Comparison of Low-Rank Update Techniques for $\mathcal{H}$-Arithmetic

R. Kriemann
MPI MIS Leipzig

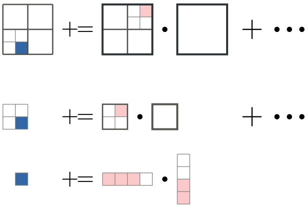
SIAM CSE21

Update Handling

Eager Update Evaluation

Updates are computed as soon as possible and immediately applied to leaf blocks.

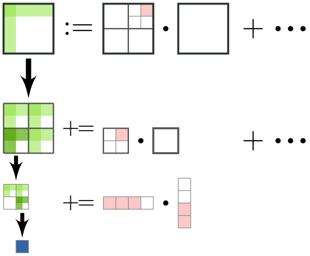
For low-rank matrices this induces a truncation. For structured matrices *multiple* truncations may result.



function HMUL(**in**: A, B, **inout**: C)
  if A, B, C are structured **then**
    for $i = 0, \ldots, n$ **do**
      for $j = 0, \ldots, m$ **do**
        for $k = 0, \ldots, \ell$ **do**
          `hmul`(A_{ik}, B_{kj}, C_{ij});
  else
    $T := A \cdot B$;
    if C is low-rank **then**
      $C :=$ `truncate`$(C + T)$;
    else if C is structured **then**
      for all sub-blocks C_{ij} of C **do**
        $C_{ij} :=$ `truncate`$(C_{ij} + T_{ij})$;
    else
      $C := C + T$;

Accumulated Updates

Updates are computed and collected *per level* and *shifted down* to sub blocks.



Reduced number of low-rank truncations.

Start for $C := C + A \cdot B$:

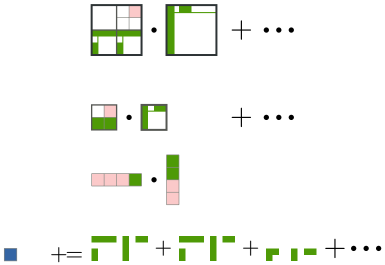
```
hmul(C, {A, B}, 0);
```

function HMUL(**inout**: C, **in**: $U_C, \mathcal{U}_C$)
  // Compute non-recursive updates
  for all $A^k, B^k \in \mathcal{U}_C$ **do**
    if A^k or B^k is not structured **then**
      $U_C :=$ truncate$(U_C + A^k \cdot B^k)$;
      $\mathcal{U}_C := \mathcal{U}_C \setminus \{(A^k, B^k)\}$;
  if C is structured **then**
    // Push down recursive updates
    for $i = 0, \ldots, n$ **do**
      for $j = 0, \ldots, m$ **do**
        $\mathcal{U}_{C_{ij}} = \emptyset$;
        for all $(A^k, B^k) \in \mathcal{U}_C$ **do**
          for $\ell = 0, \ldots, r$ **do**
            $\mathcal{U}_{C_{ij}} := \mathcal{U}_{C_{ij}} \cup \left\{(A^k_{i\ell}, B^k_{\ell,j})\right\}$;
        hmul$(C_{ij}, U_C|_{t_i,s_j}, \mathcal{U}_{C_{ij}})$;
  else
    // Apply all accumulated updates
    $C :=$ truncate$(C + U_C)$;

S. Börm: "Hierarchical matrix arithmetic with accumulated updates", Comput. Visual Sci., 20, 71–84 (2019)

Lazy Update Evaluation

Updates are handled *implicitly* until leaves are reached, i.e., low-rank (and dense) blocks are (virtually) sub-divided.



function HMUL(**inout**: $C_{t \times s}$, **in**: $\mathcal{U}$)
  if $C_{t \times s}$ is structured **then**
    // Push down updates
    for $t_i \in \mathcal{S}(t)$ **do**
      for $s_j \in \mathcal{S}(s)$ **do**
        $\mathcal{U}' = \emptyset;$
        for all $(A^k_{t \times r}, B^k_{r \times s}) \in \mathcal{U}$ **do**
          $\mathcal{U}' = \mathcal{U}' \cup \{(A^k|_{t_i \times r}, B^k|_{r \times s_j})\};$
        hmul$(C_{t_i \times s_j}, \mathcal{U}');$
  else
    // Apply all updates
    $C := \texttt{truncate}(C + \sum_k A^k \cdot B^k);$

All updates are applied *simultaneously*.

Number of updates in $\mathcal{O}(\log n)$.

J. Dölz, H. Harbrecht, M.D. Multerer: "On the Best Approximation of the Hierarchical Matrix Product", SIAM J. Matrix Anal. Appl., 40(1), 147–174 (2019)

Lowrank Approximation

Lowrank Approximation

Singular Value Decomposition (SVD)

Rank Revealing QR (RRQR)

Randomized LR/SVD (RandLR/SVD)

Cross Approximation (ACA)

Lánczos Bidiagonalization (Lanzcos)

Computes best approximation.

Runtime complexity is $\mathcal{O}\left(n \cdot k^2 + k^3\right)$ for input rank k and block size n.

function SVD(**in**: U, V, ε, **out**: W, X)
$[Q_U, R_U] :=$ `qr(U);`
$[Q_V, R_V] :=$ `qr(V);`
$[U_s, S_s, V_s] :=$ `svd(` $R_U \cdot R_V^H$ `);`
$k :=$ `rank(`S_s, ε`);`
$W := Q_U \cdot U_s(:, 1:k) \cdot S_s(1:k, 1:k)$;
$X := Q_V \cdot V_s(:, 1:k)$;

Lowrank Approximation

Singular Value Decomposition (SVD)

Rank Revealing QR (RRQR)

Randomized LR/SVD (RandLR/SVD)

Cross Approximation (ACA)

Lánczos Bidiagonalization (Lanzcos)

Based on reordering the remaining columns during QR.

Approximation rank and error control defined by matrices $R(i:k, i:k)$.

function RRQR(**in**: U, V, ε, **out**: W, X)
 $k := \texttt{rank}(U)$;
 $[Q_V, R_V] = \texttt{qr}(V)$;
 $[Q, R, P] = \texttt{qrp}(U \cdot R_V^H)$;
 for $i = 1, \ldots, k$ **do**
  $S(i) := \left\| R(i:k, i:k) \right\|_F$;
 $k' := \texttt{rank}(S, \varepsilon)$;
 $W := Q(:, 1:k')$;
 $X := Q_V \cdot P \cdot R(1:k', :)^H$;

Lowrank Approximation

Singular Value Decomposition (SVD)

Rank Revealing QR (RRQR)

Randomized LR/SVD (RandLR/SVD)

Cross Approximation (ACA)

Lánczos Bidiagonalization (Lanzcos)

Approximate column basis of operator.

Only operator evaluation required.

function RandLR(**in**: M, ε, **out**: W, X)
  $W :=$ `ColumnBasis`(M, ε);
  $X := M^H \cdot W$;

function RandSVD(**in**: M, ε, **out**: W, X)
  $B :=$ `ColumnBasis`(M, ε);
  $[Q, R] :=$ `qr`$(M^H \cdot B)$;
  $[U_s, S_s, V_s] :=$ `svd`(R);
  $k :=$ `rank`(S_s, ε);
  $W := B \cdot V_s(:, 1 : k) S(1 : k, 1 : k)$;
  $X := M^H \cdot B \cdot U_s(:, 1 : k)$;

Lowrank Approximation

Singular Value Decomposition (SVD)

Rank Revealing QR (RRQR)

Randomized LR/SVD (RandLR/SVD)

Cross Approximation (ACA)

Lánczos Bidiagonalization (Lanzcos)

Successively selects pairs of rows/columns for rank-1 updates.

Only *requested coefficients* needed.

Different pivot search strategies available.

function ACA(**in**: M, ε, **out**: W, X)
  $c_1 = 1$;
  for $i = 1, \dots$ **do**
    $w_i := \texttt{column}(M, c_i) - W \cdot X(c_i, :)'$;
    $r_i := \texttt{maxidx}(w_i); w_i := w_i / w_i(r_i)$;
    $x_i := \texttt{row}(M, r_i)' - X \cdot W(r_i, :)'$;
    $W := [W, w_i]; X := [X, x_i]$;
    if $\left\| w_i \cdot x_i' \right\|_F \leq \varepsilon \left\| W \cdot X^H \right\|_F$ **then**
      break;
    $c_{i+1} := \texttt{maxidx}(x_i)$;

Lowrank Approximation

Singular Value Decomposition (SVD)

Rank Revealing QR (RRQR)

Randomized LR/SVD (RandLR/SVD)

Cross Approximation (ACA)

Lánczos Bidiagonalization (Lanzcos)

For $M \in \mathbb{C}^{n \times m}$, iteratively computes bases W_k and X_k of

$$\mathcal{K}(MM^H, w_1) = \text{span}\left\{(MM^H)^i w_1 : 0 \leq i \leq k\right\}$$
$$\mathcal{K}(M^H M, x_1) = \text{span}\left\{(M^H M)^i x_1 : 0 \leq i \leq k\right\}$$

with random w_1, $x_1 = M^H w_1 / \left\|M^H w_1\right\|$ such that

$$M \approx W_k B_k X_k^H$$

and bidiagonal B_k.

Also only operator evaluation required.

Model Problems

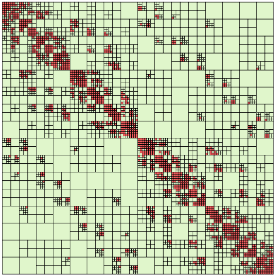
Model Problems

Laplace SLP

Defined by

$$\int_\Gamma \frac{1}{\|x - y\|_2} u(x) dy = f(x), \quad x \in \Gamma$$

with $\Gamma = \{x \in \mathbb{R}^3 : \|x\|_2 = 1\}$.



Matrix condition: $500, \ldots, 5000$.

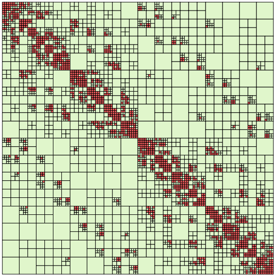
Model Problems

Laplace SLP

Defined by

$$\int_\Gamma \frac{1}{\|x-y\|_2} u(x) dy = f(x), \quad x \in \Gamma$$

with $\Gamma = \left\{ x \in \mathbb{R}^3 : \|x\|_2 = 1 \right\}$.



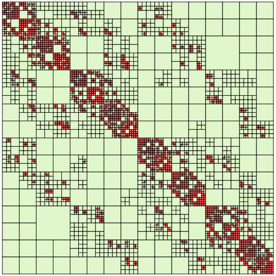
Matrix condition: $500, \ldots, 5000$.

Matérn Covariance

Defined by

$$C(x, y; \theta) = \frac{\sigma^2}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\|x-y\|}{\ell} \right)^\nu K_\nu \left(\frac{\|x-y\|}{\ell} \right)$$

with parameters $\theta = (\sigma, \ell, \nu)$ and random positions in $[0, 1]^3$.

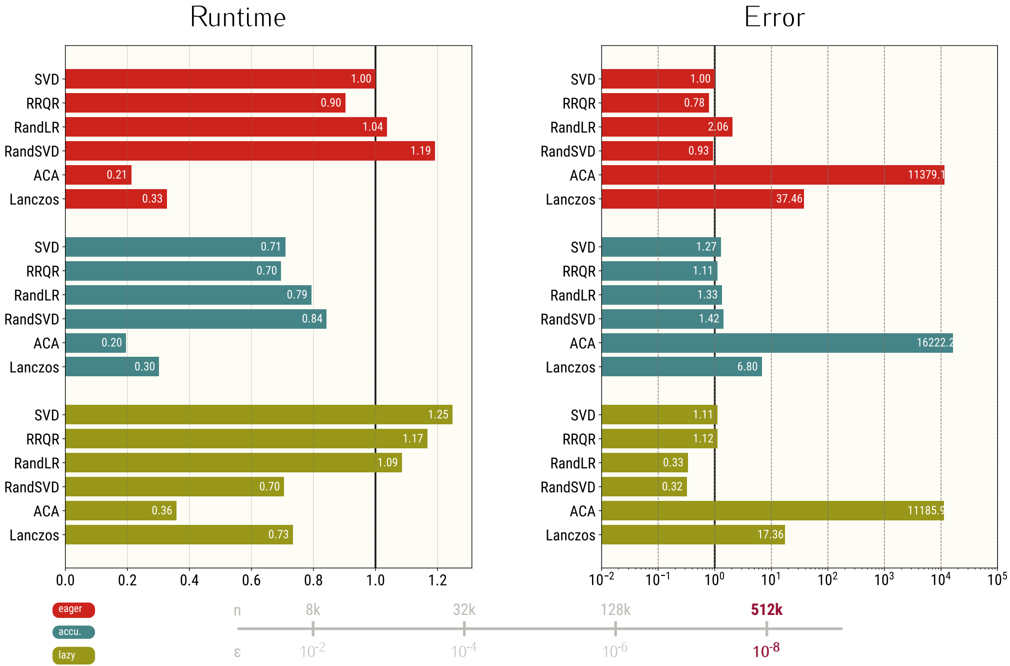


Matrix condition: $10^6, \ldots, 10^8$.

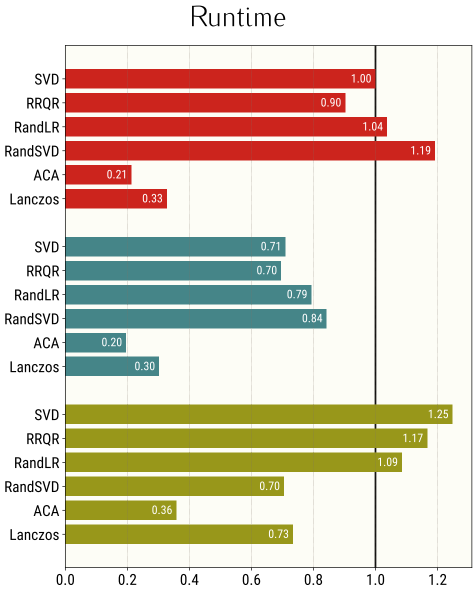
Results

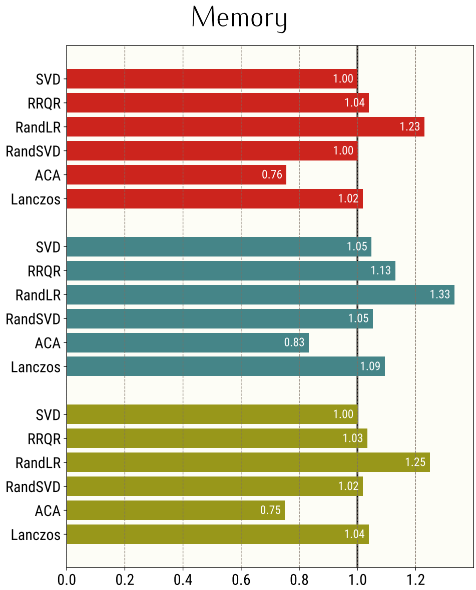
Laplace SLP: H-LU

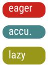
Laplace SLP: H-LU



Laplace SLP: H-LU





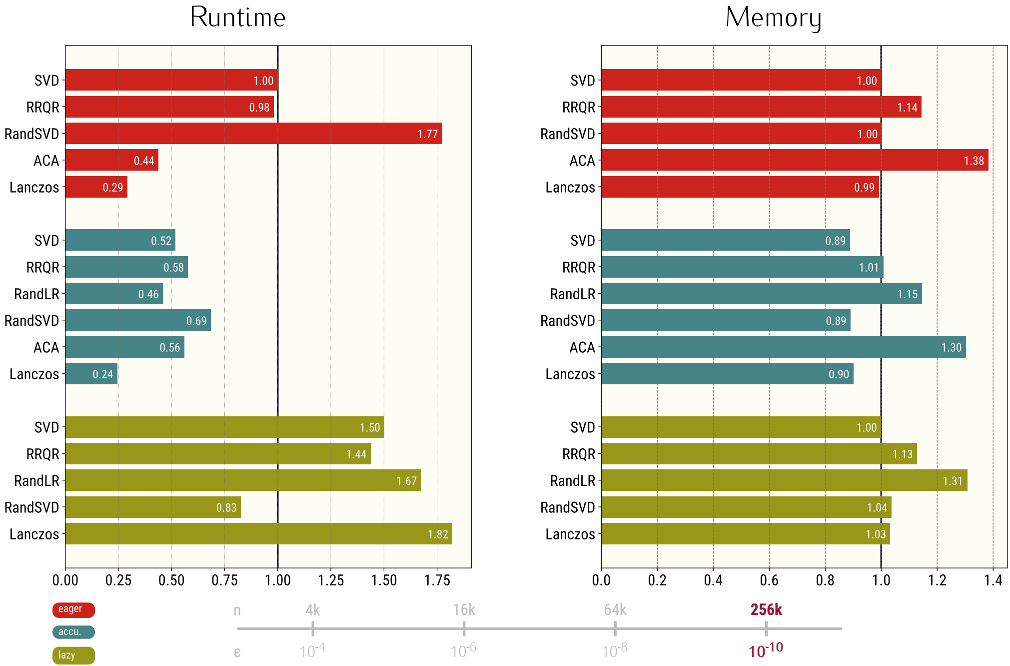


Matérn Covarience: H-LU

Matérn Covarience: H-LU

Matérn Covarience: H-LU

Matérn Covarience: H-LU



Conclusion

Based on the presented results:

- RRQR is a safe, faster replacement for SVD,
- accumulator arithmetik is faster but less accurate,
- ACA may shine or fail,
- Lanczos is fast but has accuracy issues,
- randomized methods *efficiently* exploit lazy arithmetic with *high* accuracy.

Conclusion

Based on the presented results:

- RRQR is a safe, faster replacement for SVD,
- accumulator arithmetik is faster but less accurate,
- ACA may shine or fail,
- Lanczos is fast but has accuracy issues,
- randomized methods *efficiently* exploit lazy arithmetic with *high* accuracy.

More comparisons available at

libhlr.org/programs/approx

Conclusion

Based on the presented results:

- RRQR is a safe, faster replacement for SVD,
- accumulator arithmetik is faster but less accurate,
- ACA may shine or fail,
- Lanczos is fast but has accuracy issues,
- randomized methods *efficiently* exploit lazy arithmetic with *high* accuracy.

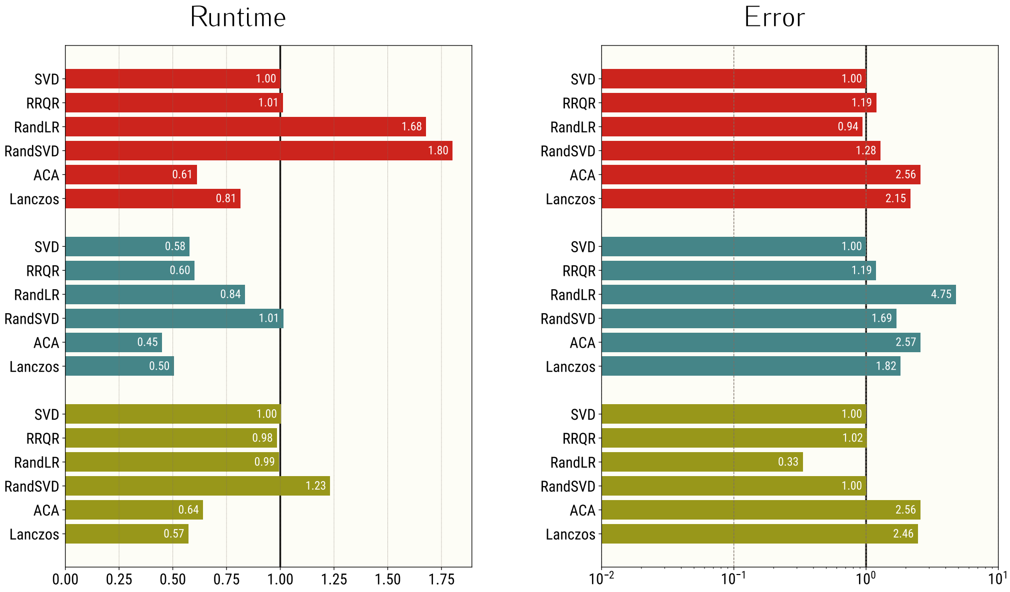
More comparisons available at

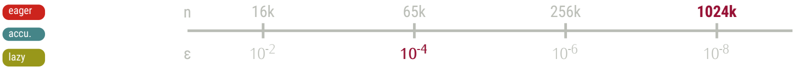
libhlr.org/programs/approx

Next:

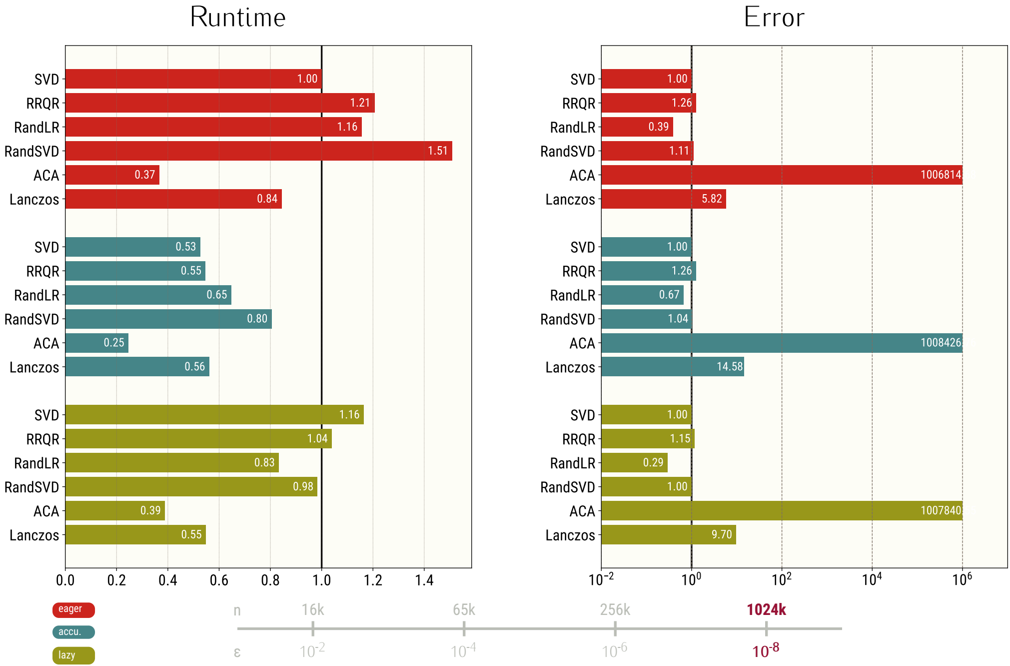
different ***H^2-arithmetic*** with different approximation techniques

LogKernel / HODLR: $\mathcal{H}$-LU





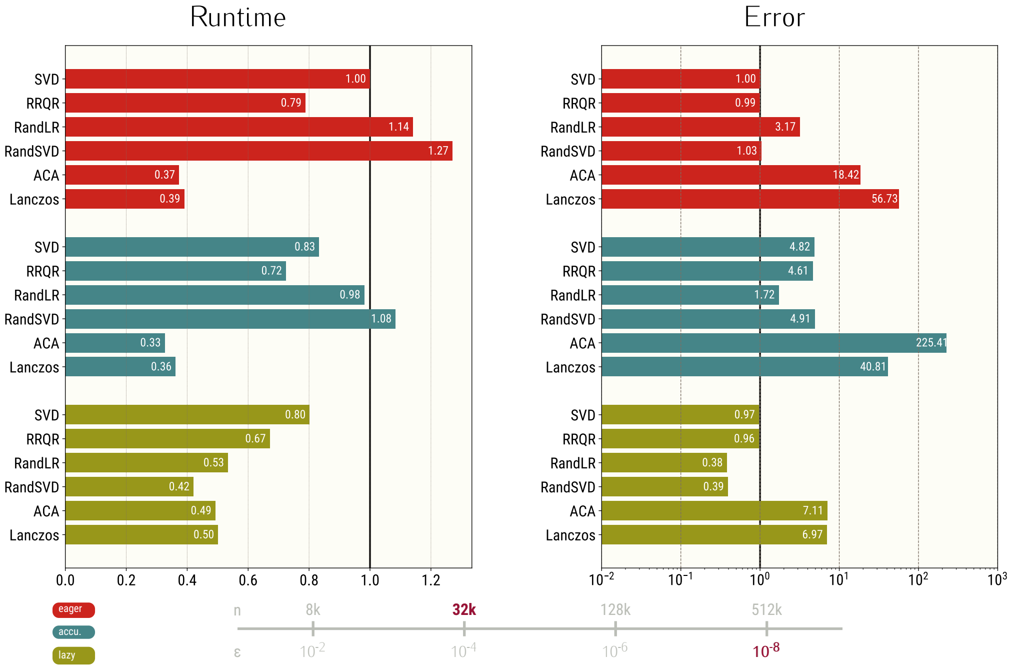
LogKernel / HODLR: H-LU



LogKernel / HODLR: H-LU

LaplaceSLP / TLR: H-LU

LaplaceSLP / TLR: H-LU



LaplaceSLP / TLR: H-LU