Randomized algorithms for least squares and matrix approximation

Tyler Chen

August 26, 2024

`chen.pw/slides`

About me

I work on the design and analysis of practically fast and theoretically justified randomized algorithms for fundamental linear algebra tasks.

I like working with nearby communities (physics, theoretical computer science, optimization, computational science, etc.).

Academic history:

- Currently an Assistant Professor / Courant Instructor at New York University
  - Sponsor: Chris Musco
- PhD in Applied Math at University of Washington
  - Advisors: Anne Greenbaum and Tom Trogdon
- B.S. in Math and Physics at Tufts University, minor in Studio Art

Today

Stochastic Optimization[1]

- First proof of $O(\sqrt{\kappa})$ convergence of minibatch stochastic gradient descent with heavy-ball momentum

Matrix approximation/operator learning[2]

- First matrix-vector product algorithm for near-optimal hierarchical matrix approximation
- Matrix-vector product algorithms for approximating fixed-sparsity matrices (no dimension dependence and matching lower bounds!)

[1]Bollapragada, Chen, and Ward 2024.

[2]Amsel, Chen, Keles, Halikias, Musco, and Musco 2024; Chen, Keles, Halikias, Musco, Musco, and Persson 2024.

Linear least squares

Consider a consistent least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}), \qquad f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 = \sum_{i=1}^{n} \frac{1}{2}(\mathbf{a}_i^\top\mathbf{x}-b_i)^2.$$

The gradient is

$$\nabla f(\mathbf{x}) = \mathbf{A}^\top(\mathbf{A}\mathbf{x}-\mathbf{b}) = \sum_{i=1}^{n} \mathbf{a}_i(\mathbf{a}_i^\top\mathbf{x}-b_i).$$

We can then implement gradient descent

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\nabla f(\mathbf{x}_k).$$

This requires one product with each $\mathbf{A}$ and $\mathbf{A}^\top$ every iteration.

Improvements to gradient descent

Stochastic gradients: Often we don't want to read the whole data matrix each iteration. Instead, we sample a random index i with probability $p_i \propto \|\mathbf{a}_i\|^2$, and then use the stochastic gradient

$$\nabla f_i(\mathbf{x}) = \frac{1}{p_i}\mathbf{a}_i(\mathbf{a}_i^\top \mathbf{x} - b_i).$$

The stochastic gradient is equal to the true gradient on average:

$$\mathbb{E}[\nabla f_i(\mathbf{x})] = \sum_{i=1}^{n} p_i \cdot \frac{1}{p_i}\mathbf{a}_i(\mathbf{a}_i^\top \mathbf{x} - b_i) = \sum_{i=1}^{n} \mathbf{a}_i(\mathbf{a}_i^\top \mathbf{x} - b_i) = \nabla f(\mathbf{x}).$$

Momentum: Instead of just updating based on the gradient, we can take previous iterates into account.

Convergence gurantees

Both momentum and stochastic gradient improve on classical gradient descent.

algorithm	iterations	cost/iter	formula
Gradient Descent	$\lambda_{\max}/\lambda_{\min}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f(\mathbf{x}_k)$
Momentum	$\sqrt{\lambda_{\max}/\lambda_{\min}}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1})$
Stochastic Gradient	$n \cdot (\lambda_{\text{ave}}/\lambda_{\min})$	d	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f_i(\mathbf{x}_k)$

However, the bounds are somewhat incomparable:

- average condition number vs. square root of condition number
- the cost per iteration is not necessarily indicative of real-world costs
- sampling assumptions

Convergence gurantees

Both momentum and stochastic gradient improve on classical gradient descent.

algorithm	iterations	cost/iter	formula
Gradient Descent	$\lambda_{\max}/\lambda_{\min}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)$
Momentum	$\sqrt{\lambda_{\max}/\lambda_{\min}}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1})$
Stochastic Gradient	$n \cdot (\lambda_{\text{ave}}/\lambda_{\min})$	d	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f_i(\mathbf{x}_k)$

However, the bounds are somewhat incomparable:

- average condition number vs. square root of condition number
- the cost per iteration is not necessarily indicative of real-world costs
- sampling assumptions

Convergence gurantees

Both momentum and stochastic gradient improve on classical gradient descent.

algorithm	iterations	cost/iter	formula
Gradient Descent	$\lambda_{\max}/\lambda_{\min}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)$
Momentum	$\sqrt{\lambda_{\max}/\lambda_{\min}}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1})$
Stochastic Gradient	$n \cdot (\lambda_{\text{ave}}/\lambda_{\min})$	d	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f_i(\mathbf{x}_k)$

However, the bounds are somewhat incomparable:

- average condition number vs. square root of condition number
- the cost per iteration is not necessarily indicative of real-world costs
- sampling assumptions

Convergence gurantees

Both momentum and stochastic gradient improve on classical gradient descent.

algorithm	iterations	cost/iter	formula
Gradient Descent	$\lambda_{\max}/\lambda_{\min}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f(\mathbf{x}_k)$
Momentum	$\sqrt{\lambda_{\max}/\lambda_{\min}}$	nd	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1})$
Stochastic Gradient	$n \cdot (\lambda_{\text{ave}}/\lambda_{\min})$	d	$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla f_i(\mathbf{x}_k)$

However, the bounds are somewhat incomparable:

- average condition number vs. square root of condition number
- the cost per iteration is not necessarily indicative of real-world costs
- sampling assumptions

Key question:

What is the role of momentum in SGD?

Minibatch gradient descent

In practice: momentum is used with stochastic methods and plays a critical role in training deep learning models!

In theory: momentum cannot be used to accelerate SGD.[3]

Minibatching: Instead of sampling a single index, we can sample several B indices at once and put them into a list S. This gives the minibatch stochstic gradient

$$\nabla f_S(\mathbf{x}) = \frac{1}{B} \sum_{i \in S} \frac{1}{p_i} \mathbf{a}_i(\mathbf{a}_i^\top \mathbf{x} - b_i).$$

One minibatch evaluation costs $O(Bd)$ operations.[4]

[3]Jain, Kakade, Kidambi, Netrapalli, and Sidford 2018.
[4]Due to parallelism, the real-world cost is often sort of independent of B when B is reasonably small.

Minibatch gradient descent

In practice: momentum is used with stochastic methods and plays a critical role in training deep learning models!

In theory: momentum cannot be used to accelerate SGD.[3]

Minibatching: Instead of sampling a single index, we can sample several B indices at once and put them into a list S. This gives the minibatch stochstic gradient

$$\nabla f_S(\mathbf{x}) = \frac{1}{B} \sum_{i \in S} \frac{1}{p_i} \mathbf{a}_i (\mathbf{a}_i^\top \mathbf{x} - b_i).$$

One minibatch evaluation costs $O(Bd)$ operations.[4]

[3]Jain, Kakade, Kidambi, Netrapalli, and Sidford 2018.
[4]Due to parallelism, the real-world cost is often sort of independent of B when B is reasonably small.

Minibatching+momentum

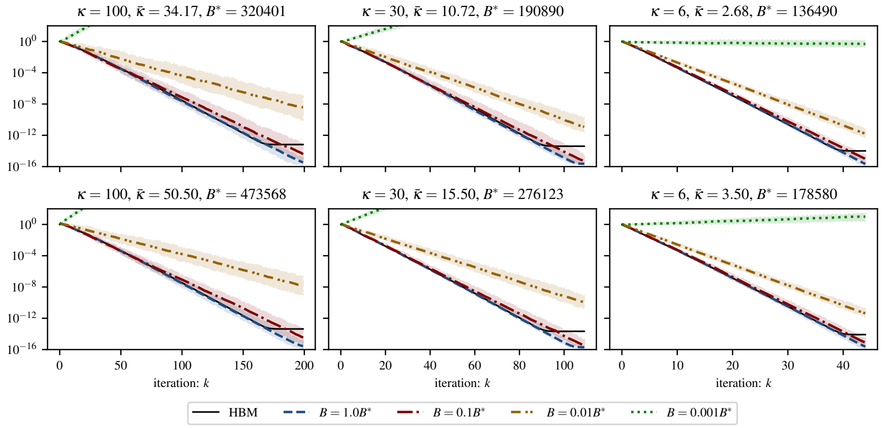
Theorem (Bollapragada, Chen, and Ward 2024). Minibatch heavy ball momentum converges in $\sqrt{\lambda_{\max}/\lambda_{\min}} \cdot \log(1/\varepsilon)$ iterations with minibatch size $B = O(d\log(d) \cdot (\lambda_{\text{ave}}/\lambda_{\min}) \cdot \sqrt{\lambda_{\max}/\lambda_{\min}})$.

Minibatch SGD can get the best of both worlds!

- same rate of convergence as full gradient methods
- cheaper gradients computations when n is large

Numerical example

Our theory might be precise enough to predict practical performance.



Proof overview (Polyak HBM)

Let's first look at how to analyze the classical heavy ball momentum algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \underbrace{(\mathbf{A}^\top \mathbf{A}\mathbf{x}_k - \mathbf{A}^\top \mathbf{b})}_{\text{gradient}} + \beta \underbrace{(\mathbf{x}_k - \mathbf{x}_{k-1})}_{\text{momentum}}.$$

Assuming a consistent system, $\mathbf{b} = \mathbf{A}\mathbf{x}^*$, so we can rewrite the update as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{A}^\top \mathbf{A}(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

Therefore,

$$\begin{aligned} \mathbf{x}_{k+1} - \mathbf{x}^* &= (\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}^* + \mathbf{x}^* - \mathbf{x}_{k-1}) \\ &= ((1+\beta)\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) - \beta(\mathbf{x}_k - \mathbf{x}_{k-1}). \end{aligned}$$

We can write this as a matrix iteration

$$\begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A} & -\beta \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}}_{\mathbf{T} = \mathbf{T}(\alpha,\beta)} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^* \end{bmatrix} = \mathbf{T}^k \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix}$$

Proof overview (Polyak HBM)

Let's first look at how to analyze the classical heavy ball momentum algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \underbrace{(\mathbf{A}^\top \mathbf{A}\mathbf{x}_k - \mathbf{A}^\top \mathbf{b})}_{\text{gradient}} + \beta \underbrace{(\mathbf{x}_k - \mathbf{x}_{k-1})}_{\text{momentum}}.$$

Assuming a consistent system, $\mathbf{b} = \mathbf{A}\mathbf{x}^*$, so we can rewrite the update as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{A}^\top \mathbf{A}(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

Therefore,

$$\begin{aligned}\mathbf{x}_{k+1} - \mathbf{x}^* &= (\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}^* + \mathbf{x}^* - \mathbf{x}_{k-1}) \\ &= ((1+\beta)\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) - \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).\end{aligned}$$

We can write this as a matrix iteration

$$\begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A} & -\beta \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}}_{\mathbf{T} = \mathbf{T}(\alpha,\beta)} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^* \end{bmatrix} = \mathbf{T}^k \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix}$$

Proof overview (Polyak HBM)

Let's first look at how to analyze the classical heavy ball momentum algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \underbrace{(\mathbf{A}^\top \mathbf{A} \mathbf{x}_k - \mathbf{A}^\top \mathbf{b})}_{\text{gradient}} + \beta \underbrace{(\mathbf{x}_k - \mathbf{x}_{k-1})}_{\text{momentum}}.$$

Assuming a consistent system, $\mathbf{b} = \mathbf{A}\mathbf{x}^*$, so we can rewrite the update as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{A}^\top \mathbf{A}(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

Therefore,

$$\begin{aligned} \mathbf{x}_{k+1} - \mathbf{x}^* &= (\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}^* + \mathbf{x}^* - \mathbf{x}_{k-1}) \\ &= ((1+\beta)\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) - \beta(\mathbf{x}_k - \mathbf{x}_{k-1}). \end{aligned}$$

We can write this as a matrix iteration

$$\begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha \mathbf{A}^\top \mathbf{A} & -\beta \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}}_{\mathbf{T} = \mathbf{T}(\alpha,\beta)} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^* \end{bmatrix} = \mathbf{T}^k \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix}$$

Proof overview (Polyak HBM)

Let's first look at how to analyze the classical heavy ball momentum algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \underbrace{(\mathbf{A}^\top\mathbf{A}\mathbf{x}_k - \mathbf{A}^\top\mathbf{b})}_{\text{gradient}} + \beta \underbrace{(\mathbf{x}_k - \mathbf{x}_{k-1})}_{\text{momentum}}.$$

Assuming a consistent system, $\mathbf{b} = \mathbf{A}\mathbf{x}^*$, so we can rewrite the update as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{A}^\top\mathbf{A}(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

Therefore,

$$\begin{aligned}\mathbf{x}_{k+1} - \mathbf{x}^* &= (\mathbf{I} - \alpha\mathbf{A}^\top\mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}^* + \mathbf{x}^* - \mathbf{x}_{k-1}) \\ &= ((1+\beta)\mathbf{I} - \alpha\mathbf{A}^\top\mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) - \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).\end{aligned}$$

We can write this as a matrix iteration

$$\begin{bmatrix}\mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^*\end{bmatrix} = \underbrace{\begin{bmatrix}(1+\beta)\mathbf{I} - \alpha\mathbf{A}^\top\mathbf{A} & -\beta\mathbf{I} \\ \mathbf{I} & \mathbf{0}\end{bmatrix}}_{\mathbf{T}=\mathbf{T}(\alpha,\beta)} \begin{bmatrix}\mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^*\end{bmatrix} = \mathbf{T}^k \begin{bmatrix}\mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^*\end{bmatrix}$$

Proof overview (Polyak HBM)

Let's first look at how to analyze the classical heavy ball momentum algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \underbrace{(\mathbf{A}^\top\mathbf{A}\mathbf{x}_k - \mathbf{A}^\top\mathbf{b})}_{\text{gradient}} + \beta \underbrace{(\mathbf{x}_k - \mathbf{x}_{k-1})}_{\text{momentum}}.$$

Assuming a consistent system, $\mathbf{b} = \mathbf{A}\mathbf{x}^*$, so we can rewrite the update as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{A}^\top\mathbf{A}(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

Therefore,

$$\begin{aligned}\mathbf{x}_{k+1} - \mathbf{x}^* &= (\mathbf{I} - \alpha\mathbf{A}^\top\mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) + \beta(\mathbf{x}_k - \mathbf{x}^* + \mathbf{x}^* - \mathbf{x}_{k-1}) \\ &= ((1+\beta)\mathbf{I} - \alpha\mathbf{A}^\top\mathbf{A})(\mathbf{x}_k - \mathbf{x}^*) - \beta(\mathbf{x}_k - \mathbf{x}_{k-1}).\end{aligned}$$

We can write this as a matrix iteration

$$\begin{bmatrix}\mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^*\end{bmatrix} = \underbrace{\begin{bmatrix}(1+\beta)\mathbf{I} - \alpha\mathbf{A}^\top\mathbf{A} & -\beta\mathbf{I} \\ \mathbf{I} & \mathbf{0}\end{bmatrix}}_{\mathbf{T}=\mathbf{T}(\alpha,\beta)}\begin{bmatrix}\mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^*\end{bmatrix} = \mathbf{T}^k\begin{bmatrix}\mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^*\end{bmatrix}$$

Proof overview (Polyak HBM)

Now, bounding the error,

$$\left\| \begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} \right\| \leq \|\mathbf{T}^k\| \left\| \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix} \right\|.$$

It's straightforward (but a bit tedious) to show that the eigenvalues of $\mathbf{T}$ are

$$z_j^{\pm} := \frac{1}{2}\left(1 + \beta - \alpha\lambda_j \pm \sqrt{(1 + \beta - \alpha\lambda_j)^2 - 4\beta}\right).$$

Finally, optimizing the choices of α and β, we can get a bound

$$\|\mathbf{T}^k\| \approx \|\mathbf{T}\|^k = \left(\frac{\sqrt{\lambda_{\max}/\lambda_{\min}} - 1}{\sqrt{\lambda_{\max}/\lambda_{\min}} + 1}\right)^k \leq \exp\left(-\frac{k}{2\sqrt{\lambda_{\max}/\lambda_{\min}}}\right).$$

Proof overview (Polyak HBM)

Now, bounding the error,

$$\left\| \begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} \right\| \leq \|\mathbf{T}^k\| \left\| \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix} \right\|.$$

It's straightforward (but a bit tedious) to show that the eigenvalues of $\mathbf{T}$ are

$$z_j^{\pm} := \frac{1}{2}\left(1 + \beta - \alpha\lambda_j \pm \sqrt{(1 + \beta - \alpha\lambda_j)^2 - 4\beta}\right).$$

Finally, optimizing the choices of α and β, we can get a bound

$$\|\mathbf{T}^k\| \approx \|\mathbf{T}\|^k = \left(\frac{\sqrt{\lambda_{\max}/\lambda_{\min}} - 1}{\sqrt{\lambda_{\max}/\lambda_{\min}} + 1}\right)^k \leq \exp\left(-\frac{k}{2\sqrt{\lambda_{\max}/\lambda_{\min}}}\right).$$

Proof overview (Polyak HBM)

Now, bounding the error,

$$\left\| \begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} \right\| \leq \|\mathbf{T}^k\| \left\| \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix} \right\|.$$

It's straightforward (but a bit tedious) to show that the eigenvalues of $\mathbf{T}$ are

$$z_j^{\pm} := \frac{1}{2}\left(1 + \beta - \alpha\lambda_j \pm \sqrt{(1+\beta-\alpha\lambda_j)^2 - 4\beta}\right).$$

Finally, optimizing the choices of α and β, we can get a bound

$$\|\mathbf{T}^k\| \approx \|\mathbf{T}\|^k = \left(\frac{\sqrt{\lambda_{\max}/\lambda_{\min}} - 1}{\sqrt{\lambda_{\max}/\lambda_{\min}} + 1}\right)^k \leq \exp\left(-\frac{k}{2\sqrt{\lambda_{\max}/\lambda_{\min}}}\right).$$

Proof overview

In minibatch-HBM, we replace the true gradient with a minibatch stochastic gradient

$$\nabla f_{S_k}(\mathbf{x}_k) = \frac{1}{B}\sum_{j\in S_k}\frac{1}{p_j}\mathbf{a}_j\mathbf{a}_j^\top(\mathbf{x}_k - \mathbf{x}^*).$$

Define the random matrix

$$\mathbf{M}_{S_k} = \frac{1}{B}\sum_{j\in S_k}\frac{1}{p_j}\mathbf{a}_j\mathbf{a}_j^\top$$

and observe that

$$\mathbb{E}[\mathbf{M}_{S_k}] = \frac{1}{B}\mathbb{E}\left[\sum_{j\in S_k}\frac{1}{p_j}\mathbf{a}_j\mathbf{a}_j^\top\right] = \mathbf{A}^\top\mathbf{A}.$$

Proof overview

A similar argument gives us a recurrence

$$\begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha \mathbf{M}_{S_k} & -\beta \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}}_{\mathbf{Y}_{S_k} = \mathbf{Y}_{S_k}(\alpha,\beta)} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^* \end{bmatrix}.$$

This gives an error bound

$$\left\| \begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} \right\| \le \|\mathbf{Y}_{S_k} \mathbf{Y}_{S_{k-1}} \cdots \mathbf{Y}_{S_1}\| \left\| \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix} \right\|.$$

We might hope that

$$\|\mathbf{Y}_{S_k} \mathbf{Y}_{S_{k-1}} \cdots \mathbf{Y}_{S_1}\| \approx \|\mathbb{E}[\mathbf{Y}_{S_k}] \mathbb{E}[\mathbf{Y}_{S_{k-1}}] \cdots \mathbb{E}[\mathbf{Y}_{S_1}]\| = \|\mathbf{T}^k\|.$$

Proof overview

A similar argument gives us a recurrence

$$\begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha \mathbf{M}_{S_k} & -\beta \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}}_{\mathbf{Y}_{S_k} = \mathbf{Y}_{S_k}(\alpha,\beta)} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^* \end{bmatrix}.$$

This gives an error bound

$$\left\| \begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} \right\| \leq \|\mathbf{Y}_{S_k} \mathbf{Y}_{S_{k-1}} \cdots \mathbf{Y}_{S_1}\| \left\| \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix} \right\|.$$

We might hope that

$$\|\mathbf{Y}_{S_k} \mathbf{Y}_{S_{k-1}} \cdots \mathbf{Y}_{S_1}\| \approx \|\mathbb{E}[\mathbf{Y}_{S_k}]\mathbb{E}[\mathbf{Y}_{S_{k-1}}] \cdots \mathbb{E}[\mathbf{Y}_{S_1}]\| = \|\mathbf{T}^k\|.$$

Proof overview

A similar argument gives us a recurrence

$$\begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} = \underbrace{\begin{bmatrix} (1+\beta)\mathbf{I} - \alpha \mathbf{M}_{S_k} & -\beta \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}}_{\mathbf{Y}_{S_k} = \mathbf{Y}_{S_k}(\alpha,\beta)} \begin{bmatrix} \mathbf{x}_k - \mathbf{x}^* \\ \mathbf{x}_{k-1} - \mathbf{x}^* \end{bmatrix}.$$

This gives an error bound

$$\left\| \begin{bmatrix} \mathbf{x}_{k+1} - \mathbf{x}^* \\ \mathbf{x}_k - \mathbf{x}^* \end{bmatrix} \right\| \le \|\mathbf{Y}_{S_k} \mathbf{Y}_{S_{k-1}} \cdots \mathbf{Y}_{S_1}\| \left\| \begin{bmatrix} \mathbf{x}_1 - \mathbf{x}^* \\ \mathbf{x}_0 - \mathbf{x}^* \end{bmatrix} \right\|.$$

We might hope that

$$\|\mathbf{Y}_{S_k} \mathbf{Y}_{S_{k-1}} \cdots \mathbf{Y}_{S_1}\| \approx \|\mathbb{E}[\mathbf{Y}_{S_k}] \mathbb{E}[\mathbf{Y}_{S_{k-1}}] \cdots \mathbb{E}[\mathbf{Y}_{S_1}]\| = \|\mathbf{T}^k\|.$$

Products of random matrices

Theorem (Huang, Niles-Weed, Tropp, and Ward 2021)**.** Consider an independent sequence of $d \times d$ random matrices $\mathbf{X}_1, \dots, \mathbf{X}_k$, and form the product

$$\mathbf{Z} = \mathbf{X}_k \mathbf{X}_{k-1} \cdots \mathbf{X}_1.$$

Assume $\|\mathbb{E}[\mathbf{X}_i]\| \leq q_i$ and $\mathbb{E}[\|\mathbf{X}_i - \mathbb{E}\mathbf{X}_i\|^2]^{1/2} \leq \sigma_i q_i$ for $i = 1, \dots, k$. Let $\mathrm{Q} = \prod_{i=1}^n q_i$ and $v = \sum_{i=1}^k \sigma_i^2$. Then

$$\mathbb{E}\big[\,\|\mathbf{Z}\|\,\big] \leq \mathrm{Q} \exp\left(\sqrt{2v \max\{2v, \log(d)\}}\right).$$

To determine q_i and σ_i we we use matrix concentration bounds for sums of matrices[5].

[5]Tropp 2015.

Outlook

Lots of interesting potential follow up work:

- adaptive/automatic parameter selection
- beyond least squares

Key question:

What can we learn about matrices from matvecs?

The matrix-vector product query model

In this model of computation, we are only given access to $\mathbf{A}$ via a black box that lets us perform matrix-vector products $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^\top\mathbf{x}$.

Examples:

- $\mathbf{A} = \mathbf{B}^{-1}$, and we only know $\mathbf{B}$
  - perform products by solving $\mathbf{B}\mathbf{y} = \mathbf{x}$
  - this generalizes to $\mathbf{A} = f(\mathbf{B})$
- $\mathbf{A} = \exp(-it\mathbf{H})$
  - perform products by evolving according to Hamiltonian $\mathbf{H}$ for time t from initial state $\mathbf{x}$
- $\mathbf{A}$ is the system matrix for a CT scanner
  - perform products by running the CT scanner on target $\mathbf{x}$
- $\mathbf{A}$ is a regular matrix
  - perform products with highly optimized hardware/software (e.g. GPUs, crossbar array, MKL, etc.)

The matrix-vector product query model

In this model of computation, we are only given access to $\mathbf{A}$ via a black box that lets us perform matrix-vector products $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^\mathsf{T}\mathbf{x}$.

Examples:

- $\mathbf{A} = \mathbf{B}^{-1}$, and we only know $\mathbf{B}$
  - perform products by solving $\mathbf{B}\mathbf{y} = \mathbf{x}$
  - this generalizes to $\mathbf{A} = f(\mathbf{B})$
- $\mathbf{A} = \exp(-it\mathbf{H})$
  - perform products by evolving according to Hamiltonian $\mathbf{H}$ for time t from initial state $\mathbf{x}$
- $\mathbf{A}$ is the system matrix for a CT scanner
  - perform products by running the CT scanner on target $\mathbf{x}$
- $\mathbf{A}$ is a regular matrix
  - perform products with highly optimized hardware/software (e.g. GPUs, crossbar array, MKL, etc.)

The matrix-vector product query model

In this model of computation, we are only given access to $\mathbf{A}$ via a black box that lets us perform matrix-vector products $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^\top\mathbf{x}$.

Examples:

- $\mathbf{A} = \mathbf{B}^{-1}$, and we only know $\mathbf{B}$
  - perform products by solving $\mathbf{B}\mathbf{y} = \mathbf{x}$
  - this generalizes to $\mathbf{A} = f(\mathbf{B})$
- $\mathbf{A} = \exp(-\mathrm{i}t\mathbf{H})$
  - perform products by evolving according to Hamiltonian $\mathbf{H}$ for time t from initial state $\mathbf{x}$
- $\mathbf{A}$ is the system matrix for a CT scanner
  - perform products by running the CT scanner on target $\mathbf{x}$
- $\mathbf{A}$ is a regular matrix
  - perform products with highly optimized hardware/software (e.g. GPUs, crossbar array, MKL, etc.)

The matrix-vector product query model

In this model of computation, we are only given access to $\mathbf{A}$ via a black box that lets us perform matrix-vector products $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^\top\mathbf{x}$.

Examples:

- $\mathbf{A} = \mathbf{B}^{-1}$, and we only know $\mathbf{B}$
  - perform products by solving $\mathbf{B}\mathbf{y} = \mathbf{x}$
  - this generalizes to $\mathbf{A} = f(\mathbf{B})$
- $\mathbf{A} = \exp(-\mathrm{i}t\mathbf{H})$
  - perform products by evolving according to Hamiltonian $\mathbf{H}$ for time t from initial state $\mathbf{x}$
- $\mathbf{A}$ is the system matrix for a CT scanner
  - perform products by running the CT scanner on target $\mathbf{x}$
- $\mathbf{A}$ is a regular matrix
  - perform products with highly optimized hardware/software (e.g. GPUs, crossbar array, MKL, etc.)

The matrix-vector product query model

In this model of computation, we are only given access to $\mathbf{A}$ via a black box that lets us perform matrix-vector products $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ and $\mathbf{x} \mapsto \mathbf{A}^\mathsf{T}\mathbf{x}$.

Examples:

- $\mathbf{A} = \mathbf{B}^{-1}$, and we only know $\mathbf{B}$
  - perform products by solving $\mathbf{B}\mathbf{y} = \mathbf{x}$
  - this generalizes to $\mathbf{A} = f(\mathbf{B})$
- $\mathbf{A} = \exp(-\mathrm{i}t\mathbf{H})$
  - perform products by evolving according to Hamiltonian $\mathbf{H}$ for time t from initial state $\mathbf{x}$
- $\mathbf{A}$ is the system matrix for a CT scanner
  - perform products by running the CT scanner on target $\mathbf{x}$
- $\mathbf{A}$ is a regular matrix
  - perform products with highly optimized hardware/software (e.g. GPUs, crossbar array, MKL, etc.)

The computational complexity of linear algebra

Numerical linear algebra is one of the oldest fields of computing.

- Traditionally, the cost of algorithms is studied in terms of floating point operations, but flops are increasingly less relevant.
- Now, things like matrix products (or matrix loads) often dominate the cost of an algorithm.

If we take matrix-products as the basic unit of computation, we can also study the computational complexity of linear algebra problems.

- almost all linear algebra problems can be solved in $O(n^3)$ time, so classical complexity theory (e.g. P vs NP) is not very useful

Structrured matrices

Often, we can solved linear algebra problems faster when the matrices of interest are structured.

Examples:

- low-rank matrices
- hierarchical matrices
- sparse matrices
- banded matrices

Common framework: Approximate $\mathbf{A}$ with a structured matrix, then use the structured approximation.

Matrix approximation

Problem. Let S be some family of matrices and $\mathbf{A}$ an arbitrary matrix that can only be accessed by matrix-vector products.

Find a matrix $\widetilde{\mathbf{A}} \in S$ such that

$$\|\mathbf{A} - \widetilde{\mathbf{A}}\| \leq (1+\varepsilon) \min_{\mathbf{X} \in S} \|\mathbf{A} - \mathbf{X}\|.$$

This problem asks us to find a structured matrix approximation $\widetilde{\mathbf{A}}$ to $\mathbf{A}$ competitive with the best possible approximation.

- If $\mathbf{A} \in S$, then we require $\widetilde{\mathbf{A}} = \mathbf{A}$
- If $\mathbf{A}$ has a good approximation from S, then it might be okay for ε to be large

Wxample: low-rank approximation[6]

[6] Halko, Martinsson, and Tropp 2011; Musco and Musco 2015; Tropp and Webber 2023.

Hierarchical matrix motivation: kernel matrices

Consider data $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$ and a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$.

- examples: $k(\mathbf{x}, \mathbf{y}) = 1/\|\mathbf{x} - \mathbf{y}\|^2$, $k(, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/\sigma^2)$, $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + c)^q$

We can define a kernel matrix

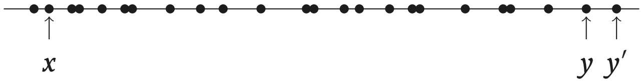
$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}.$$

Kernel matrices are widely used in data science and machine learning applications for task such as clustering and classification.[7]

[7] **?**.

Hierarchical matrix motivation: kernel matrices

For simplicity, consider 1 dimensional data:

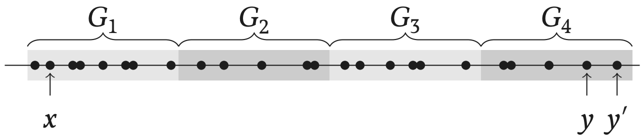


Observations:

- Interactions between points in non-adjacent groups are approximately low-rank.
- We can recursively treat interactions between adjacent groups or within a group at a finer scale.

Hierarchical matrix motivation: kernel matrices

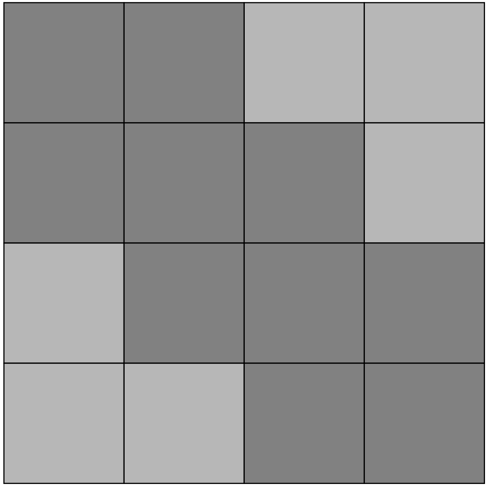
For simplicity, consider 1 dimensional data:



Observations:

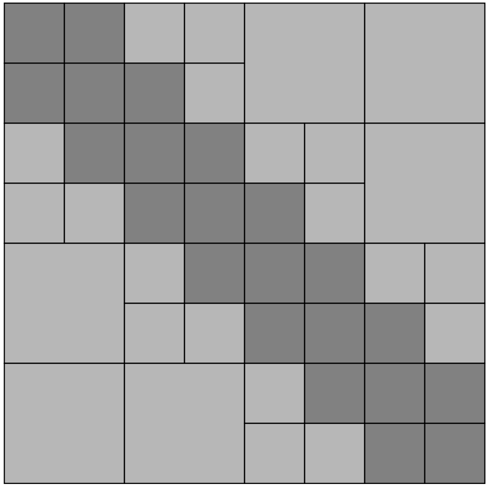
- Interactions between points in non-adjacent groups are approximately low-rank.
- We can recursively treat interactions between adjacent groups or within a group at a finer scale.

Hierarchical matrices



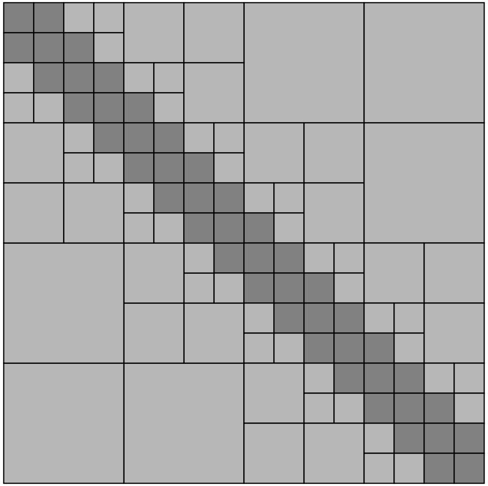
▢ low-rank block ▨ recursive block

Hierarchical matrices



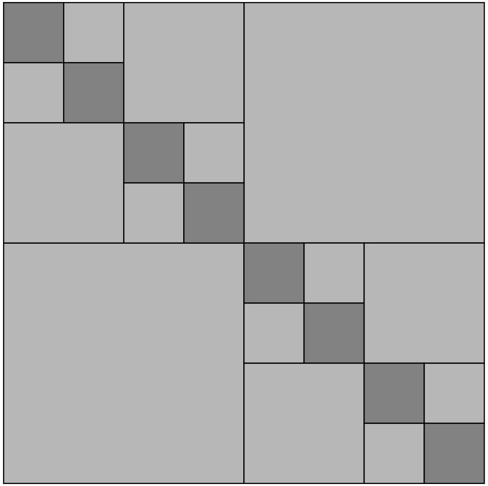
⬜ low-rank block ⬛ recursive block

Hierarchical matrices



low-rank block
recursive block

HODLR matrices



▢ low-rank block ▩ recursive block

The HODLR approximation problem

Problem. Given an $n \times n$ matrix $\mathbf{A}$, accessible only by matrix-vector products, a rank parameter k, and an accuracy parameter ε, find a HODLR(k) matrix $\widetilde{\mathbf{A}}$ such that

$$\mathbb{E}[\|\mathbf{A} - \widetilde{\mathbf{A}}\|_{\mathsf{F}}] \le (1+\varepsilon) \min_{\mathbf{H} \in \mathrm{HODLR}(k)} \|\mathbf{A} - \mathbf{H}\|_{\mathsf{F}}.$$

The best HODLR approximation to $\mathbf{A}$ is obtained by applying a rank-k SVD to each low-rank block of $\mathbf{A}$.

- This is too expensive in the matrix-vector product model (n products)

In the special case that $\mathbf{A} \in \mathrm{HODLR}(k)$, then we require $\widetilde{\mathbf{A}} = \mathbf{A}$ (regardless of ε).

- There are several matvec algorithms for this setting[8]

[8] Lin, Lu, and Ying 2011; Martinsson 2016; Levitt and Martinsson 2022; Halikias and Townsend 2023.

The HODLR approximation problem

Problem. Given an $n \times n$ matrix $\mathbf{A}$, accessible only by matrix-vector products, a rank parameter k, and an accuracy parameter ε, find a HODLR(k) matrix $\widetilde{\mathbf{A}}$ such that

$$\mathbb{E}[\|\mathbf{A} - \widetilde{\mathbf{A}}\|_\mathsf{F}] \le (1+\varepsilon) \min_{\mathbf{H} \in \mathrm{HODLR}(k)} \|\mathbf{A} - \mathbf{H}\|_\mathsf{F}.$$

The best HODLR approximation to $\mathbf{A}$ is obtained by applying a rank-k SVD to each low-rank block of $\mathbf{A}$.

- This is too expensive in the matrix-vector product model (n products)

In the special case that $\mathbf{A} \in \mathrm{HODLR}(k)$, then we require $\widetilde{\mathbf{A}} = \mathbf{A}$ (regardless of ε).

- There are several matvec algorithms for this setting[8]

[8] Lin, Lu, and Ying 2011; Martinsson 2016; Levitt and Martinsson 2022; Halikias and Townsend 2023.

The HODLR approximation problem

Problem. Given an $n \times n$ matrix $\mathbf{A}$, accessible only by matrix-vector products, a rank parameter k, and an accuracy parameter ε, find a HODLR(k) matrix $\widetilde{\mathbf{A}}$ such that

$$\mathbb{E}[\|\mathbf{A} - \widetilde{\mathbf{A}}\|_{\mathsf{F}}] \le (1+\varepsilon) \min_{\mathbf{H} \in \mathrm{HODLR}(k)} \|\mathbf{A} - \mathbf{H}\|_{\mathsf{F}}.$$

The best HODLR approximation to $\mathbf{A}$ is obtained by applying a rank-k SVD to each low-rank block of $\mathbf{A}$.

- This is too expensive in the matrix-vector product model (n products)

In the special case that $\mathbf{A} \in \mathrm{HODLR}(k)$, then we require $\widetilde{\mathbf{A}} = \mathbf{A}$ (regardless of ε).

- There are several matvec algorithms for this setting[8]

[8]Lin, Lu, and Ying 2011; Martinsson 2016; Levitt and Martinsson 2022; Halikias and Townsend 2023.

Learning low-rank matrices from matrix-vector products

The Randomized SVD (RSVD) is a well-known algorithm for obtaining a low-rank approximation to a matrix $\mathbf{B}$:

1. Sample Gaussian matrix $\mathbf{\Omega}$
2. Form $\mathbf{Q} = \operatorname{orth}(\mathbf{B\Omega})$
3. Compute $\mathbf{X} = \mathbf{Q}^\top \mathbf{B}$
4. Output $\mathbf{Q}[\![\mathbf{X}]\!]_k$

Theorem. If $\mathbf{\Omega}$ has $O(k/\varepsilon)$ columns, then

$$\|\mathbf{B} - \mathbf{Q}[\![\mathbf{X}]\!]_k\|_\mathsf{F} \le (1+\varepsilon) \min_{\operatorname{rank}(\mathbf{X}) \le k} \|\mathbf{B} - \mathbf{X}\|_\mathsf{F}.$$

Corollary. If $\mathbf{B}$ is rank-k, then $\mathbf{Q}[\![\mathbf{X}]\!]_k = \mathbf{B}$ (with probability one).

Peeling: an algorithm for the recovery problem[9]

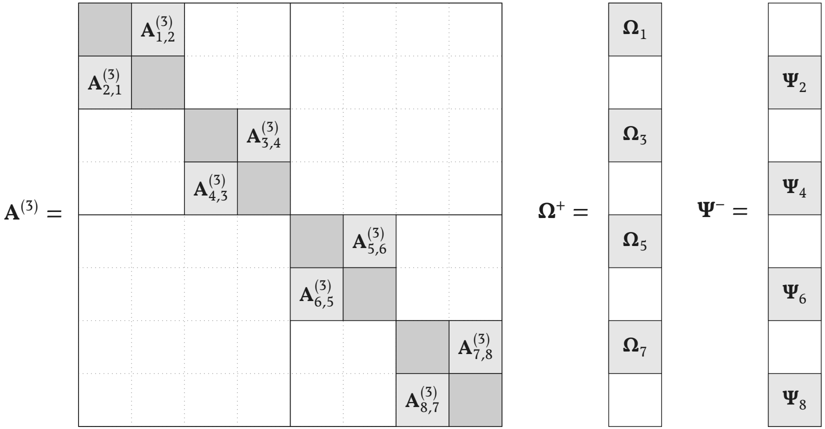
The algorithm works from the top layer down.

At each level, we simultaneosly apply the RSVD to the low-rank off-diagonal blocks.

We then “peel” off these blocks before proceeding to the next level

[9]Lin, Lu, and Ying 2011; Martinsson 2016.

Peeling: an algorithm for the recovery problem



Peeling: an algorithm for the recovery problem

At each level we use k matrix-vector products with $\mathbf{A}$ and $\mathbf{A}^\top$.

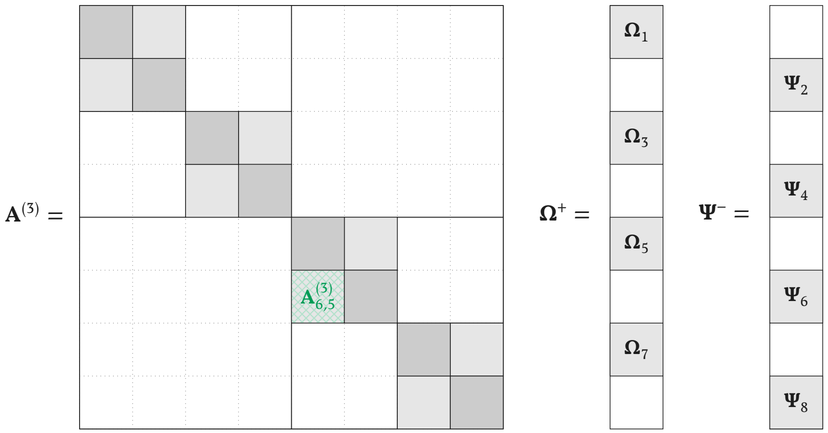
There are $\log_2(n/k) \leq \log_2(n)$ levels until the blocks are of size k

- then we can directly recover them at once with k products

> **Theorem** (Lin, Lu, and Ying 2011)**.** There is an algorithm to recover a HODLR matrix using $O(k \log_2(n))$ products with $\mathbf{A}$.

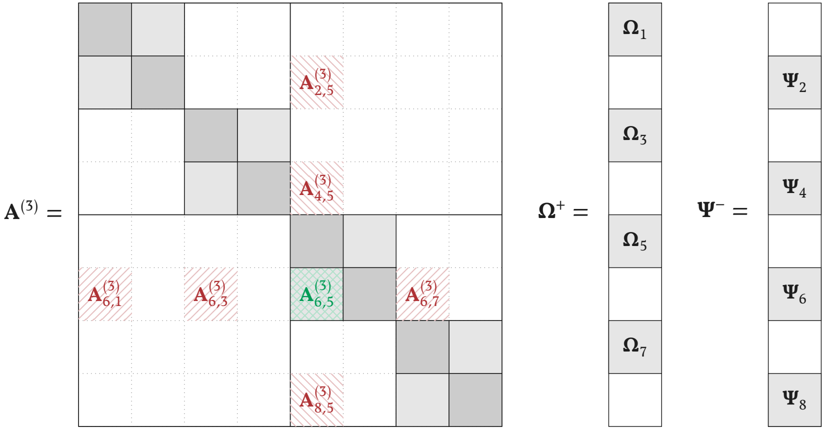
If $\mathbf{A}$ is not HODLR, we can still apply this algorithm, but we might worry about exponential blow-up in the error!

Peeling with error



We obtain perturbed sketches of the form $\mathbf{A}^{(3)}_{6,5}\mathbf{\Omega}_5 + \mathbf{A}^{(3)}_{6,1}\mathbf{\Omega}_1 + \mathbf{A}^{(3)}_{6,3}\mathbf{\Omega}_3 + \mathbf{A}^{(3)}_{6,7}\mathbf{\Omega}_7$ and $(\mathbf{A}^{(3)}_{6,5})^{\mathsf{T}}\mathbf{\Psi}_6 + (\mathbf{A}^{(3)}_{2,5})^{\mathsf{T}}\mathbf{\Psi}_2 + (\mathbf{A}^{(3)}_{4,5})^{\mathsf{T}}\mathbf{\Psi}_4 + (\mathbf{A}^{(3)}_{8,5})^{\mathsf{T}}\mathbf{\Psi}_8$.

Peeling with error



We obtain perturbed sketches of the form $\mathbf{A}^{(3)}_{6,5}\mathbf{\Omega}_5 + \mathbf{A}^{(3)}_{6,1}\mathbf{\Omega}_1 + \mathbf{A}^{(3)}_{6,3}\mathbf{\Omega}_3 + \mathbf{A}^{(3)}_{6,7}\mathbf{\Omega}_7$ and $(\mathbf{A}^{(3)}_{6,5})^\mathsf{T}\mathbf{\Psi}_6 + (\mathbf{A}^{(3)}_{2,5})^\mathsf{T}\mathbf{\Psi}_2 + (\mathbf{A}^{(3)}_{4,5})^\mathsf{T}\mathbf{\Psi}_4 + (\mathbf{A}^{(3)}_{8,5})^\mathsf{T}\mathbf{\Psi}_8$.

Generalized Nyström[10]

The RSVD tries to compute $\mathbf{Q}^\top\mathbf{B}$ directly; this is the solution to:

$$\min_{\mathbf{X}} \|\mathbf{A} - \mathbf{Q}\mathbf{X}\|_\mathsf{F}.$$

Instead, we can solve a sketched problem:

$$\min_{\mathbf{X}} \|\mathbf{\Psi}^\top\mathbf{A} - \mathbf{\Psi}^\top\mathbf{Q}\mathbf{X}\|_\mathsf{F}.$$

This means $\mathbf{X} = (\mathbf{\Psi}^\top\mathbf{Q})^\dagger\mathbf{\Psi}^\top\mathbf{A}$.

Observation. By adding columns to $\mathbf{\Psi}$, we can average out certain errors in the product $\mathbf{\Psi}^\top\mathbf{A}$.

The algorithm is also non-adaptive (we can do products with $\mathbf{\Psi}$ in advance)

[10]Clarkson and Woodruff 2009; Tropp, Yurtsever, Udell, and Cevher 2017; Nakatsukasa 2020.

Generalized Nyström[10]

The RSVD tries to compute $\mathbf{Q}^\top \mathbf{B}$ directly; this is the solution to:

$$\min_{\mathbf{X}} \|\mathbf{A} - \mathbf{Q}\mathbf{X}\|_\mathsf{F}.$$

Instead, we can solve a sketched problem:

$$\min_{\mathbf{X}} \|\boldsymbol{\Psi}^\top \mathbf{A} - \boldsymbol{\Psi}^\top \mathbf{Q}\mathbf{X}\|_\mathsf{F}.$$

This means $\mathbf{X} = (\boldsymbol{\Psi}^\top \mathbf{Q})^\dagger \boldsymbol{\Psi}^\top \mathbf{A}$.

Observation. By adding columns to $\boldsymbol{\Psi}$, we can average out certain errors in the product $\boldsymbol{\Psi}^\top \mathbf{A}$.

The algorithm is also non-adaptive (we can do products with $\boldsymbol{\Psi}$ in advance)

[10] Clarkson and Woodruff 2009; Tropp, Yurtsever, Udell, and Cevher 2017; Nakatsukasa 2020.

Generalized Nyström[10]

The RSVD tries to compute $\mathbf{Q}^\top\mathbf{B}$ directly; this is the solution to:

$$\min_{\mathbf{X}} \|\mathbf{A} - \mathbf{Q}\mathbf{X}\|_{\mathsf{F}}.$$

Instead, we can solve a sketched problem:

$$\min_{\mathbf{X}} \|\boldsymbol{\Psi}^\top\mathbf{A} - \boldsymbol{\Psi}^\top\mathbf{Q}\mathbf{X}\|_{\mathsf{F}}.$$

This means $\mathbf{X} = (\boldsymbol{\Psi}^\top\mathbf{Q})^\dagger\boldsymbol{\Psi}^\top\mathbf{A}$.

Observation. By adding columns to $\boldsymbol{\Psi}$, we can average out certain errors in the product $\boldsymbol{\Psi}^\top\mathbf{A}$.

The algorithm is also non-adaptive (we can do products with $\boldsymbol{\Psi}$ in advance)

[10] Clarkson and Woodruff 2009; Tropp, Yurtsever, Udell, and Cevher 2017; Nakatsukasa 2020.

A perturbation bound for the RSVD

We prove a perturbation bound for the RSVD. This is likely of independent interest.

Theorem (Chen, Keles, Halikias, Musco, Musco, and Persson 2024)**.** Let $\mathbf{Q} = \operatorname{orth}(\mathbf{B\Omega} + \mathbf{E}_1)$ and $\mathbf{X} = \mathbf{Q}^\top\mathbf{B} + \mathbf{E}_2$. Then

$$\|\mathbf{B} - \mathbf{Q}[\![\mathbf{Q}^\top\mathbf{B} + \mathbf{E}_2]\!]_k\|_{\mathsf{F}} \le \underbrace{\|\mathbf{E}_1\mathbf{\Omega}_{\text{top}}^\dagger\|_{\mathsf{F}} + 2\|\mathbf{E}_2\|_{\mathsf{F}}}_{\text{perturbations}} + \underbrace{\left(\|\mathbf{\Sigma}_{\text{bot}}\|_{\mathsf{F}}^2 + \|\mathbf{\Sigma}_{\text{bot}}\mathbf{\Omega}_{\text{bot}}\mathbf{\Omega}_{\text{top}}^\dagger\|_{\mathsf{F}}^2\right)^{1/2}}_{\text{classical RSVD bound}}.$$

Takeaway: The pseudoinverse will help damp the perturbation $\mathbf{E}_1$, but (unsurprisingly) all of the perturbation $\mathbf{E}_2$ can propagate.

Matrix approximation

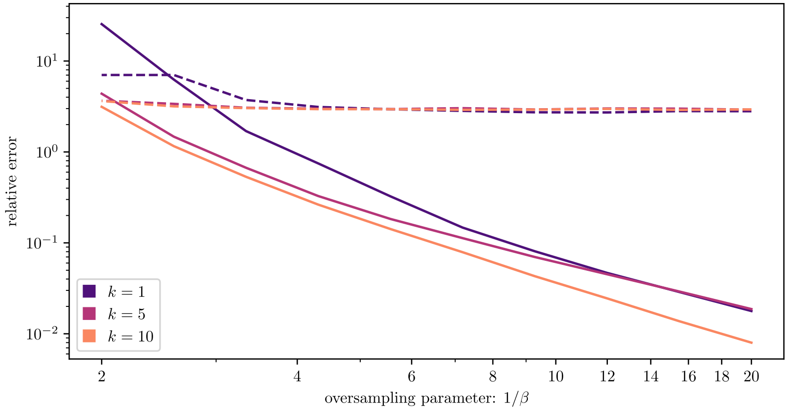
Theorem (Chen, Keles, Halikias, Musco, Musco, and Persson 2024)**.** There exist matvec algorithms which use $O\big(k\log(n)\cdot\operatorname{poly}(1/\beta)\big)$ products with $\mathbf{A}$ to obtain a HODLR(k) matrix $\widetilde{\mathbf{A}}$ satisfying

$$\|\mathbf{A}-\widetilde{\mathbf{A}}\|_{\mathsf{F}} \leq (1+\beta)^{\log_2(n)} \min_{\mathbf{H}\in\mathrm{HODLR}(k)} \|\mathbf{A}-\mathbf{H}\|_{\mathsf{F}}.$$

Corollary. $(1{+}\varepsilon)$-optimal approximation with $O\big(k\log(n)\cdot\operatorname{poly}(\log(n)/\varepsilon)\big)$ matvecs

Corollary. $n^{0.01}$-optimal approximation with $O\big(k\log(n)\big)$ matvecs

Example



solid = Nyström based algorithm,   dashed = RSVD based algorithm

Analysis of peeling

- At each level, we use our perturbation bound to show that we obtain a near-optimal approximation to the blocks at that level, with additional error depending on the quality of approximation at previous levels
- We then analyze the propogation of error through the levels
- The error at any one given level is bounded in terms of the overall best possible error (truncated SVD on each low-rank block)

The fixed-sparsity approximation problem

Fix a $n \times n$ binary matrix $\mathbf{S}$, and consider the set of matrices whose nonzero pattern is given by $\mathbf{S}$; i.e. $S = \{\mathbf{X} : \mathbf{X} \circ \mathbf{S} = \mathbf{X}\}$, where "$\circ$" is entrywise product.

- examples: diagonal matrices, banded matrices

Problem. Given an $n \times n$ matrix $\mathbf{A}$, accessible only by matrix-vector products, find a matrix $\widetilde{\mathbf{A}}$ with sparsity $\mathbf{S}$ such that

$$\|\mathbf{A} - \widetilde{\mathbf{A}}\|_\mathsf{F} \leq (1+\varepsilon) \min_{\mathbf{X} \circ \mathbf{S} = \mathbf{X}} \|\mathbf{A} - \mathbf{X}\|_\mathsf{F}.$$

The best sparse approximation to $\mathbf{A}$ is $\mathbf{A} \circ \mathbf{S}$.

- This is too expensive in the matrix-vector product model (n products)

Lots of past work for exact recovery[11] and diagonal estimation[12]

[11] Coleman and Moré 1983; Coleman and Moré 1983.

[12] Baston and Nakatsukasa 2022; Hallman, Ipsen, and Saibaba 2023; Dharangutte and Musco 2023.

The fixed-sparsity approximation problem: an algorithm

One might try a simple sketching algorithm:

$$\widetilde{\mathbf{A}} = \operatorname*{argmin}_{\mathbf{X}\circ\mathbf{S}=\mathbf{X}} \|\mathbf{AG} - \mathbf{XG}\|_{\mathsf{F}}.$$

Theorem (Amsel, Chen, Keles, Halikias, Musco, and Musco 2024)**.** Suppose $\mathbf{S}$ has at most s nonzeros per row. There exists a matvec algorithms which use $O(s/\varepsilon)$ products with $\mathbf{A}$ to obtain a matrix $\widetilde{\mathbf{A}}$ with sparsity $\mathbf{S}$ satisfying

$$\mathbb{E}[\|\mathbf{A} - \widetilde{\mathbf{A}}\|_{\mathsf{F}}] \leq (1+\varepsilon) \min_{\mathbf{X}\circ\mathbf{S}=\mathbf{X}} \|\mathbf{A} - \mathbf{X}\|_{\mathsf{F}}.$$

Unlike compressed-sensing type problems, no dimension dependence!

- in the compressed sensing setting, we have a necessary $\log(n)$ factor

The fixed-sparsity approximation problem (hardness)

We also prove a lower-bound, which reveals that our algorithm is optimal, up to constant factors.

Theorem (Amsel, Chen, Keles, Halikias, Musco, and Musco 2024)**.** For any $\mathbf{S}$ with $\Theta(s)$ nonzeros per row/column, there exist hard instances for which any algorithm producing $\widetilde{\mathbf{A}}$ with sparsity $\mathbf{S}$ must use $\Omega(s/\varepsilon)$ products with $\mathbf{A}$ in order to guarantee

$$\mathbb{P}\left[\|\mathbf{A} - \widetilde{\mathbf{A}}\|_\mathsf{F} \leq (1+\varepsilon) \min_{\mathbf{X}\circ\mathbf{S}=\mathbf{X}} \|\mathbf{A} - \mathbf{X}\|_\mathsf{F} \right] > 1/2.$$

Lower bound case study: diagonal approximation

To illustrate the key ideas, let's consider the special case $\mathbf{S} = \mathbf{I}$; i.e. diagonal approximation.

Theorem. There exists a constant $C > 0$ such that the following holds:

For any $\varepsilon > 0$, there there exists a distribution $\mathbf{A}$ on $d \times d$ matrices such that, any algorithm which uses $m < C/\varepsilon$ queries cannot output an approximation $\mathbf{d}$ to $\operatorname{diag}(\mathbf{A})$ satisfying

$$\mathbb{P}\Big[\|\operatorname{diag}(\mathbf{A}) - \mathbf{d}\|_2^2\Big] \leq \varepsilon\|\mathbf{A} - \mathbf{A} \circ \mathbf{I}\|_\mathsf{F}^2\Big] > 1/2.$$

If $\widetilde{\mathbf{A}} = \operatorname{diag}(\mathbf{d})$ and the event in the probability holds,

$$\|\mathbf{A} - \widetilde{\mathbf{A}}\|_\mathsf{F}^2 = \|\mathbf{A} \circ \mathbf{S} - \widetilde{\mathbf{A}}\|_\mathsf{F}^2 + \|\mathbf{A} - \mathbf{A} \circ \mathbf{I}\|_\mathsf{F}^2 \leq (1+\varepsilon)\|\mathbf{A} - \mathbf{A} \circ \mathbf{I}\|_\mathsf{F}^2.$$

Since $\sqrt{1+\varepsilon} = 1 + O(\varepsilon)$, this is also a lower bound for the approximation problem.

Gaussian matrices after adaptive queries

Lemma. Let $\mathbf{A} \sim \text{Gaussian}(d, d)$. Suppose we make a sequence of adaptive queries $\mathbf{x}_1, \ldots, \mathbf{x}_m$ with responses $\mathbf{y}_1 = \mathbf{A}\mathbf{x}_1, \ldots, \mathbf{y}_m = \mathbf{A}\mathbf{x}_m$.

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m]$ and $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_m]$. Then, $\mathbf{A}$ can be factored as

$$\mathbf{A} = \begin{bmatrix} \mathbf{Y} & \mathbf{G} \end{bmatrix} \begin{bmatrix} \mathbf{X}^\top \\ \mathbf{Z}^\top \end{bmatrix},$$

where $\mathbf{Z}^\top \mathbf{X} = \mathbf{0}$, $\mathbf{Z}^\top \mathbf{Z} = \mathbf{I}$, and $\mathbf{G} \sim \text{Gaussian}(d, d - m)$, independently of $\mathbf{X}$ and $\mathbf{Y}$.

Conditioned on the queries and responses, $\mathbf{A}$ still has a lot of randomness!

Analysis

We have that

$$\mathbf{A} = \mathbf{Y}\mathbf{X}^\top + \mathbf{G}\mathbf{Z}^\top,$$

where $\mathbf{Z}^\top\mathbf{X} = \mathbf{0}$, $\mathbf{Z}^\top\mathbf{Z} = \mathbf{I}$, and $\mathbf{G} \sim \text{Gaussian}(d, d-m)$, independently of $\mathbf{X}$ and $\mathbf{Y}$.

Conditioned on the queries and responses,

$$\mathbf{A}_{i,i} = \mathbf{e}_i^\top\mathbf{Y}\mathbf{X}^\top\mathbf{e}_i + \mathbf{e}_i^\top\mathbf{G}\mathbf{Z}^\top\mathbf{e}_i = \text{determistic} + \mathbf{g}_i^\top\mathbf{z}_i.$$

Observe that the $\mathbf{g}_i$ are all independent Gaussian vectors of length $d-m$. Hence

$$d_i = \mathbf{g}_i^\top\mathbf{z}_i \sim N(0, \|\mathbf{z}_i\|^2) \text{ independently.}$$

Suppose we try to output an approximation $\mathbf{d}$ to $\text{diag}(\mathbf{A})$. Based on what we know (queries and responses), the best thing we can do is output $\mathbf{d} = \text{diag}(\mathbf{Y}\mathbf{X}^\top)$. So then

$$\mathbb{E}[\|\text{diag}(\mathbf{A}) - \mathbf{d}\|_2^2] = \sum_{i=1}^n \mathbb{E}[d_i^2] = \sum_{i=1}^n \|\mathbf{z}_i\|^2 = \|\mathbf{Z}\|_F^2 = d-m.$$

Choosing the parameters

Applying concentration inequality,

$$\mathbb{P}\big[\,\|\operatorname{diag}(\mathbf{A}) - \mathbf{d}\|_2^2 > 100(d-m)\big] \le \text{small.}$$

In addition,

$$\mathbb{P}\big[\,\|\mathbf{A} - \mathbf{A}\circ\mathbf{I}\|_\mathsf{F}^2 < d^2/100\big] \le \text{small.}$$

If we set $d = 5000/\varepsilon$ and $m < 2500/\varepsilon$,

$$100(d-m) > 100d/2 = \varepsilon d^2/100.$$

Therefore, by a union bound,

$$\mathbb{P}\big[\,\|\operatorname{diag}(\mathbf{A}) - \mathbf{d}\|_2^2 < \varepsilon\|\mathbf{A} - \mathbf{A}\circ\mathbf{I}\|_\mathsf{F}^2\big] \le \text{small.}$$

Conclusion

We saw two broad directions in randomized linear algebra that I have worked on.

I'm also intereted in many other problems including (but certainly not limited to):

- (partial) trace approximation
- spectrum approximation
- classical Krylov subspace methods

References I

Amsel, Noah et al. (2024). *Fixed-sparsity matrix approximation from matrix-vector products*.

Baston, Robert A. and Yuji Nakatsukasa (2022). *Stochastic diagonal estimation: probabilistic bounds and an improved algorithm*.

Bollapragada, Raghu, Tyler Chen, and Rachel Ward (2024). “On the fast convergence of minibatch heavy ball momentum”. In: *IMA Journal of Numerical Analysis*. To appear.

Braverman, Vladimir, Aditya Krishnan, and Christopher Musco (June 2022). “Sublinear time spectral density estimation”. In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. arXiv cs.DS 2104.03461. ACM.

Campisi, Michele, David Zueco, and Peter Talkner (Oct. 2010). “Thermodynamic anomalies in open quantum systems: Strong coupling effects in the isotropic XY model”. In: *Chemical Physics* 375.2-3, pp. 187–194.

Chen, Tyler and Yu-Chen Cheng (Aug. 2022). “Numerical computation of the equilibrium-reduced density matrix for strongly coupled open quantum systems”. In: *The Journal of Chemical Physics* 157.6, p. 064106.

Chen, Tyler, Thomas Trogdon, and Shashanka Ubaru (July 2021). “Analysis of stochastic Lanczos quadrature for spectrum approximation”. In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 1728–1739.

— (2022). *Randomized matrix-free quadrature for spectrum and spectral sum approximation*.

Chen, Tyler et al. (2024). *Near-optimal hierarchical matrix approximation from matrix-vector products*.

Clarkson, Kenneth L. and David P. Woodruff (May 2009). “Numerical linear algebra in the streaming model”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. STOC ’09. ACM.

References II

Coleman, Thomas F. and Jorge J. Moré (Feb. 1983). “Estimation of Sparse Jacobian Matrices and Graph Coloring Blems”. In: *SIAM Journal on Numerical Analysis* 20.1, pp. 187–209.

Dharangutte, Prathamesh and Christopher Musco (Jan. 2023). “A Tight Analysis of Hutchinson’s Diagonal Estimator”. In: *Symposium on Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics, pp. 353–364.

Halikias, Diana and Alex Townsend (Sept. 2023). “Structured matrix recovery from matrix-vector products”. In: *Numerical Linear Algebra with Applications* 31.1.

Halko, N. , P. G. Martinsson, and J. A. Tropp (2011). “Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions”. In: *SIAM Rev.* 53.2, pp. 217–288.

Hallman, Eric, Ilse C. F. Ipsen, and Arvind K. Saibaba (Mar. 2023). “Monte Carlo Methods for Estimating the Diagonal of a Real Symmetric Matrix”. In: *SIAM Journal on Matrix Analysis and Applications* 44.1, pp. 240–269.

Han, Insu et al. (Jan. 2017). “Approximating Spectral Sums of Large-Scale Matrices using Stochastic Chebyshev Approximations”. In: *SIAM Journal on Scientific Computing* 39.4, A1558–A1585.

Huang, De et al. (2021). “Matrix concentration for products”. In: *Foundations of Computational Mathematics*, pp. 1–33.

Ingold, Gert-Ludwig, Peter Hänggi, and Peter Talkner (June 2009). “Specific heat anomalies of open quantum systems”. In: *Physical Review E* 79.6.

Jain, Prateek et al. (2018). “Accelerating stochastic gradient descent for least squares regression”. In: *Conference On Learning Theory*. PMLR, pp. 545–604.

Levitt, James and Per-Gunnar Martinsson (2022). *Randomized Compression of Rank-Structured Matrices Accelerated with Graph Coloring*.

References III

Li, Ruipeng et al. (Jan. 2019). “The Eigenvalues Slicing Library (EVSL): Algorithms, Implementation, and Software”. In: *SIAM Journal on Scientific Computing* 41.4, pp. C393–C415.

Lin, Lin, Jianfeng Lu, and Lexing Ying (May 2011). “Fast construction of hierarchical matrix representation from matrix–vector multiplication”. In: *Journal of Computational Physics* 230.10, pp. 4071–4087.

Martinsson, Per-Gunnar (Jan. 2016). “Compressing Rank-Structured Matrices via Randomized Sampling”. In: *SIAM Journal on Scientific Computing* 38.4, A1959–A1986.

Musco, Cameron and Christopher Musco (2015). *Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition*. Montreal, Canada.

Nakatsukasa, Yuji (2020). “Fast and stable randomized low-rank matrix approximation”. In: *ArXiv* abs/2009.11392.

Polizzi, Eric (Mar. 2009). “Density-matrix-based algorithm for solving eigenvalue problems”. In: *Physical Review B* 79.11.

Talkner, Peter and Peter Hänggi (Oct. 2020). “Colloquium : Statistical mechanics and thermodynamics at strong coupling: Quantum and classical”. In: *Reviews of Modern Physics* 92.4.

Tropp, Joel A. (2015). “An Introduction to Matrix Concentration Inequalities”. In: *Foundations and Trends® in Machine Learning* 8.1-2, pp. 1–230.

Tropp, Joel A. and Robert J. Webber (2023). *Randomized algorithms for low-rank matrix approximation: Design, analysis, and applications*.

Tropp, Joel A. et al. (Jan. 2017). “Practical Sketching Algorithms for Low-Rank Matrix Approximation”. In: *SIAM Journal on Matrix Analysis and Applications* 38.4, pp. 1454–1485.

References IV

Ubaru, Shashanka, Jie Chen, and Yousef Saad (Jan. 2017). “Fast Estimation of $\mathrm{tr}(f(A))$ via Stochastic Lanczos Quadrature”. In: *SIAM Journal on Matrix Analysis and Applications* 38.4, pp. 1075–1099.

Quantum equilibrium thermodynamics

Consider a quantum system consisting of subsystems (s) and (b) with Hamiltonian

$$\mathbf{H} = \bar{\mathbf{H}}_\mathrm{s} + \bar{\mathbf{H}}_\mathrm{b} + \mathbf{H}_\mathrm{sb}, \qquad \bar{\mathbf{H}}_\mathrm{s} = \mathbf{H}_\mathrm{s} \otimes \mathbf{I}_\mathrm{b}, \quad \bar{\mathbf{H}}_\mathrm{b} = \mathbf{I}_\mathrm{s} \otimes \mathbf{H}_\mathrm{b}. \tag{1}$$

In thermal equilibrium at interver temperature β, the state of the system is described by a density matrix

$$\boldsymbol{\rho}_\mathrm{t}(\beta) = \frac{\exp(-\beta\mathbf{H})}{Z_\mathrm{t}(\beta)}, \qquad Z_\mathrm{t}(\beta) = \mathrm{tr}(\exp(-\beta\mathbf{H}); \tag{2}$$

The denisty matrix for subsystem (s) is given by

$$\boldsymbol{\rho}^*(\beta) = \mathrm{tr}_\mathrm{b}(\boldsymbol{\rho}_\mathrm{t}(\beta)) = \frac{\mathrm{tr}_\mathrm{b}(\exp(-\beta\mathbf{H}))}{\mathrm{tr}(\exp(-\beta\mathbf{H}))}, \tag{3}$$

where $\mathrm{tr}_b(\,\cdot\,)$ is the *partial trace* over subsystem (b).[13]

[13] Campisi, Zueco, and Talkner 2010; Ingold, Hänggi, and Talkner 2009; Talkner and Hänggi 2020.

von Neumann entropy of Heisenberg spin chains

The von Neumann entropy $-\operatorname{tr}(\boldsymbol{\rho}^*(\beta)\ln(\boldsymbol{\rho}^*(\beta)))$ is a measure of the entanglement betweeen subsystems (s) and (b).

Understanding the von Neumann entropy for a range of a system with Hamiltonian $\mathbf{H}(\theta)$ at a range of parameter values θ and inverse temperatures β is of interest.

We will consider a special case

$$\mathbf{H} = \sum_{i,j} \left[J^{\mathrm{x}}_{i,j} \boldsymbol{\sigma}^{\mathrm{x}}_i \boldsymbol{\sigma}^{\mathrm{x}}_j + J^{\mathrm{y}}_{i,j} \boldsymbol{\sigma}^{\mathrm{y}}_i \boldsymbol{\sigma}^{\mathrm{y}}_j + J^{\mathrm{z}}_{i,j} \boldsymbol{\sigma}^{\mathrm{z}}_i \boldsymbol{\sigma}^{\mathrm{z}}_j \right] + \frac{h}{2} \sum_{i=1}^{N} \boldsymbol{\sigma}^{\mathrm{z}}_i .$$

where h is the magnetic field strength.

Subsystem (s) corresponds to $i = 1, 2$ and subsystem (b) corresponds to the rest of the spins.

Key question:

How to compute reduced density matrices numerically?

A starting point: stochastic trace estimation

If $\mathbf{b}$ is a standard Gaussian random vector:

$$\mathbb{E}[\mathbf{b}^\mathsf{T} f(\mathbf{A})\mathbf{b}] = \operatorname{tr}(f(\mathbf{A})), \qquad \mathbb{V}[\mathbf{b}^\mathsf{T} f(\mathbf{A})\mathbf{b}] = 2\|f(\mathbf{A})\|_\mathsf{F}^2.$$

It's standard to use a KSM to approximate products $\mathbf{b} \mapsto \mathbf{b}^\mathsf{T} f(\mathbf{A})\mathbf{b}$.

Lots of work balancing the cost of the KSM with the variance of the estimator[14].

[14] Han, Malioutov, Avron, and Shin 2017; Ubaru, Chen, and Saad 2017; Chen, Trogdon, and Ubaru 2021; Chen, Trogdon, and Ubaru 2022; Braverman, Krishnan, and Musco 2022.

Partial traces

Suppose $\mathbf{A}$ is a $d_\mathrm{s}d_\mathrm{b} \times d_\mathrm{s}d_\mathrm{b}$ matrtix partitioned as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,d_\mathrm{s}} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \cdots & \mathbf{A}_{2,d_\mathrm{s}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{d_\mathrm{s},1} & \mathbf{A}_{d_\mathrm{s},2} & \cdots & \mathbf{A}_{d_\mathrm{s},d_\mathrm{s}} \end{bmatrix},$$

Partial traces

Then the partial trace (wrt. this partitioning) is defined as:

$$\mathrm{tr}_{\mathrm{b}}(\mathbf{A}) = \begin{bmatrix} \mathrm{tr}(\mathbf{A}_{1,1}) & \mathrm{tr}(\mathbf{A}_{1,2}) & \cdots & \mathrm{tr}(\mathbf{A}_{1,d_{\mathrm{s}}}) \\ \mathrm{tr}(\mathbf{A}_{2,1}) & \mathrm{tr}(\mathbf{A}_{2,2}) & \cdots & \mathrm{tr}(\mathbf{A}_{2,d_{\mathrm{s}}}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{tr}(\mathbf{A}_{d_{\mathrm{s}},1}) & \mathrm{tr}(\mathbf{A}_{d_{\mathrm{s}},2}) & \cdots & \mathrm{tr}(\mathbf{A}_{d_{\mathrm{s}},d_{\mathrm{s}}}) \end{bmatrix}.$$

An algorithm for partial traces[15]

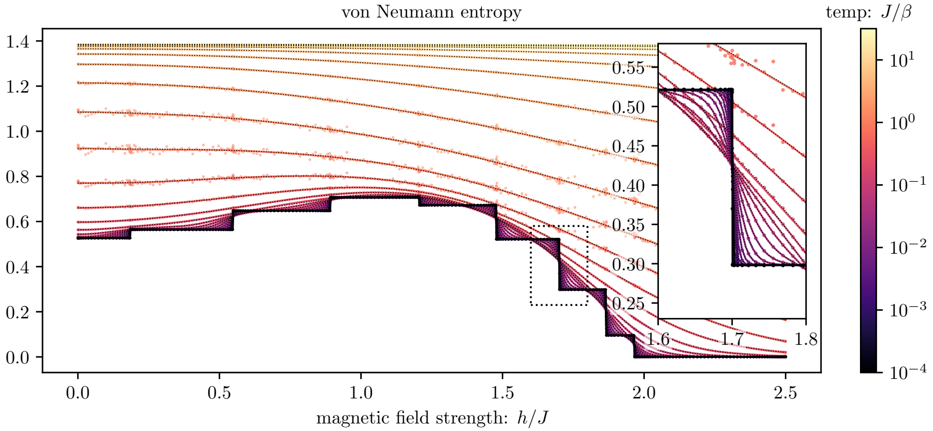
We can use a randomized estimator:

$$(\mathbf{I}_{d_s} \otimes \mathbf{b})^\top \mathbf{A} (\mathbf{I}_{d_s} \otimes \mathbf{b}) = \begin{bmatrix} \mathbf{b}^\top \mathbf{A}_{1,1} \mathbf{b} & \mathbf{b}^\top \mathbf{A}_{1,2} \mathbf{b} & \cdots & \mathbf{b}^\top \mathbf{A}_{1,d_s} \mathbf{b} \\ \mathbf{b}^\top \mathbf{A}_{2,1} \mathbf{b} & \mathbf{b}^\top \mathbf{A}_{2,2} \mathbf{b} & \cdots & \mathbf{b}^\top \mathbf{A}_{2,d_s} \mathbf{b} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{b}^\top \mathbf{A}_{d_s,1} \mathbf{b} & \mathbf{b}^\top \mathbf{A}_{d_s,2} \mathbf{b} & \cdots & \mathbf{b}^\top \mathbf{A}_{d_s,d_s} \mathbf{b} \end{bmatrix}.$$

Then use a KSM to approximate products with $\mathbf{A} = f(\mathbf{H})$.

[15] Chen and Cheng 2022.

von Neumann entropy



Spectrum approximation

Recall a symmetric matrix $\mathbf{A}$ has a real spectrum $\{\lambda_i\}$. We can encode the spectrum in the spectral density function

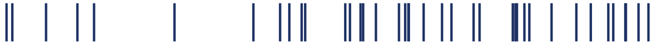
$$\varphi(x) = \sum_{i=1}^{n} \frac{1}{n}\delta(x - \lambda_i).$$

Here $\delta(x)$ is a Dirac delta point mass centered at zero.

> **Problem.** Given only matrix-vector product access to $\mathbf{A}$, approximate $\varphi(x)$.

Example application: high performance computing

State of the art parallel eigensolvers such as FEAST and EVSL work by splitting the spectrum of $\mathbf{A}$ into pieces, which can each be solved on different machines in parallel.[16]



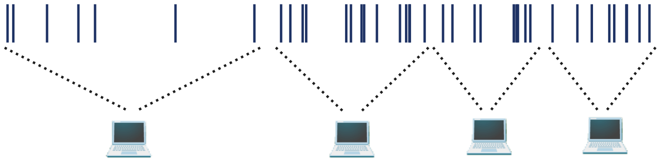
The spectral density tells us how many eigenvalues in a region

$$\text{number of eigenvalues in } [a,b] = n\int_a^b \varphi(x)\mathrm{d}x.$$

[16]Polizzi 2009; Li, Xi, Erlandson, and Saad 2019.

Example application: high performance computing

State of the art parallel eigensolvers such as FEAST and EVSL work by splitting the spectrum of $\mathbf{A}$ into pieces, which can each be solved on different machines in parallel.[16]



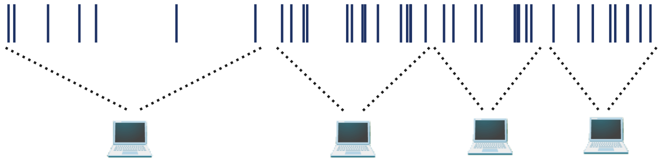
The spectral density tells us how many eigenvalues in a region

$$\text{number of eigenvalues in } [a,b] = n\int_a^b \varphi(x)\mathrm{d}x.$$

[16] Polizzi 2009; Li, Xi, Erlandson, and Saad 2019.

Example application: high performance computing

State of the art parallel eigensolvers such as FEAST and EVSL work by splitting the spectrum of $\mathbf{A}$ into pieces, which can each be solved on different machines in parallel.[16]



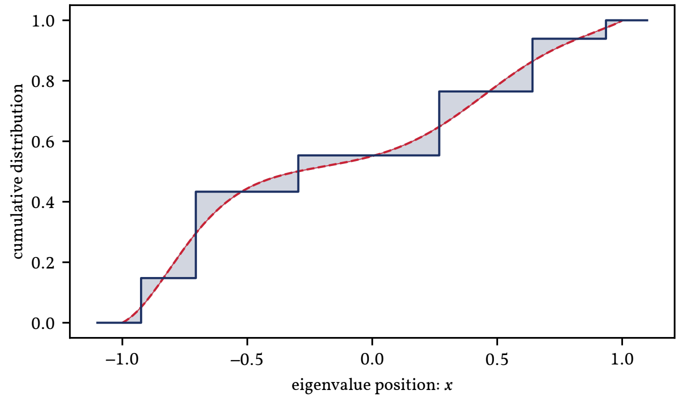
The spectral density tells us how many eigenvalues in a region

$$\text{number of eigenvalues in } [a,b] = n\int_a^b \varphi(x)\,\mathrm{d}x.$$

[16]Polizzi 2009; Li, Xi, Erlandson, and Saad 2019.

Wasserstein distance

To measure the error in approximating $\varphi(x)$, we use the Wasserstein distance.



Stochatic Lanczos Quadrature

A widely used algorithm for approximating spectral densities Stocahstic Lanczos Quadrature (SLQ).

- used in ML to study the training of neural networks
- used in quantum physics for understanding many different systems

Until recently, no theoretical gurantees!

> **Theorem** (Chen, Trogdon, and Ubaru 2021). For large matrices,[a] SLQ produces an ε accurate approximation to $\varphi(x)$ using $O(1/\varepsilon)$ products with $\mathbf{A}$.
>
> [a]specifically, for $n \gg 1/\varepsilon^2$

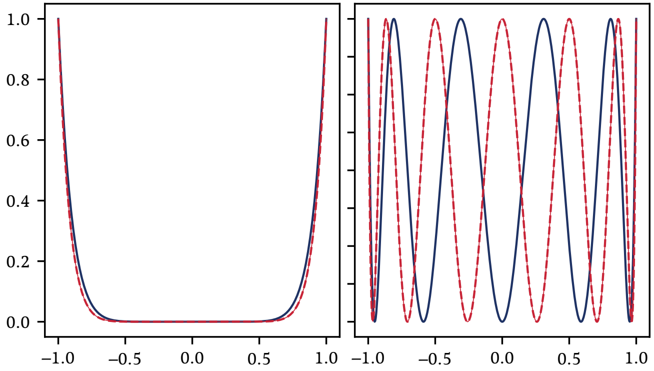
Proof sketch

We use a general technique which actually applies to some other algorithms such as the Kernel Polynomial Method[17].

- If two distributions have similar Chebyshev moments through degree $O(1/\varepsilon)$, they are ε close in Wasserstein distance.
  - This follows from Jackson's Theorem from approximation theory and the dual representation of the Wasserstein distance in terms of 1-Lipshitz functions
- The Chebyshev moments of $\varphi(x)$ are $\int T_k(x)\varphi(x)\mathrm{d}x = n^{-1}\operatorname{tr}(T_k(\mathbf{A}))$
- We can estimate $\operatorname{tr}(T_k(\mathbf{A}))$ using stochastic trace estimation
  - If $\mathbf{x}$ is an isotropic random vector, $\mathbb{E}[\mathbf{x}^\top\mathbf{M}\mathbf{x}] = \operatorname{tr}(\mathbf{M})$ and $\mathbb{V}[\mathbf{x}^\top\mathbf{M}\mathbf{x}] \approx 2\|\mathbf{M}\|_\mathsf{F}$.
- We can compute $\mathbf{x}^\top T_k(\mathbf{A})\mathbf{x}$ using $O(k)$ products with $\mathbf{A}$.

[17] Chen, Trogdon, and Ubaru 2022.

Chebyshev moments


1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0
−1.0
−0.5
0.0
0.5
1.0