
RandLinearAlgebra

If the docker is not working

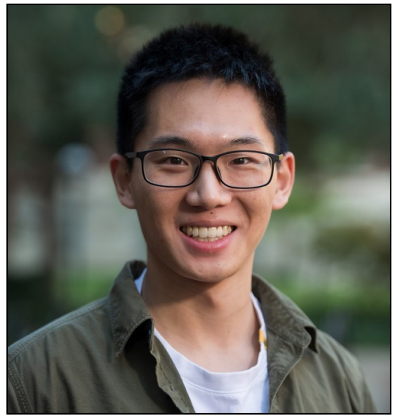
You can find all the python notebooks here:

(`https://github.com/nathanielpritchard/RandLinearAlgebra_examples`)

Contributors

Vivak Patel

Adrian Maldonado



Tunan Wang

Giselle Labrador Badia

Christian Varner

Tongtong Jin

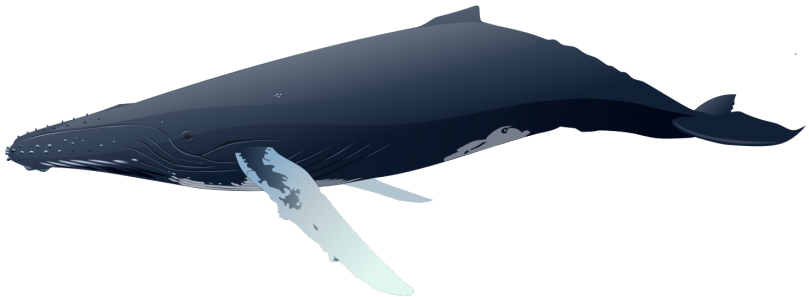
This project is supported by NSF Awards 2309445 and 2309446.

A typical RandNLA Algorithm

1. Use randomization to make a matrix smaller

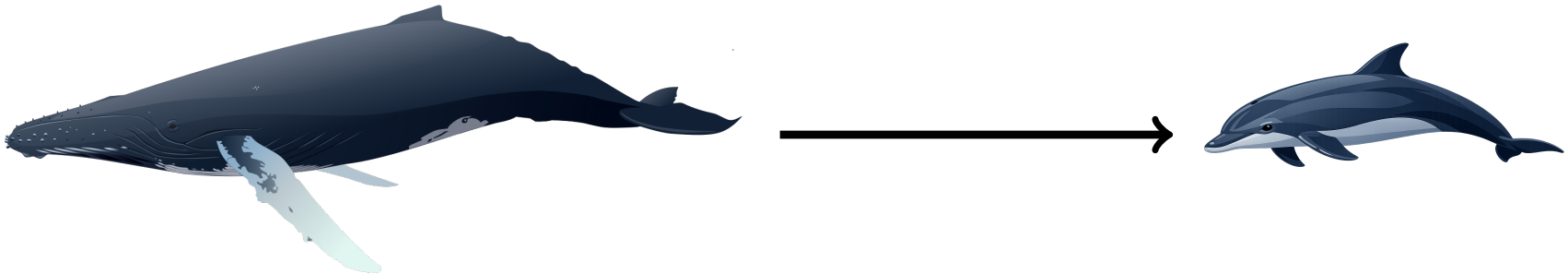
A typical RandNLA Algorithm

1. Use randomization to make a matrix smaller



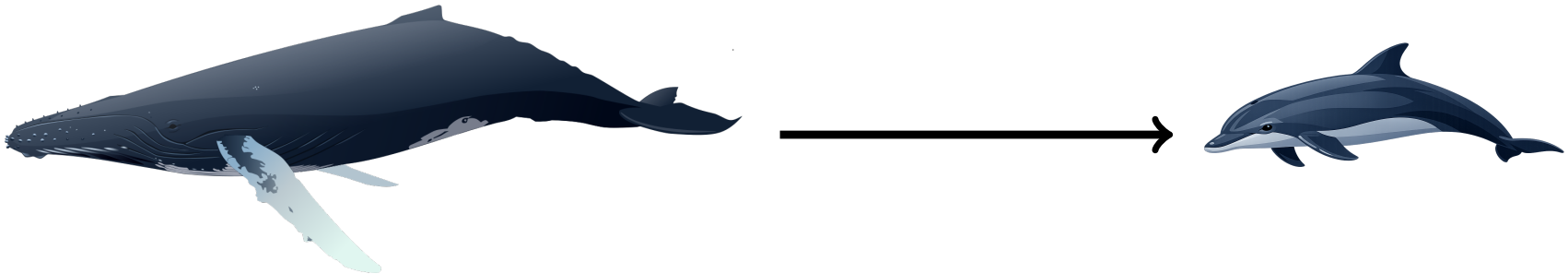
A typical RandNLA Algorithm

1. Use randomization to make a matrix smaller



A typical RandNLA Algorithm

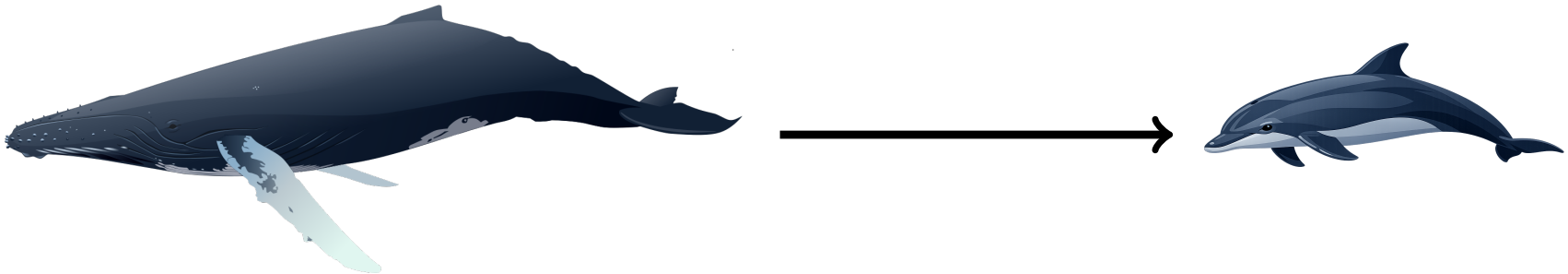
1. Use randomization to make a matrix smaller



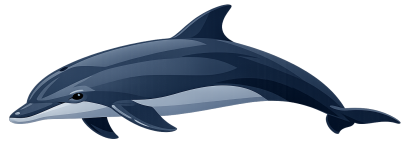
2. Perform a classical NLA operation

A typical RandNLA Algorithm

1. Use randomization to make a matrix smaller



2. Perform a classical NLA operation

SVD()

How do we do this?

Theory often tells us how one set of choices performs on the worst possible problem. But,

- What if my problem has much better properties than in the worst case?

How do we do this?

Theory often tells us how one set of choices performs on the worst possible problem. But,

- What if my problem has much better properties than in the worst case?
- What if it's sparse?

How do we do this?

Theory often tells us how one set of choices performs on the worst possible problem. But,

- What if my problem has much better properties than in the worst case?
- What if it's sparse?
- What if I don't have direct access to my system?

How do we do this?

Theory often tells us how one set of choices performs on the worst possible problem. But,

- What if my problem has much better properties than in the worst case?
- What if it's sparse?
- What if I don't have direct access to my system?
- What if I want to use a GPU?

How do we do this?

Theory often tells us how one set of choices performs on the worst possible problem. But,

- What if my problem has much better properties than in the worst case?
- What if it's sparse?
- What if I don't have direct access to my system?
- What if I want to use a GPU?
- What if I want to use mixed precision?

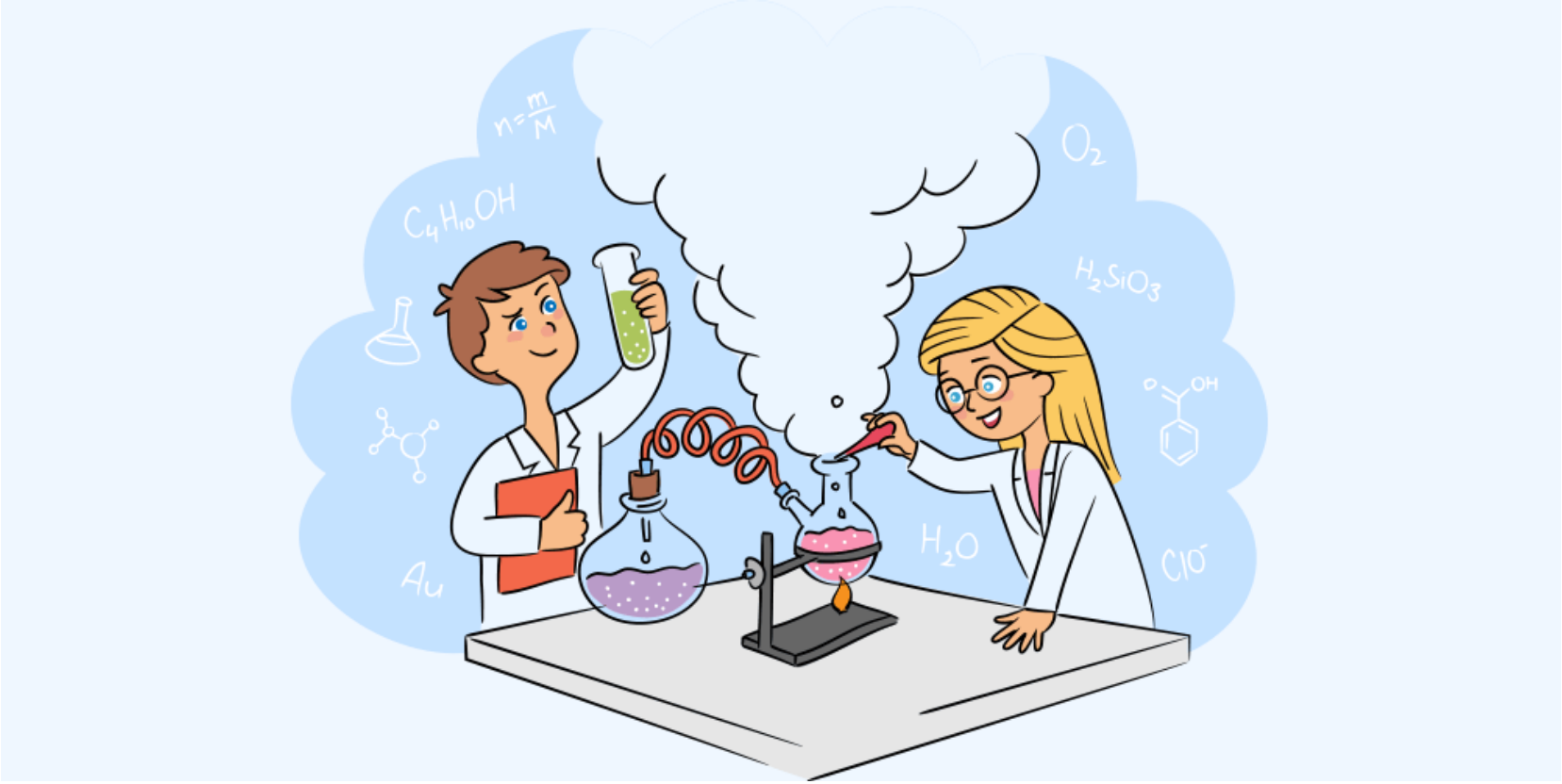
How do we do this?

Theory often tells us how one set of choices performs on the worst possible problem. But,

- What if my problem has much better properties than in the worst case?
- What if it's sparse?
- What if I don't have direct access to my system?
- What if I want to use a GPU?
- What if I want to use mixed precision?

Then how should I implement these algorithms?

You just have to try...



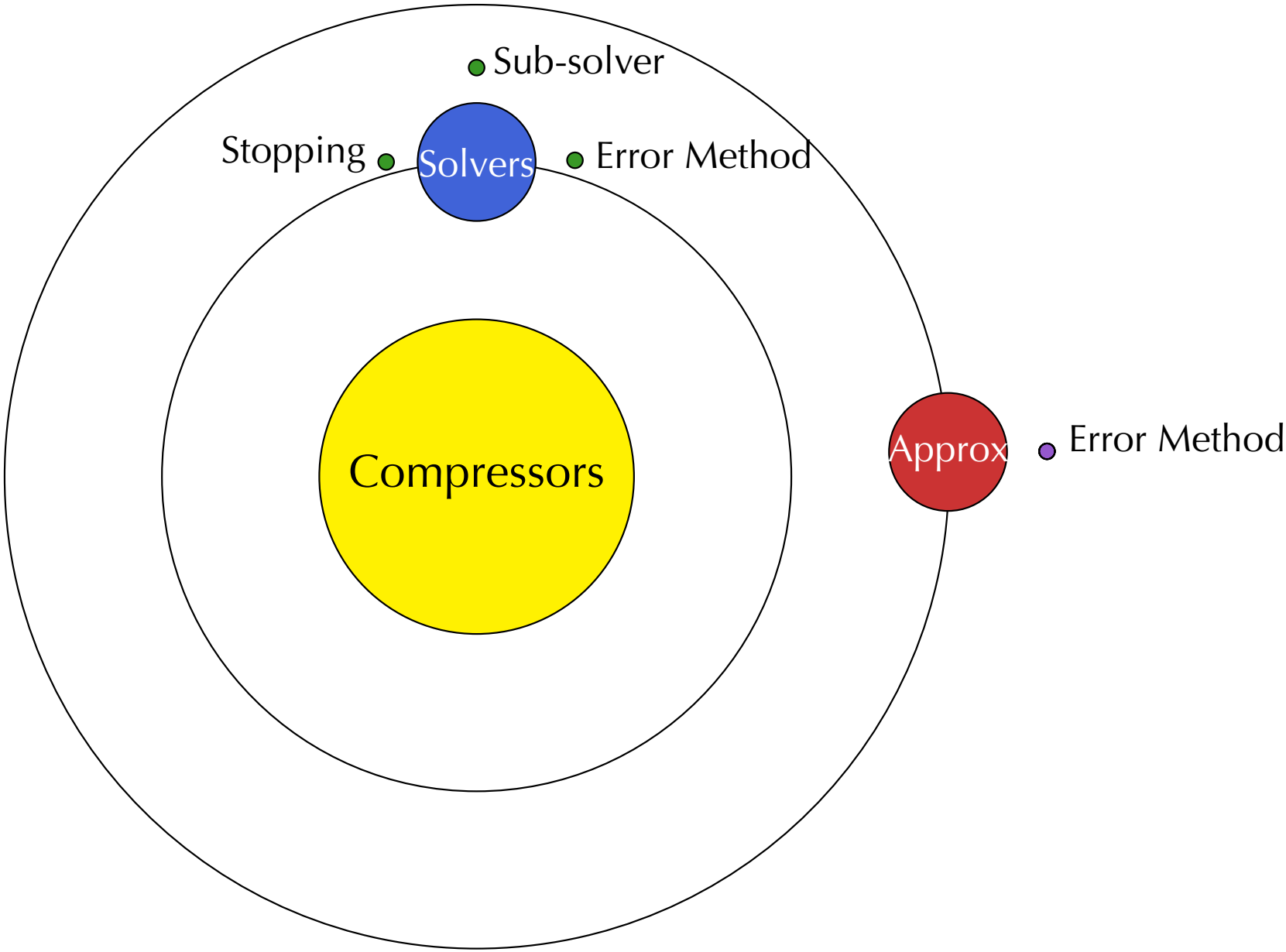
RandLinearAlgebra.jl: A library for rapid prototyping

- RandLinearAlgebra.jl is a **modular** Julia library
- Allows for **easy, in-depth** customization of RandNLA routines
- Can be plugged into established Julia codebases
- Julia also makes it easy to test on GPUs and with different levels of precision

Goals for this talk

1. Understand the structure of `RandLinearAlgebra.jl`
2. Understand how to use the library
   - **Attention** (How to compress a matrix)
   - **Preconditioning** (How to Solve a linear system)
   - **Operator Inference** (How to Approximate a matrix)

The Solar System of RandLinearAlgebra.jl Methods



Key Structures in `RandLinearAlgebra.jl`

Ingredient Structure:

- Denoted the name of the Technique
- Location where user specifies all their parameters

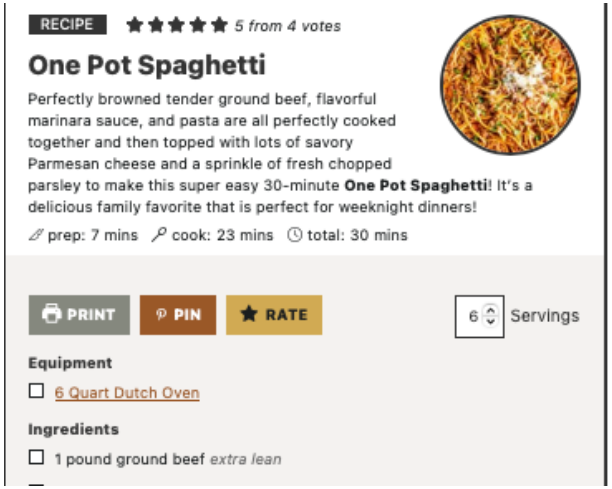
Key Structures in `RandLinearAlgebra.jl`

Ingredient Structure:

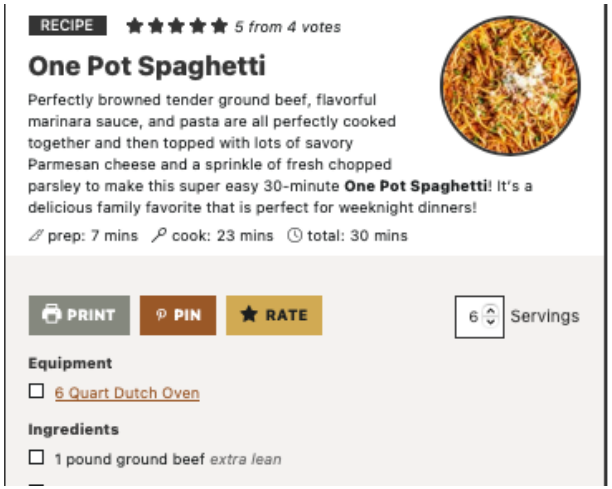
- Denoted the name of the Technique
- Location where user specifies all their parameters

Recipe Structure:

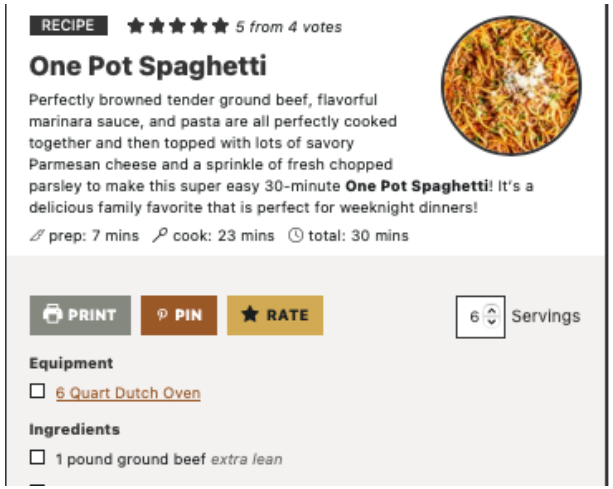
- Denoted with the [Ingredient]`Recipe`
- Contains all the information necessary for carrying out the technique



How to get a recipe

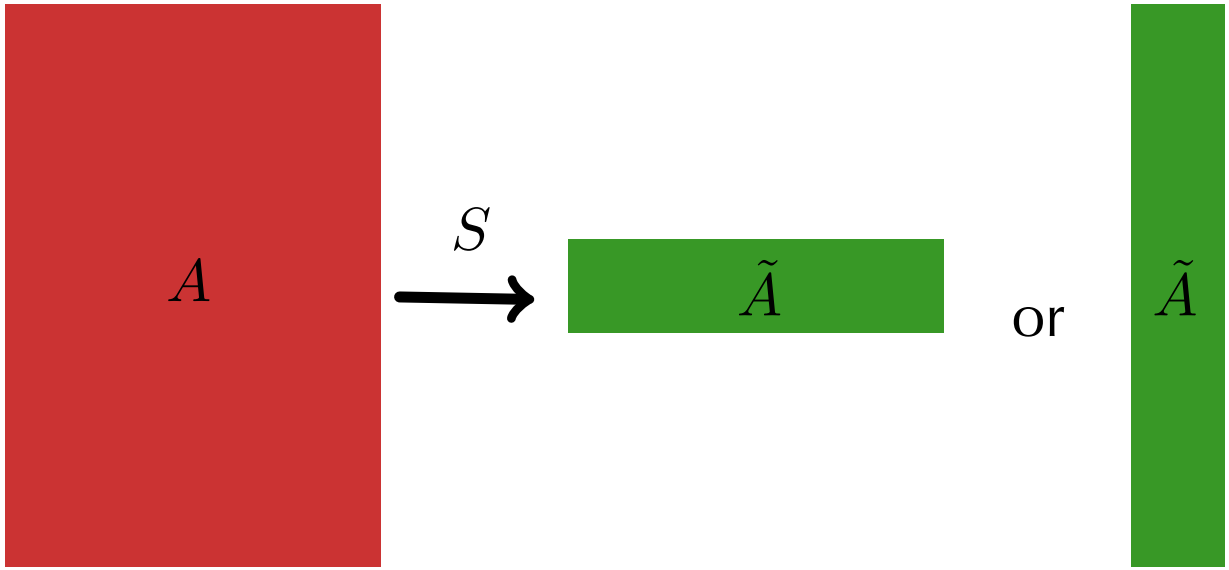


How to get a recipe



In `RandLinearAlgebra.jl` you just need to call the `complete_`[dataType]

Compressors



Left Compression
$\tilde{A} = SA \in \mathbb{R}^{p \times n}$

Right Compression
$\tilde{A} = AS \in \mathbb{R}^{m \times p}$

Types of Compressors

Sampling:

- S is a random subset of identity
- Random Subset chosen by sampling from a particular distribution
  - uniform distribution
  - leverage scores
  - determinantal point process
  - column/row norms

Sketching:

- Random linear combinations of rows/columns of the matrix
- Combination can arise from
  - Gaussian matrices
  - Randomized Trigormetric Transforms
  - Random Sparse matrices

Functions and Sub Structures

Functions:

- `complete_compressor`: turns your ingredients into a recipe
- `update_compressor!`: updates the randomness in your recipe

Sub Structures:

- `Cardinality`: Which side you primarily aim to apply the sketch
- `Distributions`: How you specify what you will sample indices from

Forming a Sketching Compressor

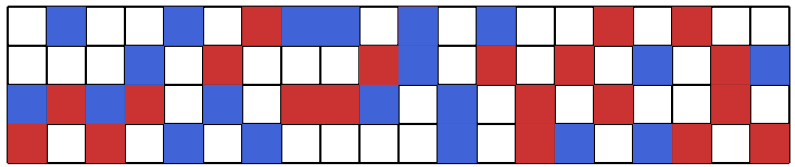
Decide:

- The method you want to use: **SparseSign**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**

Forming a Sketching Compressor

Decide:

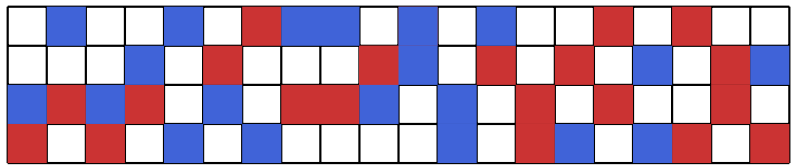
- The method you want to use: **SparseSign**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**



Forming a Sketching Compressor

Decide:

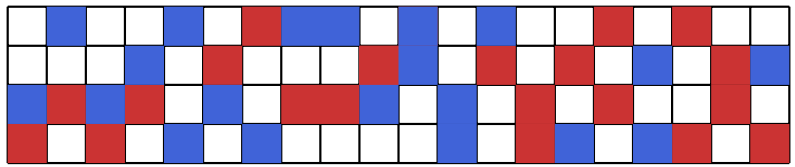
- The method you want to use: **SparseSign**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**



Forming a Sketching Compressor

Decide:

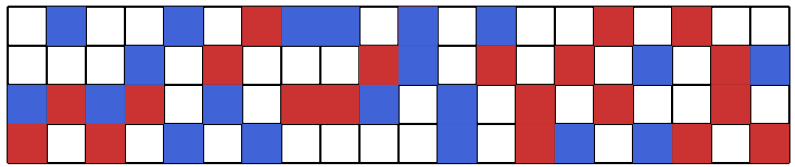
- The method you want to use: **SparseSign**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**



Forming a Sketching Compressor

Decide:

- The method you want to use: **SparseSign**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**



```
recipe = complete_compressor(
 SparseSign(
   compression_dim = 4,
   cardinality = Left()
 ),
 A
)
```

Forming a Sampling Compressor

- The method you want to use: **Strohmer Vershynin**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**

Forming a Sampling Compressor

- The method you want to use: **Strohmer Vershynin**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**

Forming a Sampling Compressor

- The method you want to use: **Strohmer Vershynin**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**

Forming a Sampling Compressor

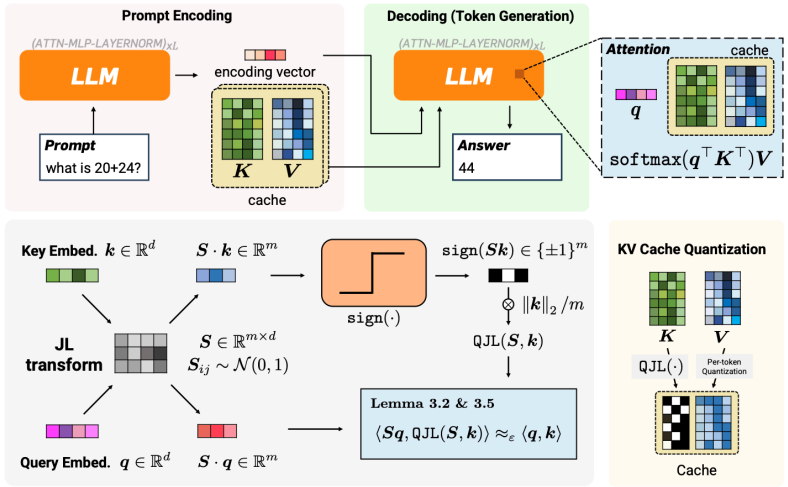
- The method you want to use: **Strohmer Vershynin**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**

Forming a Sampling Compressor

- The method you want to use: **Strohmer Vershynin**
- The size of compression dimension: **4**
- Whether we primarily want to apply from left or right: **Left**

```
recipe = complete_compressor(
 Sampling(
   compression_dim = 4,
   cardinality = Left(),
   distribution = L2Norm()
 ),
 A
)
```

Example Usage: Attention



- The size of the Key Value matrices makes deploying LLMs really expensive
- Google proposed using random compression plus quantization to reduce the size of these Key-Value matrices Zandieh et al. [2024]
- In our first example we show how to use compressors RandLinearAlgebra to speed up the matrix-matrix multiplication

Solvers

Want to find an x^* such that

$$Ax^* = b \tag{1}$$

or

$$x^* = \text{argmin}_x \|Ax - b\|_2 \tag{2}$$

Types of Solvers:

- **Consistent Systems:**
  - Kaczmarz
  - Randomized Krylov Methods
  - Nystrom/CUR preconditioners

- **Least Squares Solvers:**
  - Coordinate Descent
  - Iterative Hessian Sketch
  - Blendenpik, LSRN
  - Sketch and Solve
  - FOSSILs

Functions and Sub Structures

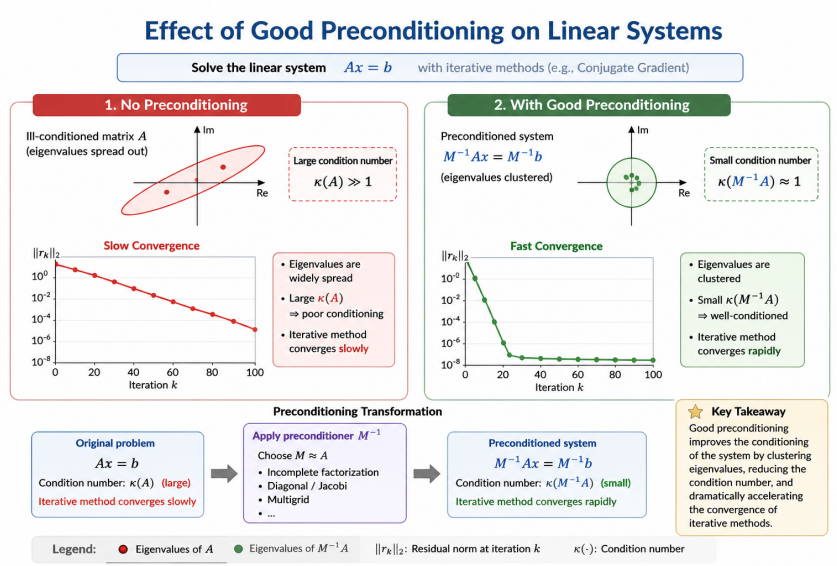
Functions:

- `complete_solver`: turns your ingredients into a recipe
- `rsolve(!)`: solves your linear system

Sub Structures:

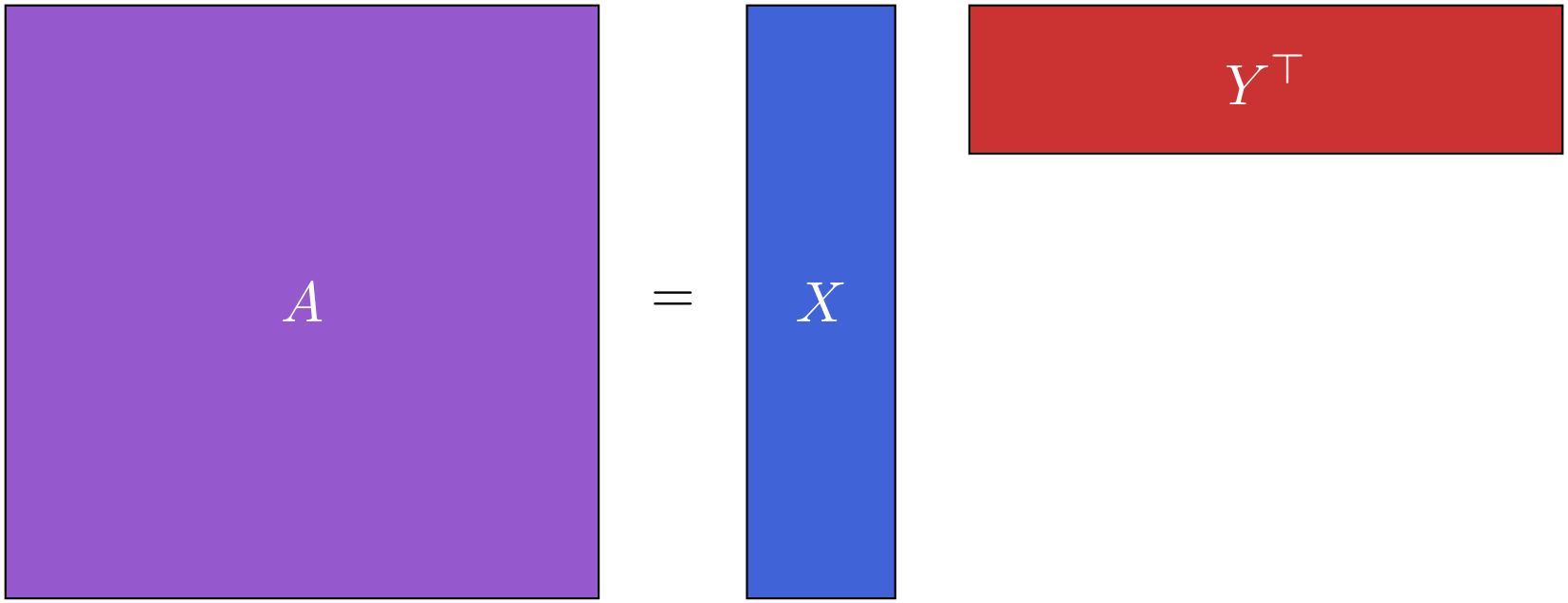
- `ErrorMethod`: Technique for measuring the accuracy of your solution
- `SubSolver`: How you will solve the compressed linear system (if necessary)
- `Logger`: How you log the progress of your solver and stop it when desired

Example Usage: Preconditioning CG



- We can use randomized solvers to precondition Krylov methods
- Now we show how we can incorporate the Kaczmarz solver to precondition CG

Approximators



Types of Approximators

Orthogonal Projection:

- Randomized RangeFinder (Q)
- RandSVD

Oblique Projection:

- Nystrom, Generalized Nystrom
- CUR, Interpolative Decomposition

Functions and Sub Structures

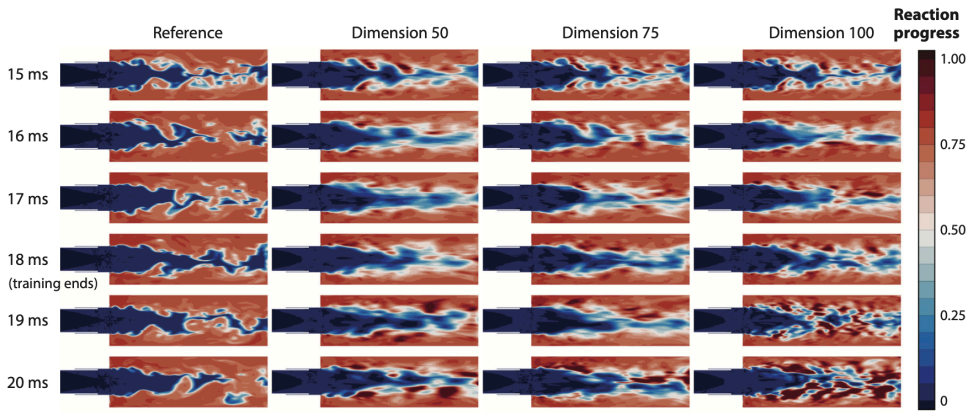
Functions:

- `complete_approximator`: turns your ingredients into a recipe
- `rapproximate(!)`: approximates your linear system

Sub Structures:

- `ErrorMethod`: Technique for measuring the accuracy of your approximation
- `Selector`: Technique for selecting indices in CUR/ID

Example Usage: Operator Inference



- Surrogate modeling approach that
  1. Projects a set of trajectories into a lower dimensional space
  2. Solves a least squares problem find the coefficients of a polynomial approximation of the trajectories Kramer et al. [2024]
- Consider Operator Inference applied to the one dimensional heat equations
- Will compare the subspace produced by `RandSVD` to that of the SVD

RandLinearAlgebra.jl makes prototyping easy

- The modular design makes it easy to test different variants of a RandNLA routine
- Julia makes it easy to test GPU performance and the use of different precisions
- Need relatively few lines of code to incorporate RandNLA routines into complex code

Check out the github!

Test out on Julia using:

```
]add RandLinearAlgebra.jl
```

or

```
using Pkg;
Pkg.add("RandLinearAlgebra")
```

Questions?

References I

Boris Kramer, Benjamin Peherstorfer, and Karen E. Willcox. Learning Nonlinear Reduced Models from Data with Operator Inference. *Annual Review of Fluid Mechanics*, 56(1):521–548, January 2024. ISSN 0066-4189, 1545-4479. doi: 10.1146/annurev-fluid-121021-025220. URL `https://www.annualreviews.org/doi/10.1146/annurev-fluid-121021-025220`.

Amir Zandieh, Majid Daliri, and Insu Han. QJL: 1-Bit Quantized JL Transform for KV Cache Quantization with Zero Overhead, July 2024.