma**: Armadillo, **Eig**: Eigen, **Mat**: Matlab. **n**/**r**: naive/recommended implementation

Results — random expressions



Jl: Julia, **Arma**: Armadillo, **Eig**: Eigen, **Mat**: Matlab. **n/r**: naive/recommended implementation