Easy* Python wrappers for RandBLAS and RandLAPACK

Riley Murray
Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This presentation describes objective technical results and analysis. Any subjective views or opinions that might be expressed here do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

RandBLAS in action: subset selection

Sandia National Laboratories

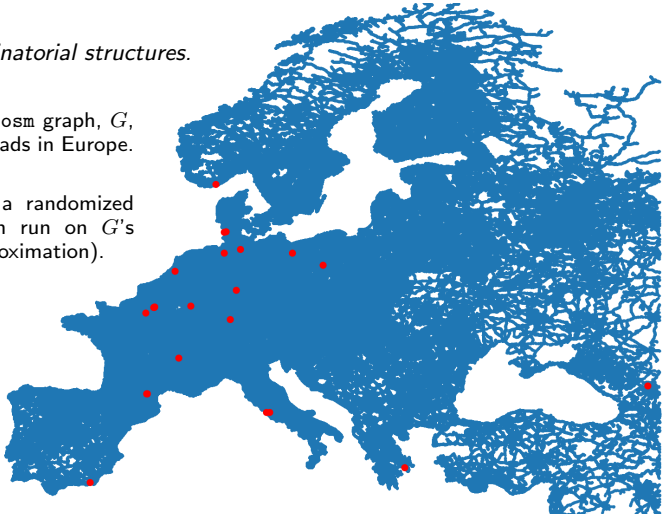
Matrices often encode combinatorial structures.

Right: the DIMACS10 `europe-osm` graph, G, representing 50 million major roads in Europe.

Red nodes were obtained by a randomized Golub-Klema-Stewart algorithm run on G's adjacency matrix (rank-32 approximation).

Algorithm ingredients:

- ⋆ Gaussian sampling
- ⋆ SPMM
  - QR & QRCP
  - dense BLAS



Householder QRCP with randomized, block pivoting

Sandia National Laboratories

Independently discovered by Duersch and Gu [DG17] and Martinsson [Mar15].

Refined in [MOHvdG17] and [XGL17]. Recently revitalized by [MMK+25].

Given $\mathbf{M} \in \mathbb{R}^{m \times n}$, find $\mathbf{Q}$, $\mathbf{R}$, $\boldsymbol{J}$

- Select a block size b (big, but $\ll n$) and tuning parameter $1 \leq \gamma < 2$ (recommend $= 1$).
- Sample a Gaussian operator $\mathbf{S} \in \mathbb{R}^{\gamma b \times m}$, and sketch $\mathbf{M}^{\mathrm{sk}} = \mathbf{S}\mathbf{M}$.

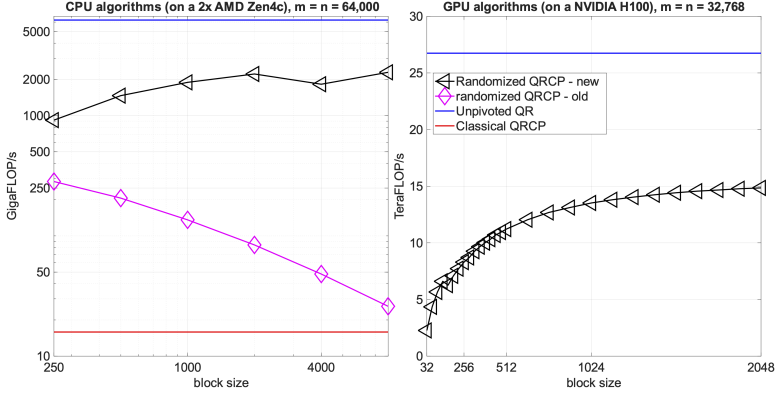
Main loop for $i = 1 : n/b$

1. $[\sim, \mathbf{R}^{\mathrm{sk}}, \boldsymbol{J}_i] = \texttt{qrcp\_wide}(\mathbf{M}^{\mathrm{sk}})$
2. $\mathbf{M}^{\mathrm{perm}} = \texttt{col\_perm}(\mathbf{M}, \boldsymbol{J}_i)$
3. $k = \texttt{rank\_est}(\mathbf{R}^{\mathrm{sk}})$
4. $[\mathbf{Q}_i, \mathbf{R}_i^{11}] = \texttt{qr\_tall}(\mathbf{M}^{\mathrm{perm}}, k)$
5. $[\mathbf{M}^{\mathrm{update}}, \mathbf{R}_i^{12}] = \texttt{apply\_trans\_q}(\mathbf{Q}_i, \mathbf{M}^{\mathrm{perm}})$
6. Update $\mathbf{M}$ with $\mathbf{M}^{\mathrm{update}}$
7. Update $\mathbf{Q}$ with $\mathbf{Q}_i$; $\mathbf{R}$ with $\mathbf{R}_i^{11}$ and $\mathbf{R}_i^{12}$; $\boldsymbol{J}$ with $\boldsymbol{J}_i$
8. Check termination criteria
9. Update $\mathbf{M}^{\mathrm{sk}}$

Speed of QR and QRCP algorithms

Sandia National Laboratories

"**Randomized QRCP – new**" is our implementation described in [MMK+25].



We can get a **100x speedup** over a classical algorithm – for a *full matrix decomposition*.

The algorithm – which is notoriously hard to parallelize – runs beautifully on GPUs.

A glimpse of the RandBLAS API

Sandia National Laboratories

```
using namespace RandBLAS; using namespace blas;
// step 1
RNGState state();
// step 2
DenseDist D(10000, 50);
DenseSkOp<double> S(D, state);
// step 3
double B* = new double[20000 * 50];
sketch_general(
      Layout::ColMajor, Op::NoTrans, Op::NoTrans,
      20000, 50,  10000, 1.0, A, 20000, S, 0.0, B, 20000
); // B = AS
```

RandBLAS types involved in sketching:

- Templated `RNGState` structs.
- Implementations of ...
  - `SketchingDistribution`
  - `SketchingOperator`

Three powerful aspects of RandBLAS' API:

1. No hidden machine state affects $\mathbf{S} = \{\texttt{dist}, \texttt{state}\}$ as a mathematical object.
2. `SketchingOperator` objects are constructed in $O(1)$ time and space.
3. We have procedural, random(ish) access to submatrices of sketching operators.

This tutorial

Sandia National Laboratories

1. `git clone https://github.com/rileyjmurray/midwest-randnla-2026.git`
2. Look at `project_instructions.md`.
   - Pull the Docker image and launch a container.
   - Setup conda. Build and install RandLAPACK within the container.
3. A "Hello World"
   - change directory into `bindings`
   - run `pip install -e .`
   - run `pytest tests`
   - Prompt: Please read AGENTS.md, then create a function ‘`suparla.add(a, b)`‘ that takes two float64 scalars and returns their sum.
4. Making real bindings!
   - Prompt: Help me expose RandLAPACK's `BQRRP` algorithm to Python so it can operate on NumPy arrays.
   - If you encounter errors, paste them into the chat window and ask how to proceed.
5. Possible next steps:
   - Make a demo script that compares speed of SciPy's QR with `pivoting=True` to `BQRRP`.
   - Expose some of `BQRRP`'s subroutine options to the Python interface.
   - Modify `BQRRP`'s C++ source so that it can terminate at a prescribed rank.

Thank you!

And thank the U.S. Government for funding this research.

- This work was supported by Laboratory Directed Research and Development (LDRD) funding from Sandia National Laboratories, operated for the US National Nuclear Security Administration, under Contract No. DE-NA0003525.
- Precursor work was partially supported by National Science Foundation grants (2004235, 2004541, and 2004763) to the International Computer Science Institute, the University of Tennessee's ICL, and the University of California at Berkeley.
- Precursor work was supported by LDRD funding from Berkeley Lab, operated for the US Department of Energy, Office of Science, under Contract No. DE-AC02-05CH11231.

References I

Sandia National Laboratories

[DG17] Jed A. Duersch and Ming Gu, *Randomized QR with column pivoting*, SIAM Journal on Scientific Computing **39** (January 2017), no. 4, C263–C291.

[KS16] R. Kyng and S. Sachdeva, *Approximate Gaussian elimination for Laplacians - fast, sparse, and simple*, 2016 IEEE 57th annual Symposium on Foundations of Computer Science (FOCS), October 2016, pp. 573–582.

[Mar15] P. G. Martinsson, *Blocked rank-revealing QR factorizations: How randomized sampling can be used to avoid single-vector pivoting*, 2015.

[MMK+25] Maksim Melnichenko, Riley Murray, William Killian, James Demmel, Michael W. Mahoney, Piotr Luszczek, and Mark Gates, *Anatomy of high-performance column-pivoted qr decomposition*, 2025.

[MOHvdG17] Per-Gunnar Martinsson, Gregorio Quintana Ortı, Nathan Heavner, and Robert van de Geijn, *Householder QR factorization with randomization for column pivoting (HQRRP)*, SIAM Journal on Scientific Computing **39** (January 2017), no. 2, C96–C115.

[XGL17] Jianwei Xiao, Ming Gu, and Julien Langou, *Fast parallel randomized QR with column pivoting algorithms for reliable low-rank matrix approximations*, 2017 IEEE 24th international conference on high performance computing (HiPC), 2017, pp. 233–242.

Example: pivot quality with randomized and classical QRCP

Sandia National Laboratories

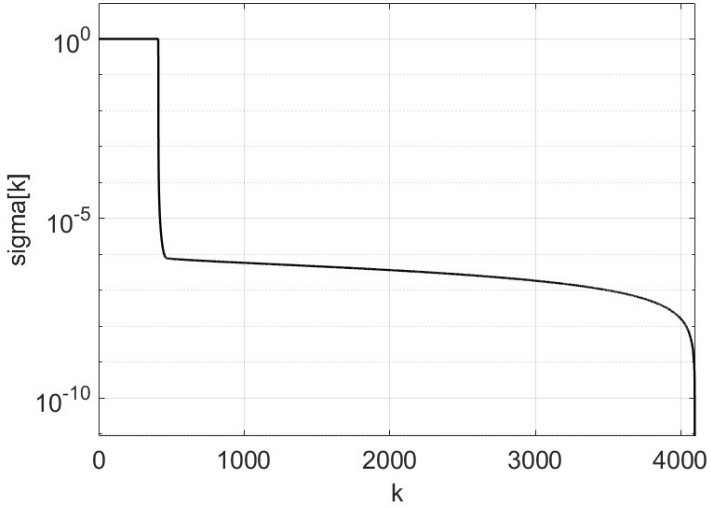


Figure: True singular values of test matrix $\mathbf{M}$.

Example: pivot quality with randomized and classical QRCP

Sandia National Laboratories

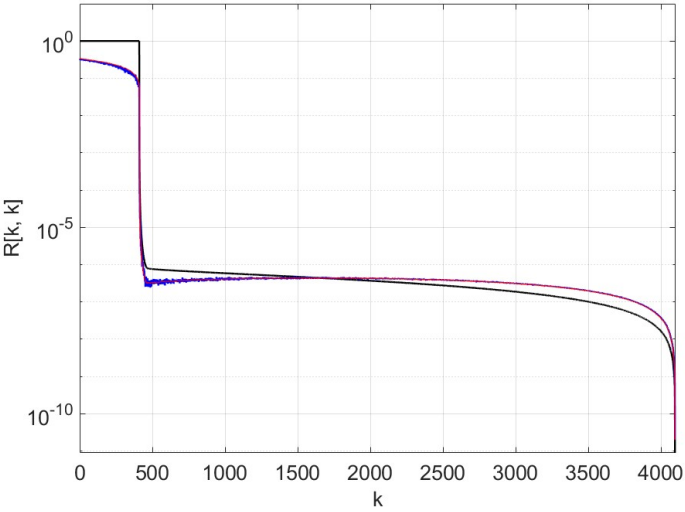


Figure: BQRRP and **classical QRCP**: diagonal entries of **R**.

Example: pivot quality with randomized and classical QRCP

Sandia National Laboratories

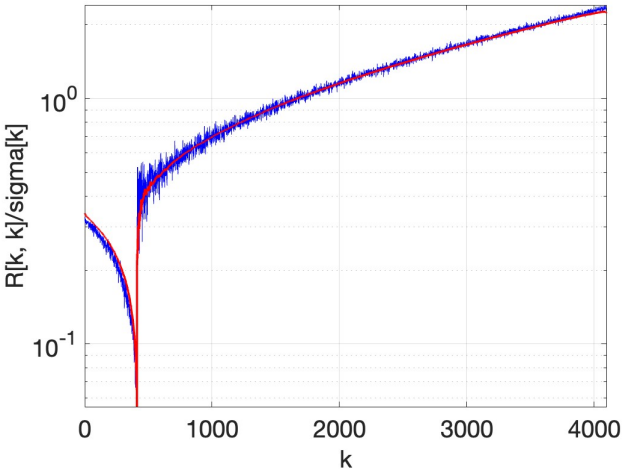


Figure: BQRRP and **classical QRCP**: ratios of $\mathrm{diag}(\mathbf{R})$ and singular values of $\mathbf{M}$.

RandBLAS' sparse matrix datastructures

Sandia National Laboratories

In RandBLAS 1.0:

- The common interface for our sparse matrix types
- COOMatrix
- CSRMatrix
- CSCMatrix

Operations with sparse matrices

Sketching

- $\mathbf{B} = \alpha \cdot \mathrm{op}(\mathrm{submat}(\mathbf{S})) \cdot \mathrm{op}(\mathbf{A}) + \beta \cdot \mathbf{B}$
- $\mathbf{B} = \alpha \cdot \mathrm{op}(\mathbf{A}) \cdot \mathrm{op}(\mathrm{submat}(\mathbf{S})) + \beta \cdot \mathbf{B}$

Deterministic operations

- $\mathbf{C} = \alpha \cdot \mathrm{op}(\mathbf{A}) \cdot \mathrm{op}(\mathbf{B}) + \beta \cdot \mathbf{C}$, with sparse $\mathbf{A}$
- $\mathbf{C} = \alpha \cdot \mathrm{op}(\mathbf{A}) \cdot \mathrm{op}(\mathbf{B}) + \beta \cdot \mathbf{C}$, with sparse $\mathbf{B}$

Flexible memory ownership policies:

Public Functions

inline **CSRMatrix**(int64_t n_rows, int64_t n_cols)

Standard constructor. Initializes n_rows and n_cols at the provided values. The vals, rowptr, and colidxs members are set to null pointers; nnz is set to zero, index_base is set to Zero, and CSRMatrix::own_memory is set to true.

This constructor is intended for use with reserve_csr(int64_t nnz, CSRMatrix &A).

inline **CSRMatrix**(int64_t n_rows, int64_t n_cols, int64_t nnz, T *vals, sint_t *rowptr, sint_t *colidxs, IndexBase index_base = IndexBase::Zero)

Expert constructor. Arguments passed to this function are used to initialize members of the same names; CSRMatrix::own_memory is set to false.

On RandBLAS' main branch, to appear in version 1.1:

$$\mathbf{B} = \alpha \cdot \mathrm{op}(\mathbf{A})^{-1} \cdot \mathbf{B}$$

where $\mathbf{A}$ is sparse triangular (upper or lower, CSC or CSR) and $\mathbf{B}$ is dense.

The BLAS data model

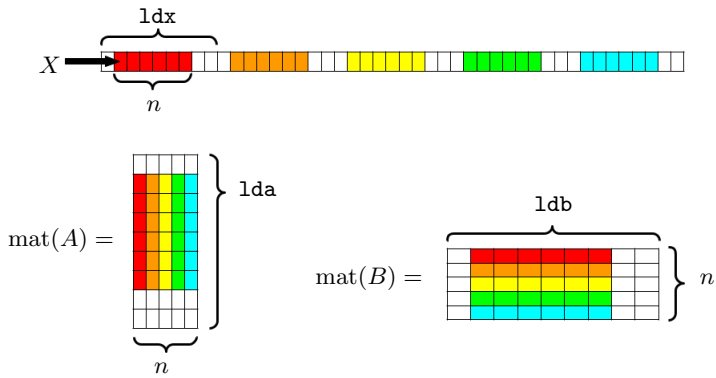
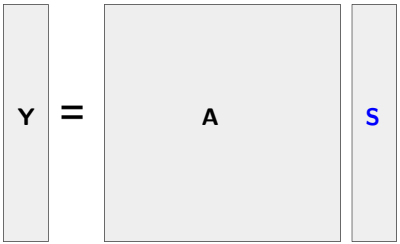


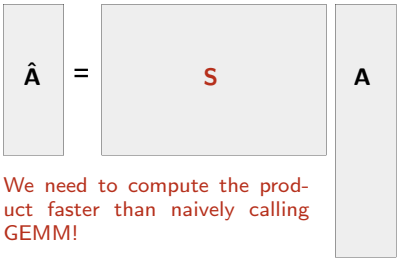
Figure: Visualization of how a buffer, pointed to by X, can be read in different ways to obtain two different matrices. We get $\mathrm{mat}(A)$ by setting $A = X$ and reading in column-major order with stride $\texttt{lda} = \texttt{ldx}$. We get $\mathrm{mat}(B)$ by setting $B = X$ and reading in row-major order with stride $\texttt{ldb} = \texttt{ldx}$. Mathematically, we have $\mathrm{mat}(A)^* = \mathrm{mat}(B)$.

Considerations in high-performance sketching

Sandia National Laboratories

There are two regimes for sketching: and *sampling* and *embedding*





We need to compute the product faster than naively calling GEMM!

- ✓ Use vendor BLAS to multiply when both $(\mathbf{S}, \mathbf{A})$ are dense.
- ✓ Use OpenMP for sparse matrix multiplies and generating $\mathbf{S}$.
- ✓ Beware cache effects and row-major vs column-major memory layout.