Faster optimization using `RandomizedPreconditioners.jl`

Theo Diamandis

Work with Z. Frangella, B. Stellato, M. Udell, and S. Zhao

ICCOPT 2022

Many ways to speed up optimization

1. Speed up convergence (fewer iterations)
   - New algorithms
   - Better parameter selection

2. Speed up iterations of existing algorithms

3. ...

Many ways to speed up optimization

1. Speed up convergence (fewer iterations)
   - New algorithms
   - Better parameter selection

2. **Speed up iterations of existing algorithms** $\leftarrow$ this talk

3. ...

Linear system solves dominate solve time

- We consider quadratic programs (QPs) of the form

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = z \\ & l \leq z \leq u, \end{array}$$

Linear system solves dominate solve time

- We consider quadratic programs (QPs) of the form

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = z \\ & l \leq z \leq u, \end{array}$$

- ADMM (as in OSQP [Ste+20]) consists of the following steps:

$$x^{k+1} \leftarrow (P + \sigma I + \rho A^T A)^{-1}(\sigma x^k - q + A^T(\rho z^k - y^k))$$

$$z^{k+1} \leftarrow \Pi_{[l,u]}\left(Ax^{k+1} + \rho^{-1} y^k\right)$$

$$y^{k+1} \leftarrow y^k + \rho(Ax^{k+1} - z^{k+1})$$

Linear system solves dominate solve time

- We consider quadratic programs (QPs) of the form

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = z \\ & l \leq z \leq u, \end{array}$$

- ADMM (as in OSQP [Ste+20]) consists of the following steps:

$$\begin{aligned} x^{k+1} &\leftarrow (P + \sigma I + \rho A^T A)^{-1}(\sigma x^k - q + A^T(\rho z^k - y^k)) \\ z^{k+1} &\leftarrow \Pi_{[l,u]}\left(Ax^{k+1} + \rho^{-1} y^k\right) \\ y^{k+1} &\leftarrow y^k + \rho(Ax^{k+1} - z^{k+1}) \end{aligned}$$

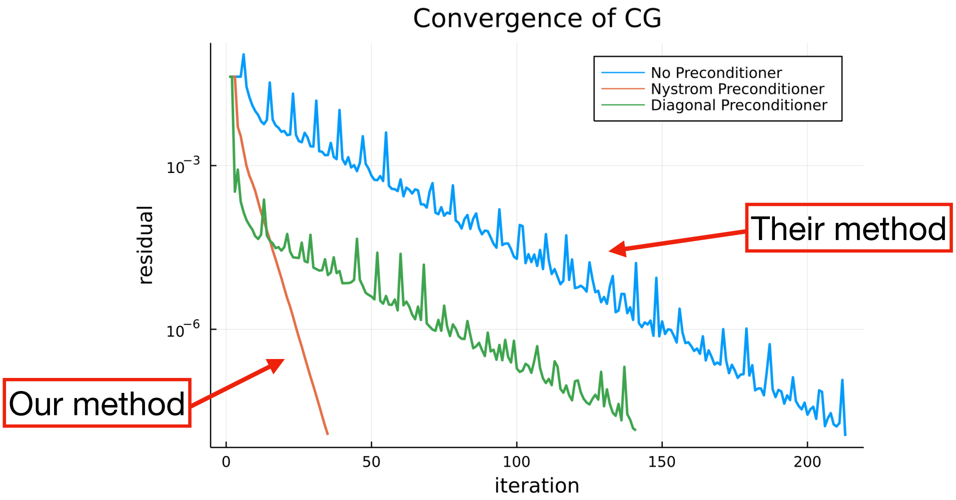
Idea: speed up bottleneck

Faster linear system solves

$$\Downarrow$$

Faster ADMM for QPs

Method: construct a good preconditioner



Outline

We want to quickly solve $Ax = b$

- We focus on large systems of the form

$$(A + \mu I)x = b$$

where $A \in \mathbb{S}_{+}^{n}$ and $\mu \geq 0$.

We want to quickly solve $Ax = b$

- We focus on large systems of the form

$$(A + \mu I)x = b$$

where $A \in \mathbb{S}_+^n$ and $\mu \geq 0$.

- "Large" means a direct solve is not computationally feasible.

We want to quickly solve $Ax = b$

- We focus on large systems of the form

$$(A + \mu I)x = b$$

  where $A \in \mathbb{S}^n_+$ and $\mu \geq 0$.

- "Large" means a direct solve is not computationally feasible.

- Ideas can be extended to other systems.

We use the conjugate gradient method (CG)

- CG only requires matrix vector products: $v \mapsto Av$
- CG converges quickly when
  1. The condition number of A is small
  2. The eigenvalues of A are clustered

We use the conjugate gradient method (CG)

- CG only requires matrix vector products: $v \mapsto Av$
- CG converges quickly when
  1. The condition number of A is small
  2. The eigenvalues of A are clustered

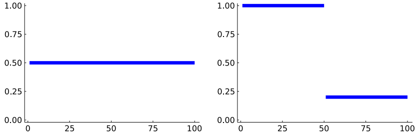


Figure: Easy for CG

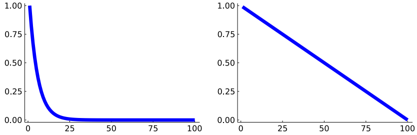


Figure: Hard for CG

A preconditioner can make the spectrum of A "nice"

Goal: Find a *preconditioner* P such that:

1. $v \mapsto P^{-1}v$ is easily evaluated
2. $P^{-1/2}(A + \mu I)P^{-1/2}$ has a "nice" spectrum for CG

Idea: figure out what you want, then approximate

We find the best possible preconditioner and instead of computing it exactly (slow), approximate it (fast).

We precondition using the dominant eigenspace

- Ideally, if we had access to the rank-k eigendecomposition $\lfloor A \rfloor_k = V_k \Lambda_k V_k^T$ and λ_{k+1} we would use

$$P = \frac{1}{\lambda_{k+1} + \mu} V_k(\Lambda_k + \mu I)V_k^T + I - V_k V_k^T.$$

- P admits an explicit cheap to apply inverse.

- Preconditioned system satisfies

$$\kappa_2(P^{-1/2} A_\mu P^{-1/2}) = \frac{\lambda_{k+1} + \mu}{\lambda_n + \mu}.$$

Approximate a decomposition via the Nystöm Sketch

- Computing exact partial eigendecompositions is expensive.
- The Nyström sketch gives an approximate eigendecomposition,

$$\hat{A}_{\mathrm{nys}} = (A\Omega)(\Omega^T A\Omega)^\dagger (A\Omega)^T = \hat{V}\hat{\Lambda}\hat{V}^T.$$

- $\Omega \in \mathbb{R}^{n\times k}$ is a random test matrix
  - A common choice is a standard normal Gaussian matrix.

The Nystöm Sketch comes from a best fit problem

- The Nyström sketch solves the optimization problem,

$$\hat{A}_{\text{nys}} = \underset{\text{range}(\hat{A}) \subset \text{range}(A\Omega)}{\text{argmin}} \|A - \hat{A}\|_F^2.$$

And sketching works well if the spectrum decays [FTU21]

- Approximation error depends on tail-eigenvalues[1]:

$$\mathbb{E}\|A - \hat{A}_r\| \leq 3\lambda_r + \frac{4e^2}{r}\sum_{j=r}^{n}\lambda_j$$

- System is well-conditioned in expectation (if r is large enough):

$$\mathbb{E}\left[\kappa\left(P^{-1/2}(A+\mu I)P^{-1/2}\right)\right] < 28.$$

[1] See [FTU21] for a more refined bound

Outline

Preconditioners can be constructed easily

- It only takes two lines of code!

```
using RandomizedPreconditioners
Anys = NystromSketch(A, k, r)
P = NystromPreconditioner(Anys, μ)
```

Preconditioners can be constructed easily

- It only takes two lines of code!

```
using RandomizedPreconditioners
Anys = NystromSketch(A, k, r)
P = NystromPreconditioner(Anys, μ)
```

- And we can get P^{-1} as well:

```
Pinv = NystromPreconditionerInverse(Anys, μ)
```

Implementation: `RandomizedPreconditioners.jl`

Preconditioners have efficient operations for solvers

- We use multiple dispatch to implement efficient
  - `ldiv!` for `P` and
  - `mul!` for `Pinv`

Preconditioners have efficient operations for solvers

- We use multiple dispatch to implement efficient
  - `ldiv!` for `P` and
  - `mul!` for `Pinv`
- These preconditioners can be easily passed to interative solvers:

```
using Krylov
x, stats = cg(A+μ*I, b; M=Pinv)
```

```
using IterativeSolvers
x, ch = cg(ATA, b; Pl = P, log=true)
```

Several sketches are included

- Positive semidefinite matrices: Nyström Sketch

```
Â = NystromSketch(A, k, r)
```

Several sketches are included

- Positive semidefinite matrices: Nyström Sketch

```
Â = NystromSketch(A, k, r)
```

- Symmetric matrices: Eigen Sketch

```
Â = EigenSketch(A, k, r)
```

Several sketches are included

- Positive semidefinite matrices: Nyström Sketch

```
Â = NystromSketch(A, k, r)
```

- Symmetric matrices: Eigen Sketch

```
Â = EigenSketch(A, k, r)
```

- General matrices: Randomized SVD

```
Â = RandomizedSVD(A, k, r; q=10)
```

Implementation: `RandomizedPreconditioners.jl`

These sketches come with several utilities including

- Fast multiplication:

```
Â = NystromSketch(A, k, r)
Â * v .== Â.U * Â.Λ * Â.U'* v

Â = RandomizedSVD(A, k, r)
Â * v .== Â.U * Â.Λ * Â.V' * v
```

These sketches come with several utilities including

- Fast multiplication:

```
Â = NystromSketch(A, k, r)
Â * v .== Â.U * Â.Λ * Â.U'* v

Â = RandomizedSVD(A, k, r)
Â * v .== Â.U * Â.Λ * Â.V' * v
```

- Adaptive sketch size selection:

```
#Doubles sketch size until ||Â - A|| is small
Â = adaptive_sketch(A, r0, EigenSketch)
```

Implementation: `RandomizedPreconditioners.jl`

And there's more...

- Eigenvalues of PSD matrices:

```
λmax_power =  RP.eigmax_power(A)
λmax_lanczos = RP.eigmax_lanczos(A)
λmin_lanczos = RP.eigmin_lanczos(A)
```

And there's more...

- Eigenvalues of PSD matrices:

```
λmax_power =  RP.eigmax_power(A)
λmax_lanczos = RP.eigmax_lanczos(A)
λmin_lanczos = RP.eigmin_lanczos(A)
```

- Different sketch matrices:

```
Ω = RP.GaussianTestMatrix(n, r)
Q = RP.rangefinder(A, r; Ω=Ω)

Ω = RP.SSFTTestMatrix(n, r)
Â = EigenSketch(A, k, r; Ω=Ω)
```

Check out the docs for `RandomizedPreconditioners.jl`

Outline

CG is faster on regression for small overhead

Ridge regression with $\sim 4.3k$ features (guillermo dataset [Van+13])

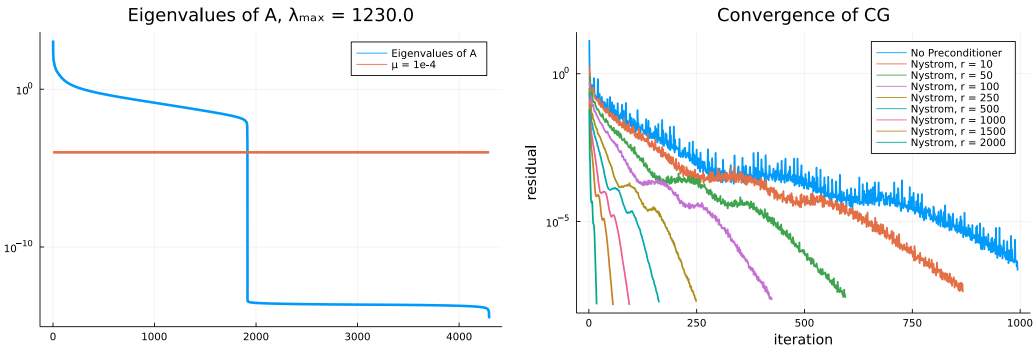


Figure: Spectrum (left) and convergence for various sketch sizes (right)

And it works on large examples too!

Ridge regression with 15k features, solved in <5s on a laptop

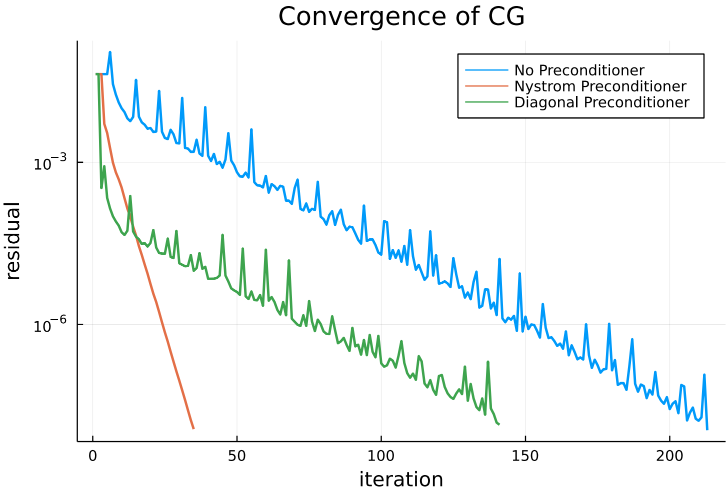


Figure: Nyström PCG vs Jacobi (diagonal) PCG vs vanilla CG

Where to go from here?

- **Package:** `RandomizedPreconditioners.jl`
  - Works with `LinearSolve.jl`
- **Theory:** Zach's paper on Nyström PCG [FTU21]
  - Also check out Martinsson & Tropp survey [MT21]
- **Extensions:** Reach out! (`tdiamand@mit.edu`)
  - Additional test matrices (esp. sparse matrix support)
  - Nonsymmetric systems (open research question!)

Outline

Preconditioning Linear Systems

Implementation: `RandomizedPreconditioners.jl`

Examples

Back to Optimization

Wrap Up

Add this to OSQP in linear system solver

- Recall, solving problems of the form

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Ax = z \\ & l \leq z \leq u, \end{array}$$

- Start by sketching linsys matrix & building preconditioner P_c
- Exploit structure to update P_c without re-sketching when parameters change
  - Requires recognizing structure in P and A

Example: bounded least squares

- We solve the problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|Ax - b\|_2^2 \\ \text{subject to} & 0 \le x \le 1 \end{array}$$

where A is $25,000 \times 15,000$.

Example: bounded least squares

- We solve the problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|Ax - b\|_2^2 \\ \text{subject to} & 0 \leq x \leq 1 \end{array}$$

  where A is $25,000 \times 15,000$.

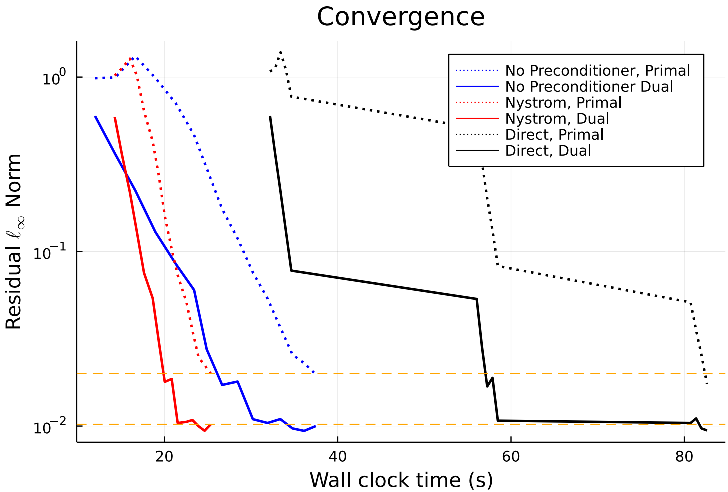
- Linear system matrix becomes $A^T A + (\sigma + \rho) I$
  - We sketch $A^T A$
  - We can easily update ρ, σ without re-sketching

Low overhead yields dramatic speedup

	Direct Solve	No Preconditioning	Nyström Preconditioning
Setup time (total)	31.693s	10.385s	13.687s
Preconditioning time	n/a	n/a	3.735s
Solve time	51.054s	27.313s	11.898s
Total time	82.747s	37.698s	29.320s

Table: ADMM runtime breakdown

Low overhead yields dramatic speedup



Outline

Randomized numerical linear algebra = powerful optimization primitives

- Solver iteration bottlenecks are often linear algebra operations

Randomized numerical linear algebra = powerful optimization primitives

- Solver iteration bottlenecks are often linear algebra operations
- Randomized techniques often give significant speed ups

Randomized numerical linear algebra = powerful optimization primitives

- Solver iteration bottlenecks are often linear algebra operations
- Randomized techniques often give significant speed ups
- Techniques usually incorporated easily & with low overhead for large problems

Randomized numerical linear algebra = powerful optimization primitives

- Solver iteration bottlenecks are often linear algebra operations
- Randomized techniques often give significant speed ups
- Techniques usually incorporated easily & with low overhead for large problems
- **Challenge:** parameter tuning for general-purpose solvers

Future Work

- `RandomizedPreconditioners.jl`
  - Adding additional test matrices
  - Providing better support for sparse matrices
  - Adding general preconditioners for nonsymmetric systems

Future Work

- `RandomizedPreconditioners.jl`
  - Adding additional test matrices
  - Providing better support for sparse matrices
  - Adding general preconditioners for nonsymmetric systems
- `NysOSQP.jl` (forthcoming)
  - `JuMP` interface (in progress)
  - Recognizing and exploiting structure in the linear system
  - Parameter tuning

References

Zachary Frangella, Joel A Tropp, and Madeleine Udell. "Randomized Nyström Preconditioning". In: *arXiv preprint arXiv:2110.02820* (2021).

PG Martinsson and JA Tropp. "Randomized numerical linear algebra: foundations & algorithms". In: *arXiv preprint arXiv:2002.01387* (2021).

B. Stellato et al. "OSQP: an operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2.

Joaquin Vanschoren et al. "OpenML: Networked Science in Machine Learning". In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60. DOI: 10.1145/2641190.2641198. URL: http://doi.acm.org/10.1145/2641190.2641198.

Thank you

- **Packages:**
  - `RandomizedPreconditioners.jl`
  - `NysOSQP.jl` (forthcoming)
- **Contact:** `tdiamand@mit.edu`