Numerical Methods for Data Science: Scalable Kernel Methods, Part I

David Bindel

19 June 2019

Department of Computer Science
Cornell University

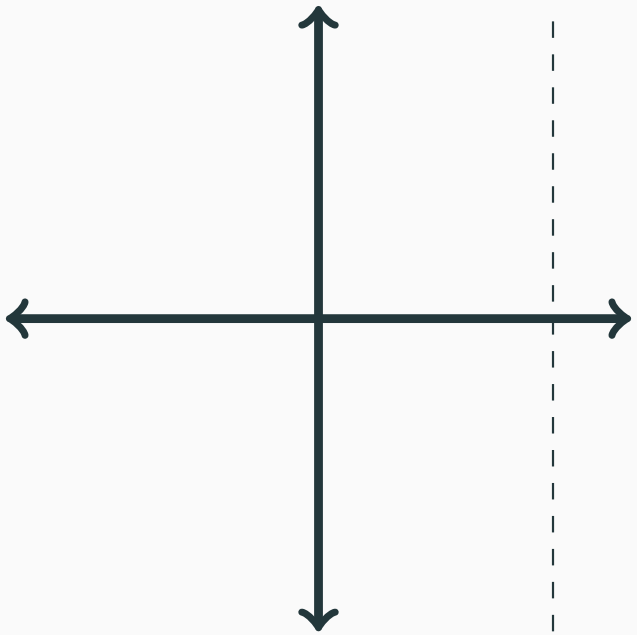
Lecture plan

Three threads from “lay of the land” to current research:

- Monday: Latent Factor Models
- Wednesday: Scalable Kernel Methods
  - 1:30-2:30: Structure and interpretation of kernels
  - 3:00-4:00: Making kernel methods scale
- Friday: Spectral Network Analysis

Slides posted on web page (linked from my Cornell page).

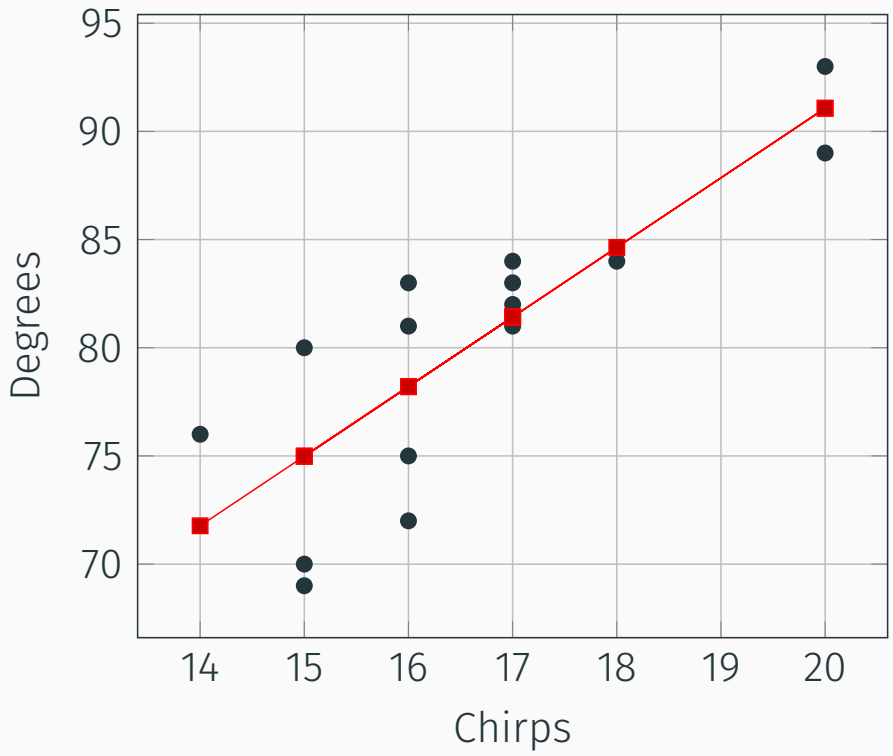
Simple and Impossible



Let $u = (u_1, u_2)$. Given an estimate of u_1, what is u_2?

We need an assumption!

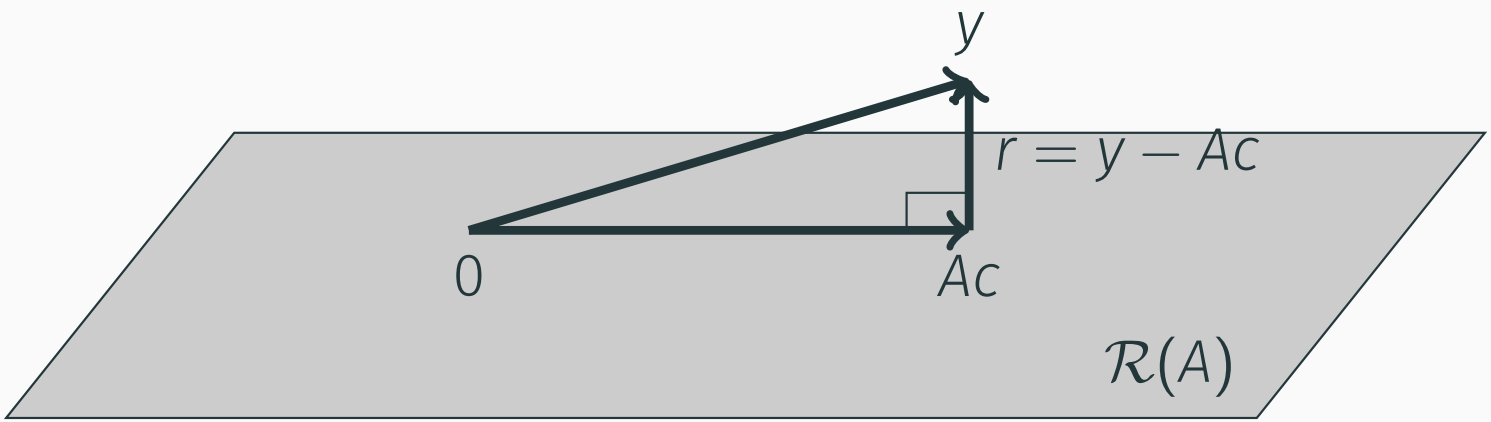
Basic Regression Task



Standard example for today:

Unknown:	$f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$
Data:	$y_j = f(x_j) + \epsilon_j$ for $x_j \in \Omega$
Goal:	Estimate $f(x)$ for general x

The Linear Least Squares Picture



Least squares approach gives good predictions assuming

- *Consistency*: Space contains good approximations
- *Stability*: Space is not sensitive to basis perturbations

In statistical terms, care about *bias* and *variance*.

The Linear Least Squares Picture

Simplest model:

$$f(x) \approx c^T x$$

If overdetermined ($n > d$), solve

$$\text{minimize } \frac{1}{2}\|Ac - y\|_2^2 \text{ where } A^T = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$$

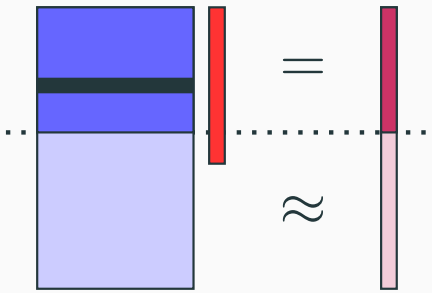
Potential problems:

- Want a different loss? LS still a good building block.
- A poorly conditioned? May need to regularize.
- Model space is not rich enough.

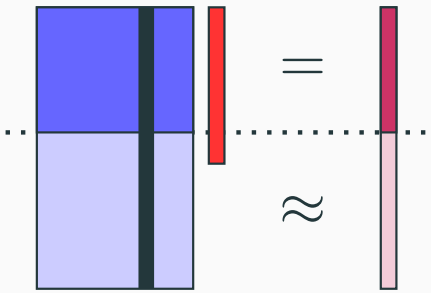
Kernel methods give us richer model spaces.
Usually so rich that we are no longer overdetermined.

Kernel-Based Regression: Four Stories

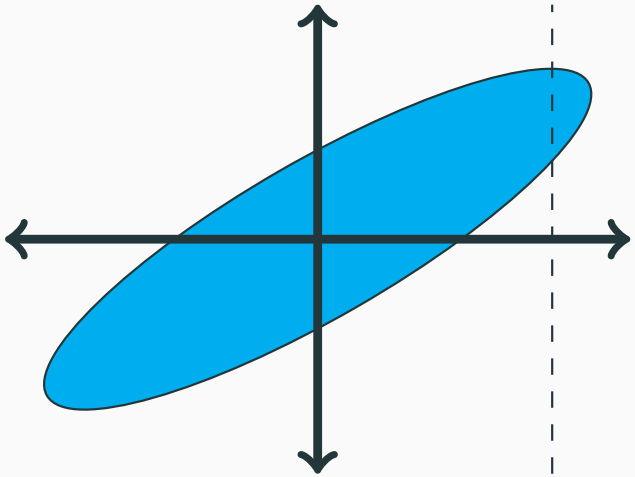
Feature map



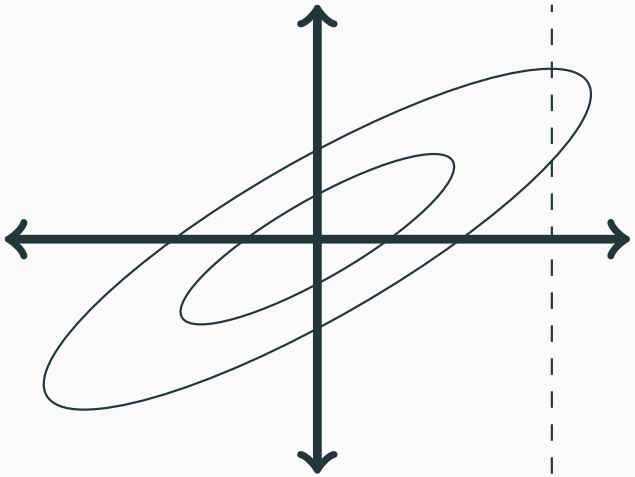
Data-dependent basis



Energy minimization



Gaussian process



Feature Maps and Data-Dependent Bases

Feature Maps

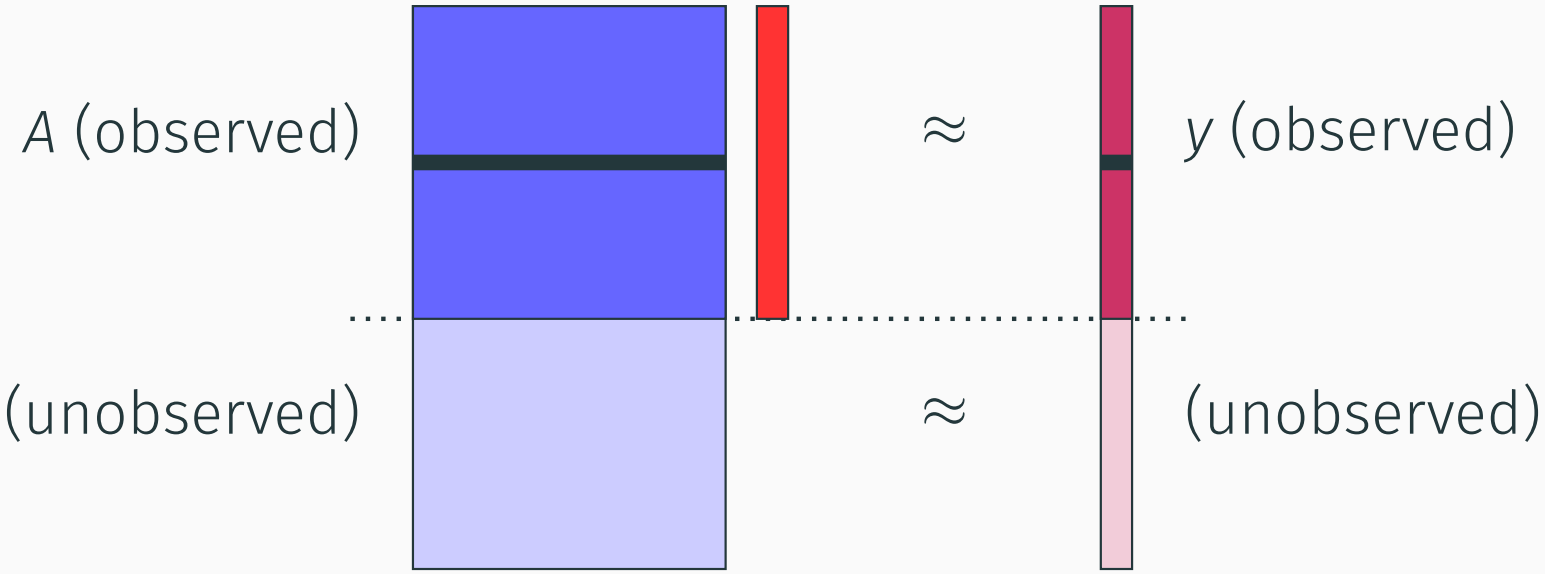
$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix}$$

Augment simple linear model ($c^T x$) with feature map:

$$f(x) \approx \langle d, \psi(x) \rangle$$

where $\psi : \Omega \to \mathcal{F}$ and $d \in \mathcal{F}$, some Hilbert space $\mathcal{F}$.

Feature Maps and Dimensionality

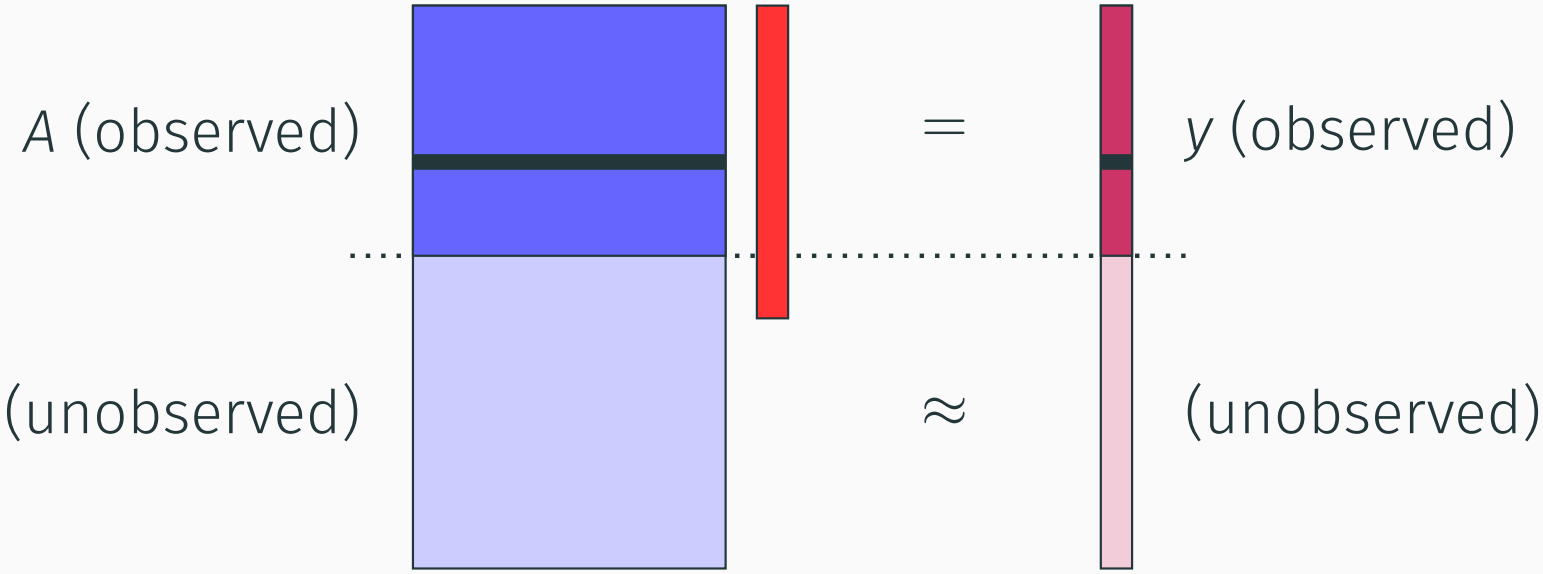


Usual idea with $A^T = \begin{bmatrix} \psi(x_1) & \dots & \psi(x_n) \end{bmatrix}^T$:

- $\dim \mathcal{F} < n$: same least squares as before
- $\dim \mathcal{F} = n$: interpolation problem

May be ill-posed for general scattered (multidimensional) case (Mairhuber-Curtis theorem)

Feature Maps and Dimensionality



Underdetermined ($\dim \mathcal{F} > n$): seek *minimal norm* solution.
For standard inner product (ℓ^2):

$$d = A^\dagger y = A^T(AA^T)^{-1}y$$

$$f(x) \approx \psi(x)^T d = \psi(x)^T A^T (AA^T)^{-1} y$$

Implicit preference for some models over others.

The Kernel Trick (Noise-Free)

Formula:

$$A^T = \begin{bmatrix} \psi(x_1) & \dots & \psi(x_n) \end{bmatrix}$$
$$f(x) \approx s(x) \equiv \psi(x)^T A^T (AA^T)^{-1} y$$

In terms of *kernel* $k(x, y) = \langle \psi(x), \psi(y) \rangle$:

$$(AA^T)_{ij} = k(x_i, x_j) = (K_{XX})_{ij}$$
$$K_{XX} c = y = f_X$$
$$s(x) = K_{xX} c = \sum_{j=1}^{n} k(x, x_j) c_j$$

Subscripts to denote vectors/matrices of function evaluations.

The Kernel Trick

May still want regularization in underdetermined case!

$$\text{minimize } \frac{1}{2}\|r\|^2 + \frac{\eta}{2}\|d\|^2 \quad \text{s.t.} \quad r = y - Ad$$

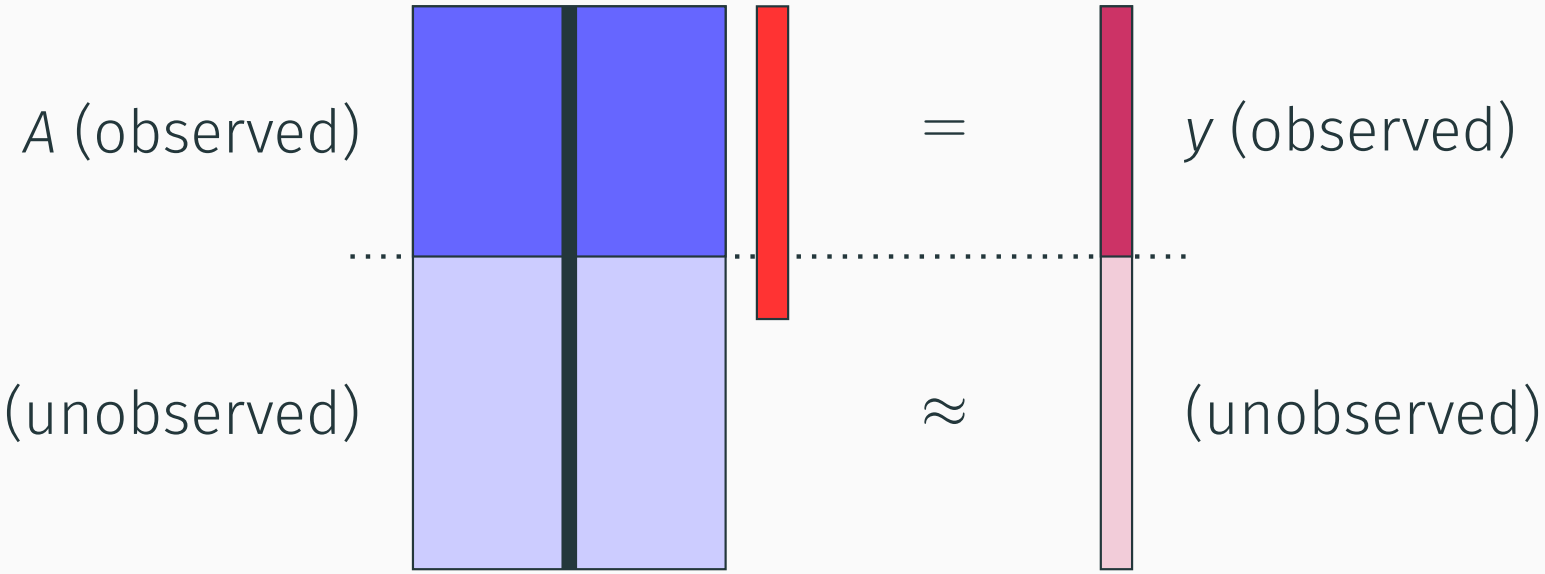
KKT conditions with Lagrange multiplier μ give

$$\begin{aligned} r - \mu &= 0 \\ \eta d - A^T\mu &= 0 \\ y - Ad - r &= 0 \end{aligned} \qquad \begin{aligned} & \\ A^T r &= \eta d \\ r &= y - Ad \end{aligned}$$

Result can again be expressed entirely via kernel:

$$\begin{aligned} (AA^T + \eta I)r &= \eta y \\ (AA^T + \eta I)c &= y \\ s(x) = \psi(x)^T d &= \psi(x)^T A^T c \end{aligned} \qquad \begin{aligned} (K_{XX} + \eta I)r &= \eta y \\ (K_{XX} + \eta I)c &= y \\ s(x) &= K_{xX} c \end{aligned}$$

Rows or Columns?



Change perspective: feature vectors $\psi(x_i)$ to basis vectors $\psi_j(x)$

- Native space: combinations of basis (with $d \in \mathcal{F}$)
- Sampling-dependent subspace of min-norm interpolants:

$$\mathcal{S}_X = \left\{ x \mapsto \sum_{j=1}^{n} k(x, x_j) c_j \right\}$$

Feature Maps to RKHS

Basis vectors $\psi_j(x)$ form orthonormal basis for a Hilbert space:

$$s(x) = \sum_i d_i \psi_i(x), \quad \|s\|_{\mathcal{H}}^2 = \|d\|^2$$

For $x \in \Omega$, define $k_x \in \mathcal{H}$ as

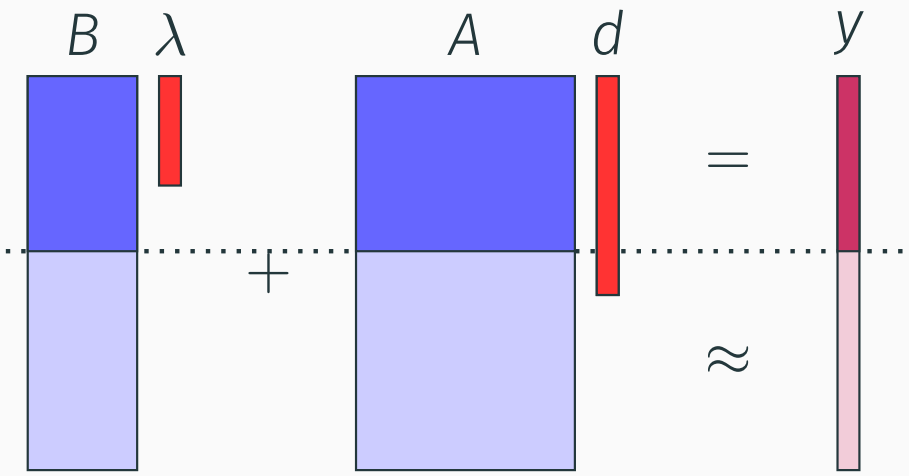
$$k_x(y) = \sum_i \psi_i(x)\psi_i(y)$$

Then k_x defines a point evaluation functional in $\mathcal{H}$

$$\begin{aligned}\langle k_x, s\rangle_{\mathcal{H}} &= \sum_i \psi_i(x) \left\langle \psi_i, \sum_j d_j \psi_j \right\rangle_{\mathcal{H}} \\ &= \sum_i \psi_i(x) d_i = s(x)\end{aligned}$$

This is a *reproducing kernel Hilbert space* (RKHS).

Role of Residual



Can also make d as small as possible for fitting a residual:

$$\text{minimize } \frac{1}{2}\|d\|^2 \text{ s.t. } B\lambda + Ad = y$$

KKT conditions (with c a Lagrange multiplier):

$$\begin{bmatrix} K_{XX} & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} c \\ \lambda \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

Note: Need B nonsingular for well-posedness.

Beyond the Basis

Beyond the Basis

- Story so far involves explicit feature maps.
- Computations only require kernel (inner products).

Putting the Kernel before the Feature Map

Start with symmetric kernel function $k : \Omega \times \Omega \to \mathbb{R}$.
k *positive definite* if K_{XX} spd for all samples X.

Often assume positive definite and:

- **Stationary**: $k(x, y)$ depends only on $x - y$
- **Isotropic**: $k(x, y)$ depends on x and $\|x - y\|$

Both: $k(x, y) = \phi(\|x - y\|)$, ϕ a *radial basis function*.

Have Mercer!

Associate integral operator with continuous spd kernel k:

$$(\mathcal{K}f)(x) = \int k(x,y)f(y)\,dy$$

$\mathcal{K}$ compact (actually Hilbert-Schmidt), so have

$$\mathcal{K} = \sum_{j=1}^{\infty} \lambda_j \psi_j \psi_j^*$$

and features are $\sqrt{\lambda_j}\psi_j(x)$.

But features are not really needed! Focus on the kernel.

From Kernels to Inner Products

To build RKHS without an explicit feature map:

- Observe that $\langle k_x, k_y \rangle_{\mathcal{H}} = k(x, y)$
- For $u(x) = \sum_{i=1}^{N} c_i k(x_i, x)$ and $v(x) = \sum_{i=1}^{N} d_i k(x_i, x)$, have

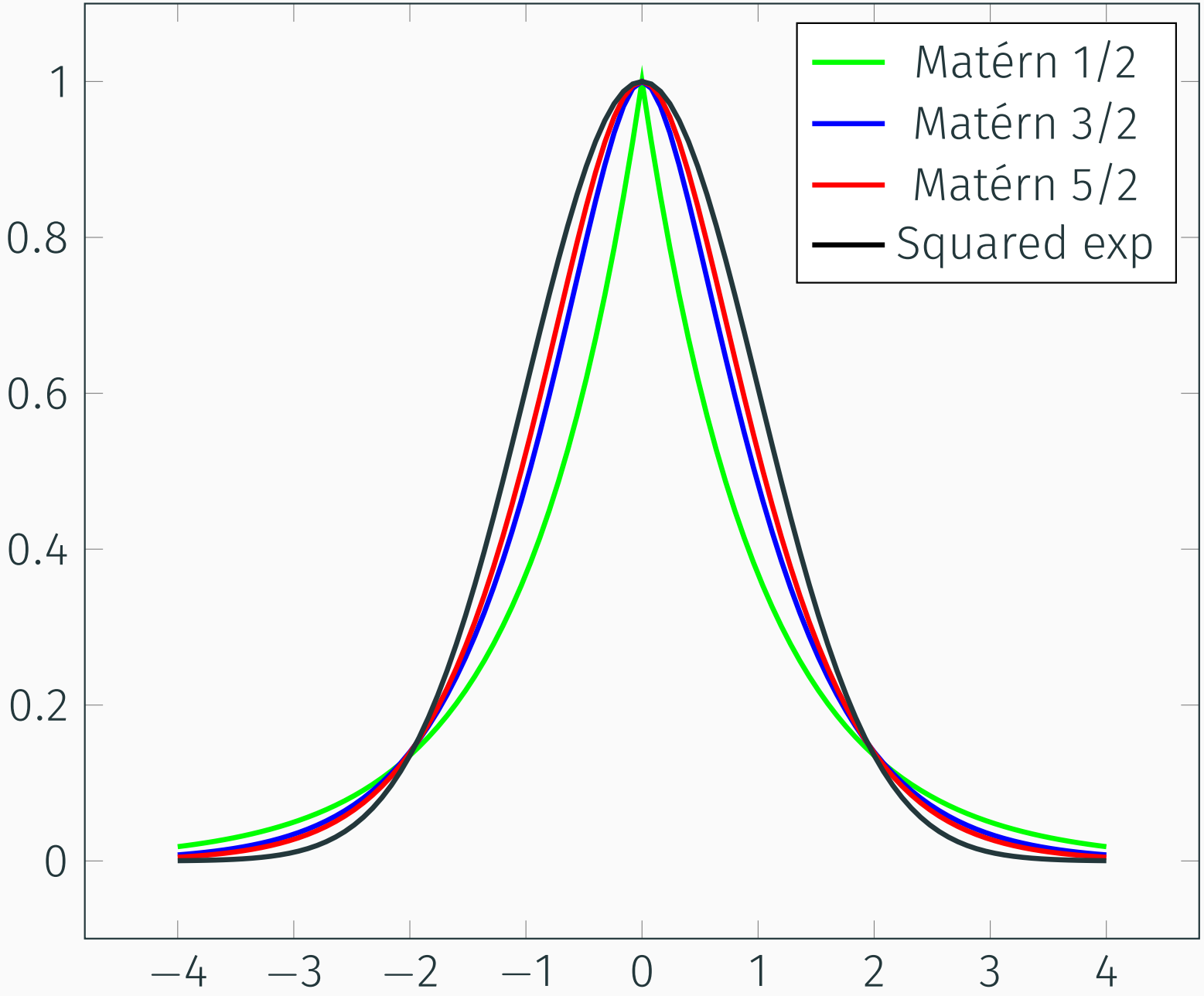
$$\langle u, v \rangle_{\mathcal{H}} = \left\langle \sum_i c_i k_{x_i}, \sum_j d_j k_{x_j} \right\rangle_{\mathcal{H}} = \sum_{i,j} c_i k(x_i, x_j) d_j = d^T K_{XX} c.$$

  Note:

$$\langle u, v \rangle_{\mathcal{H}} = v_X^T K_{XX}^{-1} u_X$$

- Gives pre-Hilbert structure, close to get Hilbert space.

Common Kernels



Common Kernels

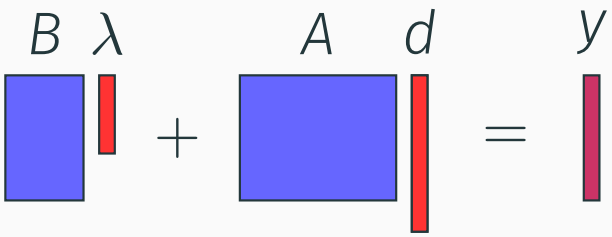
Kernel is *chosen by modeler*

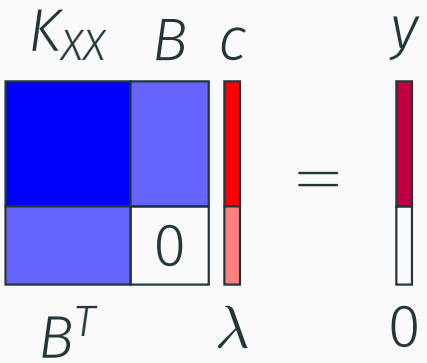
- Choose Matérn / RBF for regularity and simplicity
- Rarely have the intuition to pick the "right" kernel
- Different kernels generate different RKHS
- Common choices are *universal* (RKHS dense in $C(\Omega)$)
  - ... though with less data for a "good" choice

Properties of kernel matrices:

- Positive definite by design, but not well conditioned!
- Weyl: $k(r) \in C^\nu \implies |\lambda_n| = o(n^{-\nu-1/2})$
- RBF case: eigenvalues decay exponentially
- Adding regularization "wipes out" small eigenvalues

Conditionally Positive Definite Case





Consider kernelized "minimize $\mathcal{H}$-norm of residual" picture:

- Mental picture: $K_{XX} = AA^T$ (implicitly)
- But system with $K_{XX} - BMB^T$ gives *same answer* (for any symmetric M)
- And predictions do not depend on changes in B directions:

$$\begin{aligned} s(x) &= K_{xX}c + b(x)^T\lambda \\ &= (K_{xX} + \mu(x)^T B^T)c + b(x)^T\lambda \end{aligned}$$

Conditionally Positive Definite Case

If we have a polynomial fit + minimize $\mathcal{H}$-norm of residual, OK to "cheat" on the kernel definiteness:

- Symmetric $k : \Omega \times \Omega \to \mathbb{R}$
- $\{p_j\}$ a basis for $\mathcal{P}_{m-1}(\Omega)$ (poly of degree $< m$)
- *k conditionally positive definite of order m* if

$$c \neq 0, \Pi_X^T c = 0 \quad \Longrightarrow \quad c^T K_{XX} c > 0$$

  where $[\Pi_X]_{ij} = p_j(x_i)$.

Well-posed problem if Π_X nonsingular.
Jargon: need X well-poised (for polynomial interpolation).

More Common Kernels

	$\phi(r)$	Order
Cubic	r^3	2
Thin-plate	$r^2 \log r$	2
Multiquadric	$-\sqrt{\gamma^2 + r^2}$	1
Inverse multiquadric	$(\gamma^2 + r^2)^{-1/2}$	0
Gaussian	$\exp(-r^2/\gamma^2)$	0

An Aside

General picture: kernel-based approximation + fixed-basis approximation

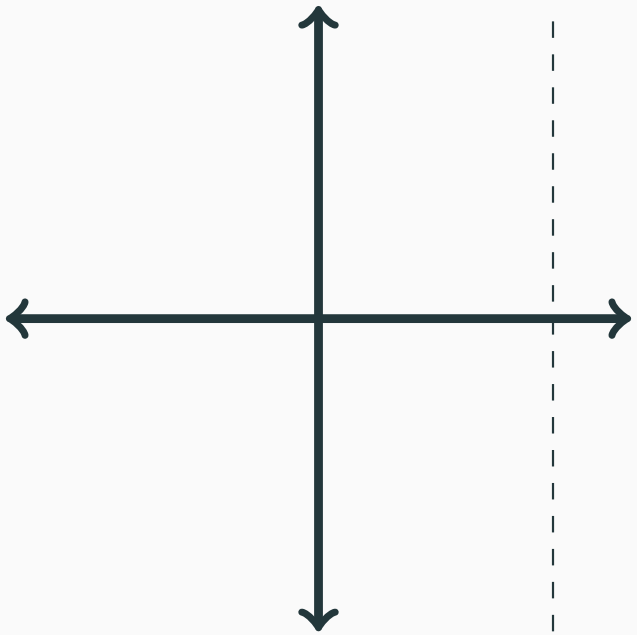
$$s(x) = k_{xX}c + b(x)^T\lambda$$

In CPD case, $b(x)$ is a basis for a polynomial space (the "tail")

- For CPD kernels we *need* a polynomial tail
- Can incorporate polynomial (or other) tails in other cases!
- A "tail" is a good way of putting in domain knowledge

Optimal Recovery vs Gaussian Processes

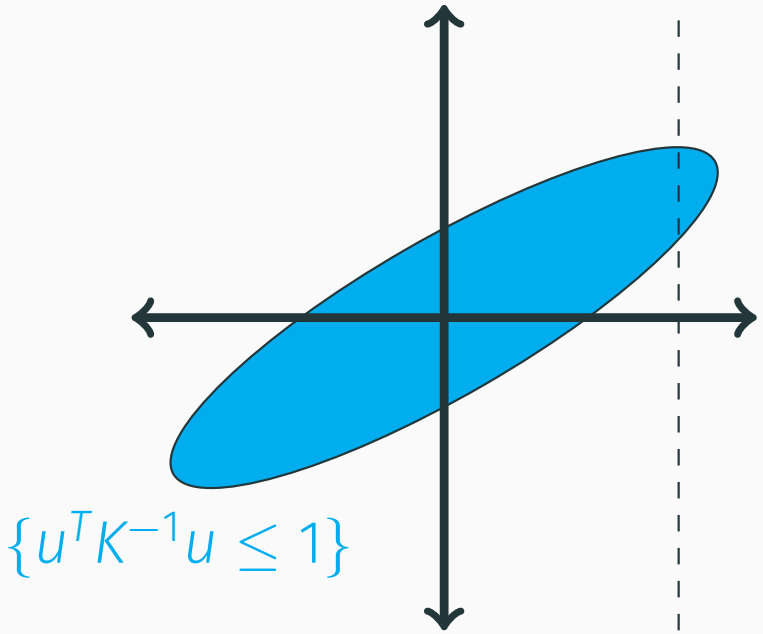
Simple and Impossible



Let $u = (u_1, u_2)$. Given u_1, what is u_2?

We need an assumption! Two different standard takes.

Being Bounded



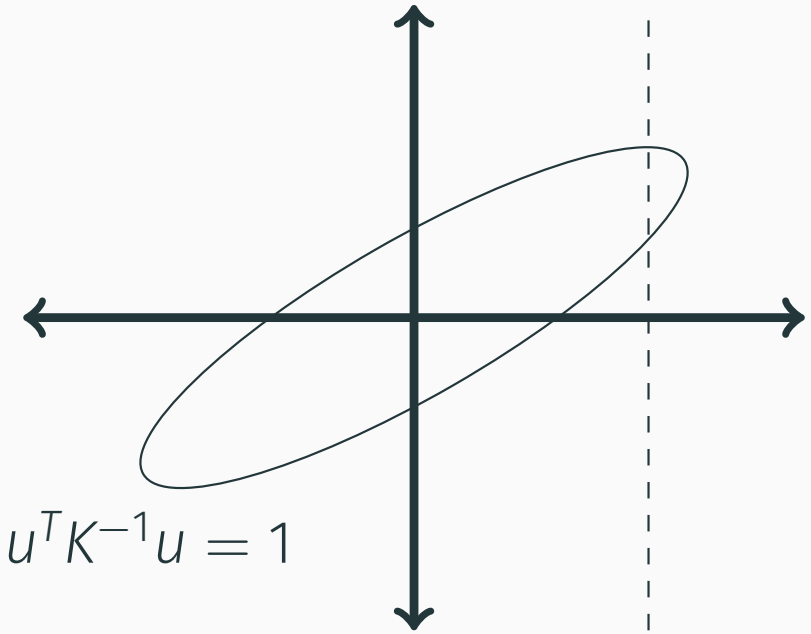
Let $u = (u_1, u_2)$ s.t. $\|u\|^2_{K^{-1}} \leq 1$. Given u_1, what is u_2?

Optimal recovery: $\|u_2 - w\|^2_{S^{-1}} \leq 1 - \|u_1\|^2_{(K_{11})^{-1}}$

$$w = K_{21} K_{11}^{-1} u_1$$

$$S = K_{22} - K_{21} K_{11}^{-1} K_{12}$$

Being Bayesian



Let $U = (U_1, U_2) \sim N(0, K)$. Given $U_1 = u_1$, what is U_2?

Posterior distribution: $(U_2 | U_1 = u_1) \sim N(w, S)$ where

$$w = K_{21} K_{11}^{-1} u_1$$

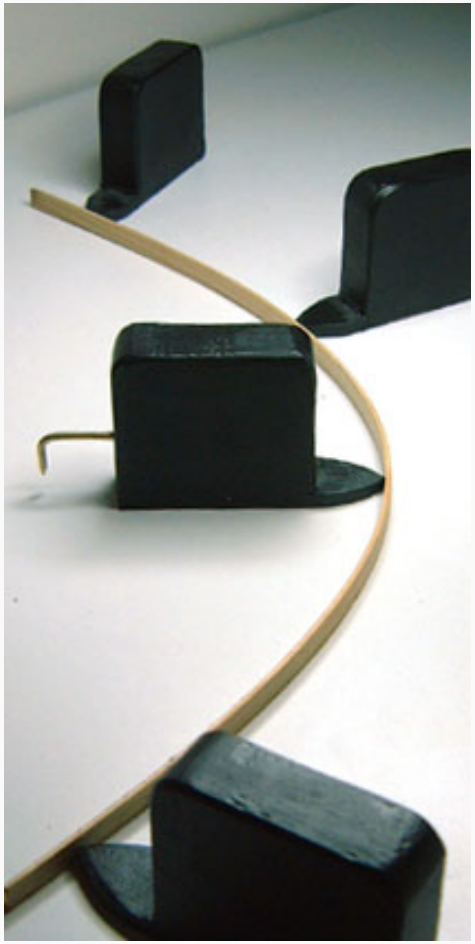
$$S = K_{22} - K_{21} K_{11}^{-1} K_{12}$$

Bounded vs Bayesian

Rest of lecture exploring these two different takes

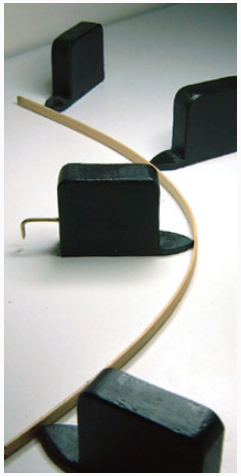
- Optimal recovery
  - Prior: bound in terms of a pos def quadratic
  - Prediction minimizes *worst-case* error given bound + data
  - Error analysis framework: worst-case bounds
- Bayesian inference
  - Prior: joint normal distribution (pos def covariance)
  - Prediction minimizes *average-case* error given prior + data
  - Error analysis framework: posterior distribution

From Energy to Error



http://www.duckworksmagazine.com/03/r/articles/splineducks/splineDucks.htm

Cubic Splines



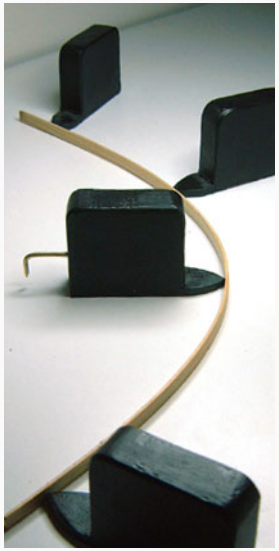
http://www.duckworksmagazine.com/03/r/articles/splineducks/splineDucks.htm

- $\phi(r) = r^3$ is conditionally positive definite of order 2
- Squared (semi-)norm is bending energy:

$$\|s\|_{\mathcal{H}}^2 \propto \frac{1}{2} \int_{\Omega} s''(x)^2 \, dx$$

- Linear polynomial tail = rigid body modes

Force, Displacement, Stiffness



Target function $f \in \mathcal{H}^2$, known bending energy

$$E[f] = \frac{1}{2} \int_\Omega f''(x)^2 \, dx$$

Cubic spline minimizes $E[s]$ s.t. $s(x_i) = f(x_i)$, so

$$E[s] \leq E[f]$$

- $f(x_i)$ as displacement, c_i as corresponding force
- Kernel matrix K_{XX} is compliance (force $\mapsto$ displacement)
- Residual compliance (inverse stiffness) at x is $P_X(x)^{-2}$
- Energy bound for error at X

$$P_X(x)^{-2} \left(s(x) - f(x)\right)^2 \leq E[f] - E[s]$$

General Picture

Interpolant is

$$s(x) = K_{xX}c + b(x)^T\lambda$$

Can compute *power function* $P_X(x)$ from factorization; SPD case:

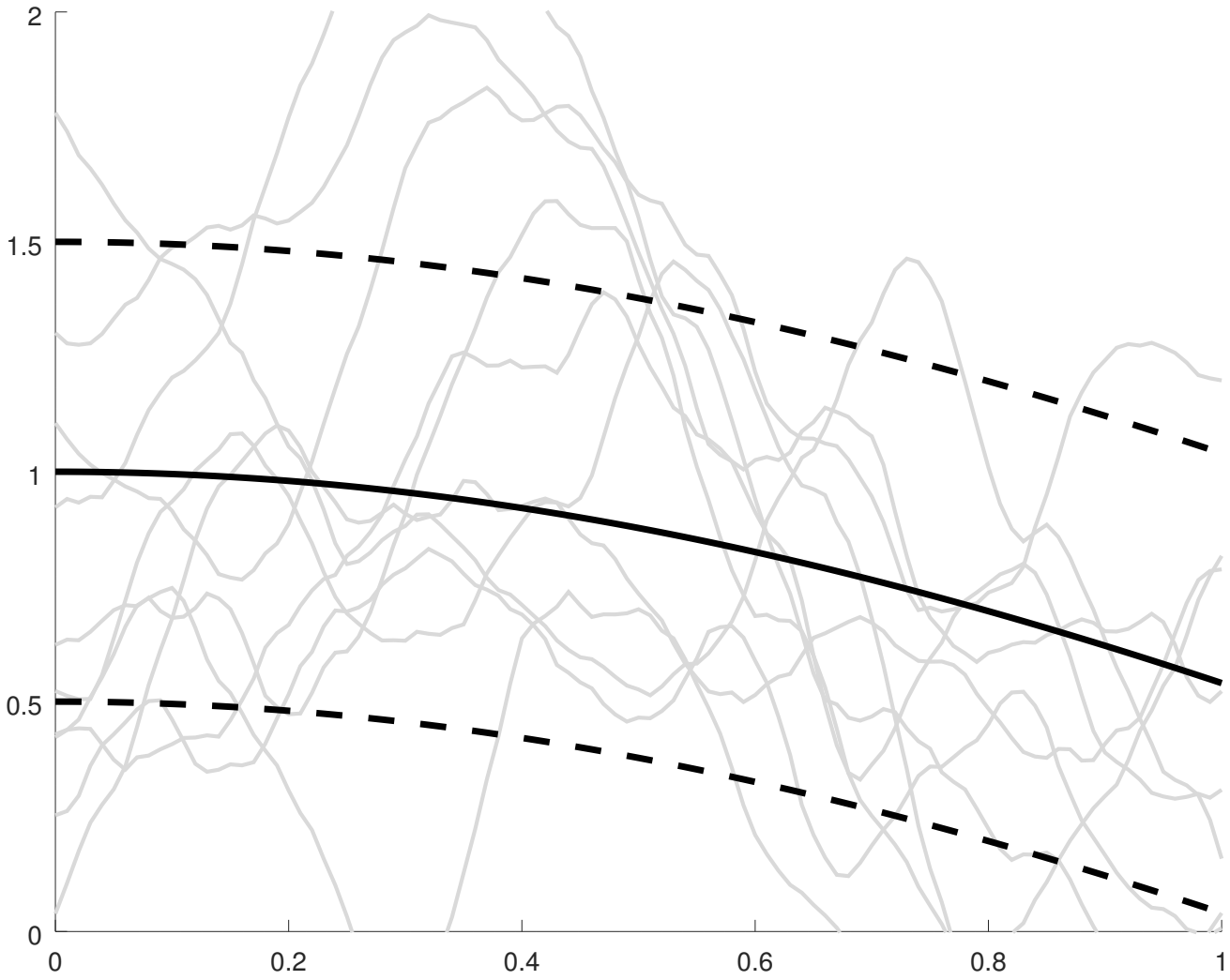
$$P_X(x)^2 = \phi(0) - K_{xX}K_{XX}^{-1}K_{Xx}$$

Bound is

$$|s(x) - f(x)| \leq P_X(x)\sqrt{\|f\|_{\mathcal{H}}^2 - \|s\|_{\mathcal{H}}^2}$$

Only thing that is hard to compute generally: $\|f\|_{\mathcal{H}}^2$.

Basic ingredient: Gaussian Processes (GPs)



Basic ingredient: Gaussian Processes (GPs)

Our favorite continuous distributions over

$\mathbb{R}$:	$\text{Normal}(\mu, \sigma^2)$,	$\mu, \sigma^2 \in \mathbb{R}$
$\mathbb{R}^n$:	$\text{Normal}(\mu, C)$,	$\mu \in \mathbb{R}^n, C \in \mathbb{R}^{n \times n}$
$\mathbb{R}^d \to \mathbb{R}$:	$\text{GP}(\mu, k)$,	$\mu : \mathbb{R}^d \to \mathbb{R}$, $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

More technically, define GPs by looking at finite sets of points:

$$\forall X = (x_1, \ldots, x_n), x_i \in \mathbb{R}^d,$$

$$\text{have } f_X \sim N(\mu_X, K_{XX}), \text{ where}$$

$$f_X \in \mathbb{R}^n, \quad (f_X)_i \equiv f(x_i)$$

$$\mu_X \in \mathbb{R}^n, \quad (\mu_X)_i \equiv \mu(x_i)$$

$$K_{XX} \in \mathbb{R}^{n \times n}, \quad (K_{XX})_{ij} \equiv k(x_i, x_j)$$

Being Bayesian

Consider a (zero-mean) GP prior with kernel k:

$$f \sim \mathrm{GP}(0, k)$$

Measure at X, apply Bayes to get posterior:

$$(f \mid f_X = y) \sim \mathrm{GP}(\mu, \tilde{k})$$

where

$$\mu(x) = k_{xX} c$$
$$\tilde{k}(x, y) = k(x, x) - k_{xX} K_{XX}^{-1} k_{Xy}$$

Specifically, posterior for $f(x)$ at given x is

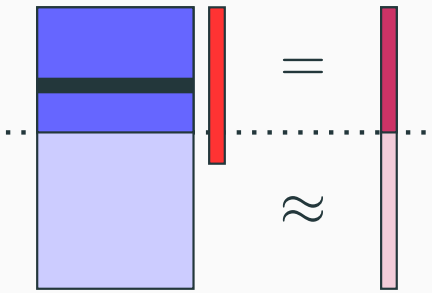
$$N(k_{xX} c, k(x, x) - k_{xX} K_{XX}^{-1} k_{Xx})$$

Predictive variance = squared power function!

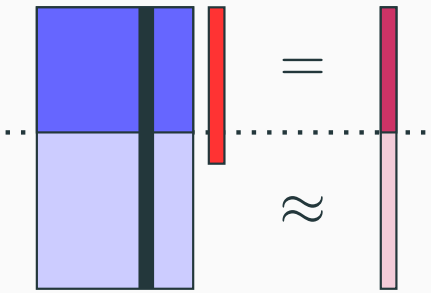
Summary and Wrap-Up

Kernel-Based Regression: Four Stories

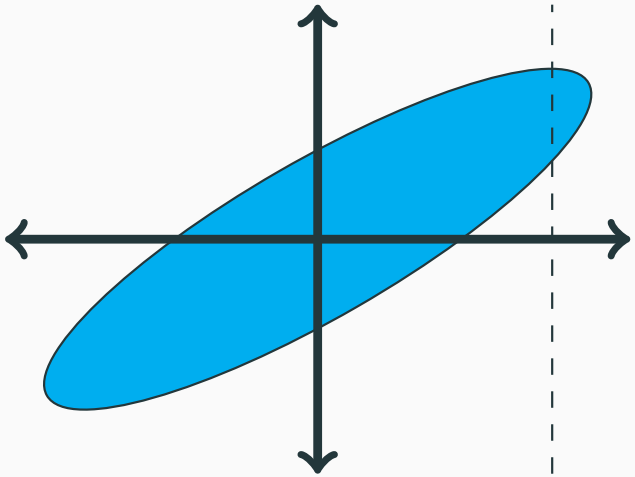
Feature map



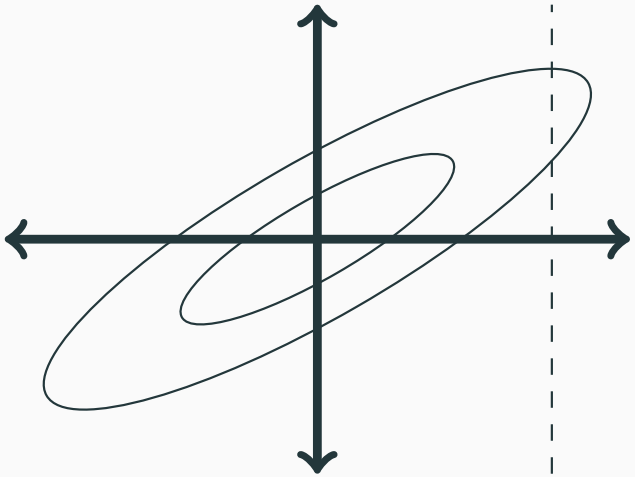
Data-dependent basis



Energy minimization



Gaussian process



The Power of Different Lenses

- "Kernel trick" used to go basis-free
  - But there is power in thinking with a basis, too!
  - Comes up as a computational tool (next time)
- Kernels can correspond to physics!
  - Ex: Cubic spline and thin-plate spline
  - Kernel as a Green's function for an elliptic PDE
  - Physical interpretation helps understand error analysis
- Optimal recovery and GP interpretation mostly coincide
  - But *only* when data is linear functionals of f
  - Ex: Different predictions for non-negativity constraints!
- CPD kernels popular in RBF literature (optimal recovery)
  - But also works for Bayesian interp — improper GP priors
  - Does appear in Wahba's work, but often overlooked
  - Tails are useful even in pos def case

Plan for Next Time

- This hour: the analysis half of numerical analysis
- Up next: the numerical side!
- Start with standard $O(n^3)$ approaches for
  - Fitting with a fixed kernel
  - Kernel selection (max likelihood and cross-validation)
  - Adaptive sampling and Bayesian optimization
- Then the fun stuff: how to scale better than $O(n^3)$!

Numerical Methods for Data Science: Scalable Kernel Methods, Part II

David Bindel

19 June 2019

Department of Computer Science
Cornell University

Kernel Approximation

Goal: Approximate $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$.
Input: (Possibly noisy) samples $y_i = f(x_i) + \epsilon_i$.

Approximation scheme: $f(x) \approx s(x)$ where

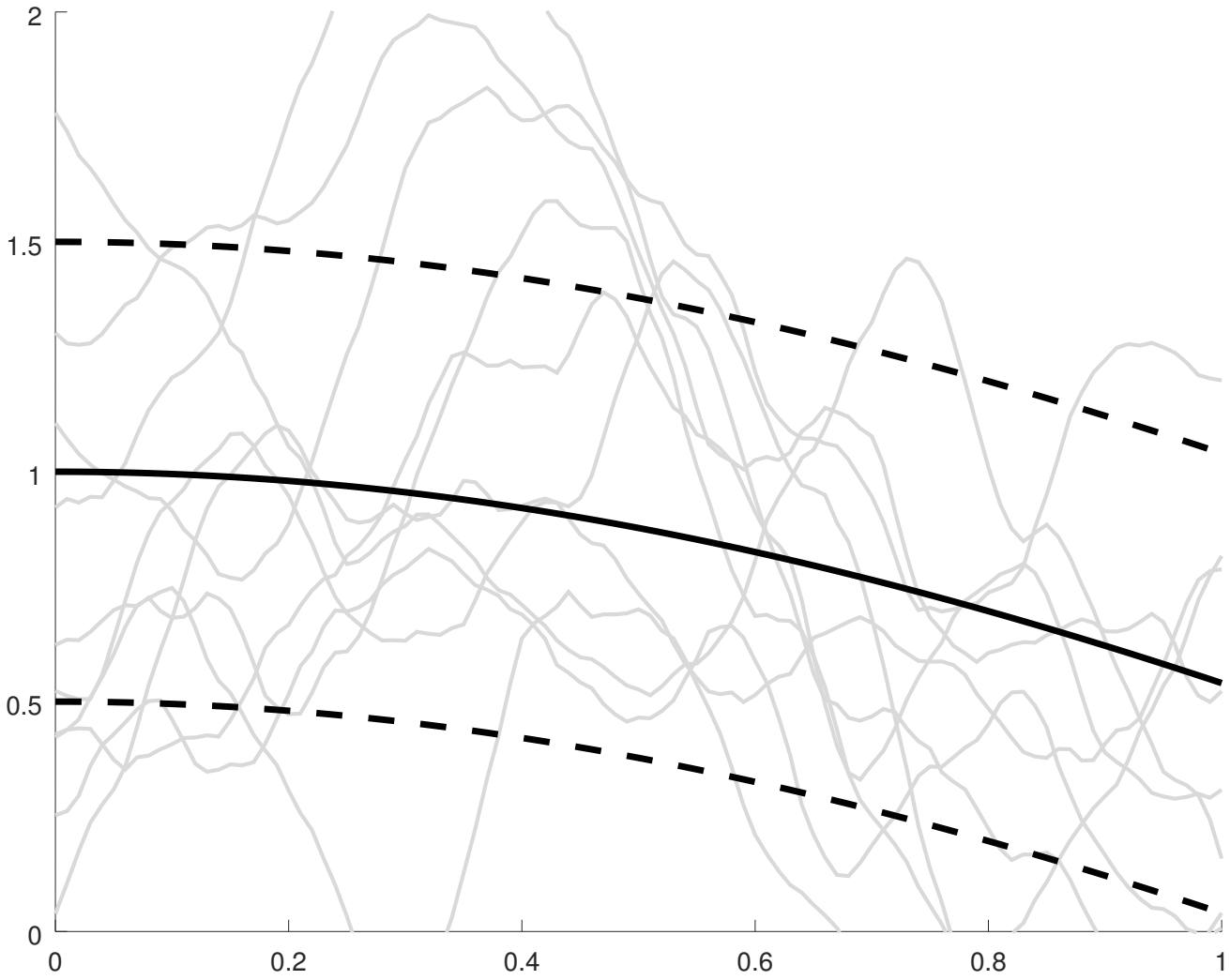
$$s(x) = \sum_{j=1}^{n} k(x, x_j) c_j + \sum_{j=1}^{m} b_j(x) \lambda_j$$

with a discrete orthogonality condition for each b_j:

$$\sum_{j=1}^{m} b_i(x_j) c_j = 0.$$

Last hour: several explanations for why this makes sense.

Picture Today: Gaussian Processes (GPs)



Gaussian Processes (GPs)

Our favorite continuous distributions over

$\mathbb{R}$:	$\text{Normal}(\mu, \sigma^2)$,	$\mu, \sigma^2 \in \mathbb{R}$
$\mathbb{R}^n$:	$\text{Normal}(\mu, C)$,	$\mu \in \mathbb{R}^n, C \in \mathbb{R}^{n \times n}$
$\mathbb{R}^d \to \mathbb{R}$:	$\text{GP}(\mu, k)$,	$\mu : \mathbb{R}^d \to \mathbb{R}$, $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

More technically, define GPs by looking at finite sets of points:

$$\forall X = (x_1, \ldots, x_n), x_i \in \mathbb{R}^d,$$

$$\text{have } f_X \sim N(\mu_X, K_{XX}), \text{ where}$$

$$f_X \in \mathbb{R}^n, \quad (f_X)_i \equiv f(x_i)$$

$$\mu_X \in \mathbb{R}^n, \quad (\mu_X)_i \equiv \mu(x_i)$$

$$K_{XX} \in \mathbb{R}^{n \times n}, \quad (K_{XX})_{ij} \equiv k(x_i, x_j)$$

When X is unambiguous, we will sometimes just write K.

Kernel Functions

Call the *kernel* (or *covariance*) function k. Required[1] property:

- **Pos def**: K_{XX} is always positive definite

Often desirable:

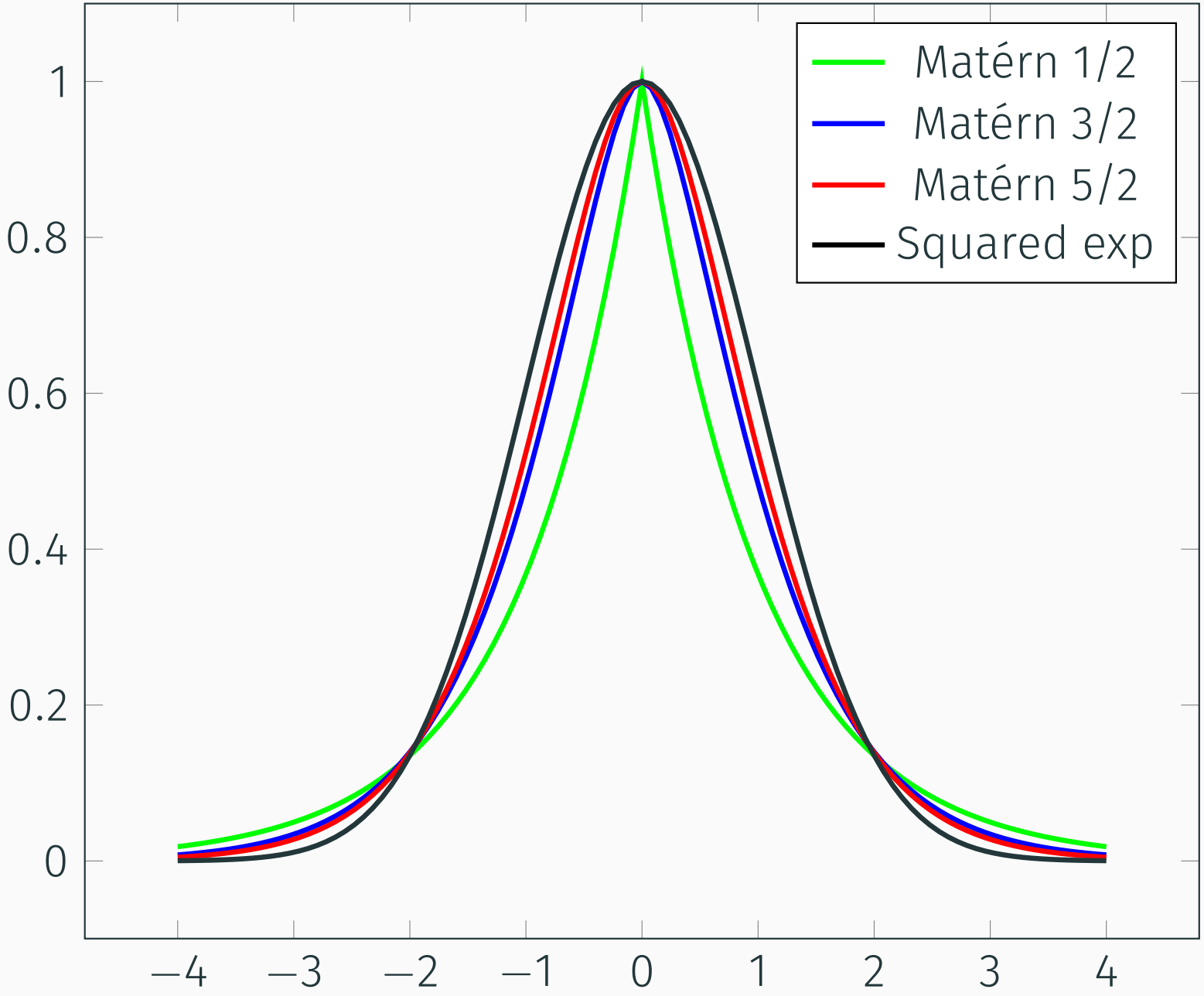
- **Stationary**: $k(x, y)$ depends only on $x - y$
- **Isotropic**: $k(x, y)$ depends only on x and $\|x - y\|$

Often want both (sloppy notation: $k = k(r)$).

Common examples (e.g. Matérn, SE) also depend on *hyper-parameters* θ — suppressed in notation unless needed.

[1] Conditionally pos def is interesting, but we'll skip it today.

Matérn and SE kernels



Choosing Kernels

Kernel is *chosen by modeler*

- Choose Matérn / SE for simplicity
- Rarely have the intuition to pick the "right" kernel
- Common choices are *universal* — can recover anything
  - Requires less data for "good" choice (inductive bias)
  - Roughly: smooth kernels good for smooth functions
- Common kernel families have *hyperparameters*
  - Length scale (almost all)
  - Smoothness parameter (Matérn)
- Also have hyperparameter for noise level ("nugget")

Would like to choose hyperparameters using data.

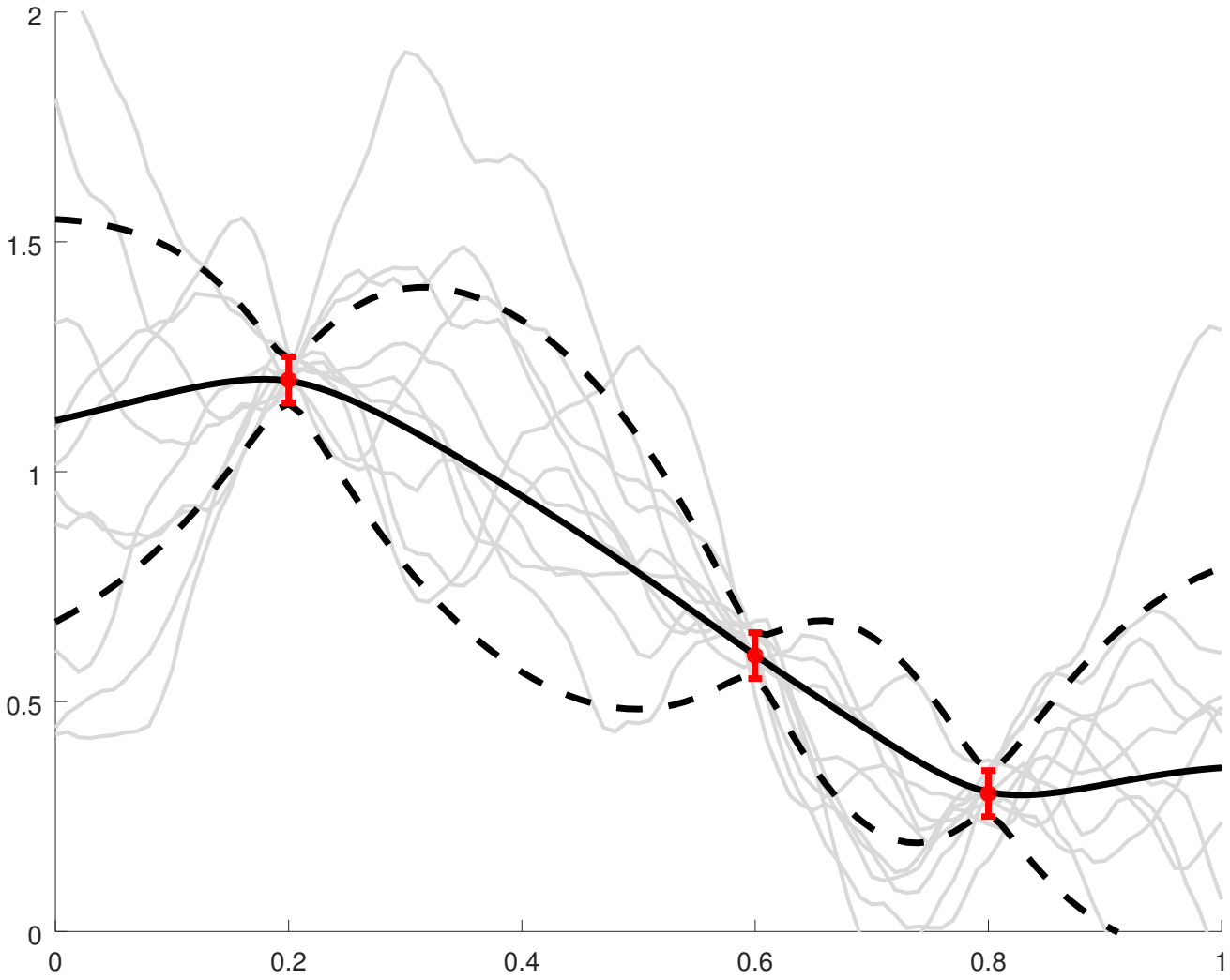
Tasks at Hand

Basic tasks in function approximation via a kernel scheme

- **Select a kernel and hyperparameters (length scale, noise)**
- **Use data to fit model parameters**
- Evaluate the model at new points
- Evaluate predictive variance at new points

For optimization: want gradients of everything

Being Bayesian



Being Bayesian

Now consider prior of $f \sim \mathrm{GP}(\mu, k)$, noisy measurements

$$f_X \sim y + \epsilon, \quad \epsilon \sim N(0, W), \qquad \text{typically } W = \sigma^2 I$$

Posterior is $f \sim \mathrm{GP}(\mu', k')$ with

$$\mu'(x) = \mu(x) + K_{xX} c \qquad \tilde{K} = K_{XX} + W$$
$$k'(x, x') = K_{xx'} - K_{xX} \tilde{K}^{-1} K_{Xx'} \qquad c = \tilde{K}^{-1}(y - \mu_X)$$

The expensive bit: solves with $\tilde{K}$.

Dealing with Derivatives

Derivative information $\implies$ multi-output GP:

$$\mu^{\nabla}(x) = \begin{bmatrix} \mu(x) \\ \partial_x \mu(x) \end{bmatrix}, \quad k^{\nabla}(x, x') = \begin{bmatrix} k(x, x') & \nabla'_x k(x, x') \\ \partial_x k(x, x') & \partial^2_{x,x'} k(x, x') \end{bmatrix}$$

Requires a little care, but ideas in this talk generalize.

NB: Other (linear) measurements also allowed – e.g. integrals.

Kernel Hyper-parameters: MLE

How to estimate *hyper-parameters* θ?

- Bayesian approach? Expensive...
- Usually just do maximum likelihood estimation (MLE)

Likelihood function is same as for a multivariate normal:

$$\ell(\theta|y) = \frac{1}{\sqrt{\det(2\pi\tilde{K})}} \exp\left(-\frac{1}{2}(y-\mu_X)^T\tilde{K}^{-1}(y-\mu_X)\right).$$

Of course, we usually work with *log-likelihood* and derivatives.

Kernel Hyper-parameters: MLE

How to estimate *hyper-parameters* θ?

- Bayesian approach? Expensive...
- Usually just do maximum likelihood estimation (MLE)

Log-likelihood function for kernel hypers θ

$$\mathcal{L}(\theta|y) = \mathcal{L}_y + \mathcal{L}_{|K|} - \frac{n}{2}\log(2\pi)$$

where (again with $c = \tilde{K}^{-1}(y - \mu_X)$)

$$\mathcal{L}_y = -\frac{1}{2}(y - \mu_X)^T c, \qquad \frac{\partial \mathcal{L}_y}{\partial \theta_i} = \frac{1}{2} c^T \left(\frac{\partial \tilde{K}}{\partial \theta_i}\right) c$$

$$\mathcal{L}_{|K|} = -\frac{1}{2}\log\det \tilde{K}, \qquad \frac{\partial \mathcal{L}_{|K|}}{\partial \theta_i} = -\frac{1}{2}\mathrm{tr}\left(\tilde{K}^{-1}\frac{\partial \tilde{K}}{\partial \theta_i}\right)$$

Kernel Hyper-parameters: LOOCV

Alternative to MLE: Leave One Out Cross-Validation (LOOCV)

$$LOOCV = \sum_{i=1}^{n} \left(r^{(-i)} \right)^2$$

where

$$r^{(-i)} = y_i - s^{(-i)}(x)$$
$$s^{(-i)} = \text{fit to all data points but } x_i$$

Kernel Hyper-parameters: LOOCV Trick

Fitting problem for all the data:

$$\tilde{K}c = y - \mu_X$$

Leave out point i: $c_i^{(-i)} = 0$ and $r^{(-i)} = y_i - \mu(x_i) - K_{x_i X} c^{(-i)}$.
Write as

$$\begin{bmatrix} \tilde{K} & e_i \\ e_i^T & 0 \end{bmatrix} \begin{bmatrix} c^{(-i)} \\ r^{(-i)} \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

where e_i denotes column i of an identity matrix.

Gaussian elimination gives

$$r^{(-i)} = \frac{c_i}{\left[\tilde{K}^{-1}\right]_{ii}}$$

NB: $\left[\tilde{K}^{-1}\right]_{ii}$ is the leave one out predictive variance.

Generalized Cross-Validation

Problem with LOOCV: no weights on residuals!

Alternative is Generalized Cross-Validation (Golub, Heath, Wahba 79):

$$V(\lambda) = \frac{\frac{1}{n}\|(I - A(\lambda))(y - \mu_X)\|^2}{\left[\frac{1}{n}\mathrm{tr}(I - A(\lambda))\right]^2}$$

where $A(\lambda)$ is the influence matrix.

In case of $\tilde{K} = K + \sigma^2 I$, have

$$I - A(\lambda) = I - K\tilde{K}^{-1} = \sigma^2 \tilde{K}^{-1}$$

Therefore

$$V = n\,\frac{\|c\|^2}{\mathrm{tr}(\tilde{K}^{-1})^2}$$

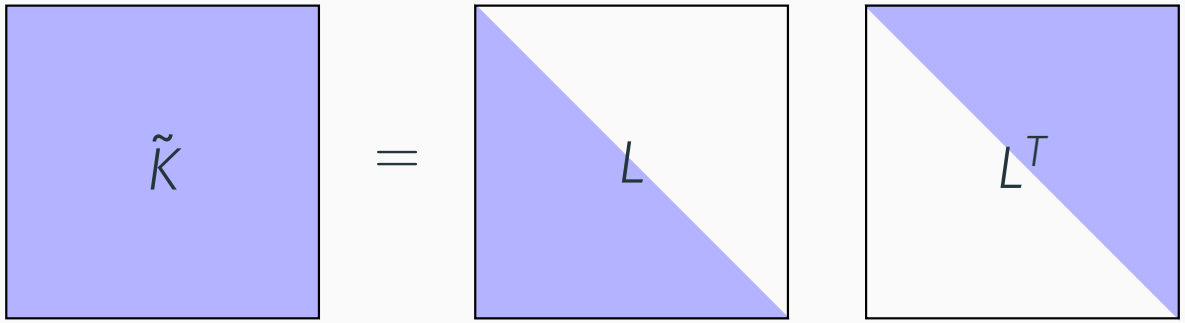
GCV Derivatives

Derivative of $V = n\|c\|^2/\mathrm{tr}\left(\tilde{K}^{-1}\right)^2$ is tedious, not difficult:

$$\delta V = n\,\frac{\|c\|^2\,\mathrm{tr}\left(\tilde{K}^{-1}(\delta\tilde{K})\tilde{K}^{-1}\right) - \left(c^T\tilde{K}^{-1}(\delta\tilde{K})c\right)\,\mathrm{tr}\left(\tilde{K}^{-1}\right)}{\mathrm{tr}\left(\tilde{K}^{-1}\right)^3}$$

Computational issues for GCV are similar to those for MLE. Focus on latter.

Learning Parameters and Hypers: Small n



In an optimization loop:

```
L = chol(Ktilde(theta));
c = L'\(L\(y-mu));
liky  = -0.5*((y-mu)'*c);
likK  = -sum(log(diag(L)));
for i = 1:len(theta)
   dliky = 0.5*(c'*dK(theta,i)*c);
   dlikK = -0.5*trace(L'\(L\dk(theta,i)));
end
```

Scalability Bottlenecks

Consider n data points

- Straightforward regression: factor $\tilde{K}$ at $O(n^3)$ cost
- Kernel hyper MLE requires multiple $O(n^3)$ ops
  - To compute $\log\det \tilde{K}$ is $O(n^3)$ per step
  - To compute $\mathrm{tr}\left(\tilde{K}^{-1}\frac{\partial \tilde{K}}{\partial \theta_i}\right)$ is $O(n^3)$ per hyper per step
- Cost of kernel hyper GCV is similar (lots of costly traces)

Two possible work-arounds

- Data-sparse factorization methods
- Methods that avoid factorization (e.g. iterative solvers)
  - Q: how to handle determinants and traces?

Scaling GPs

Several possible approaches:

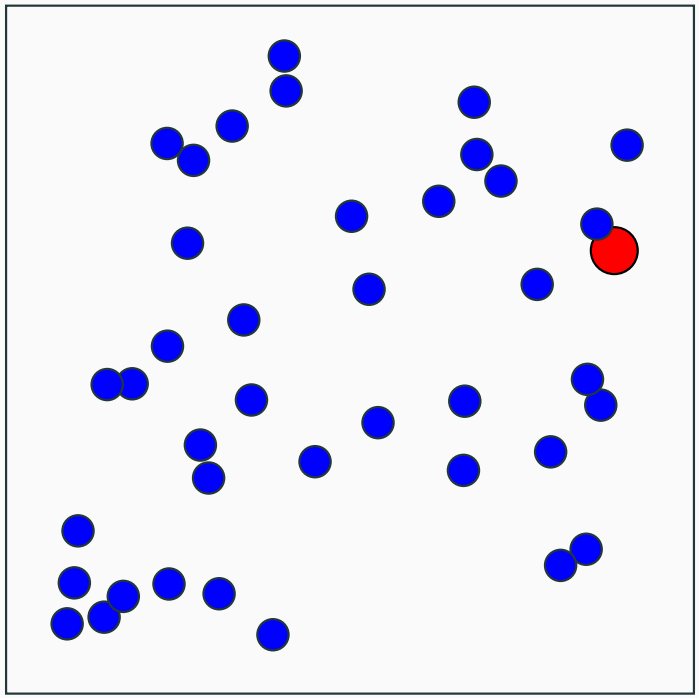
- Very smooth kernels $\implies$ low rank K
- Compactly supported kernels $\implies$ sparse K
- Can piece together local kernel approximations
- Rank-structured factorization methods
- Black-box multiplication (via FMM or FFT)

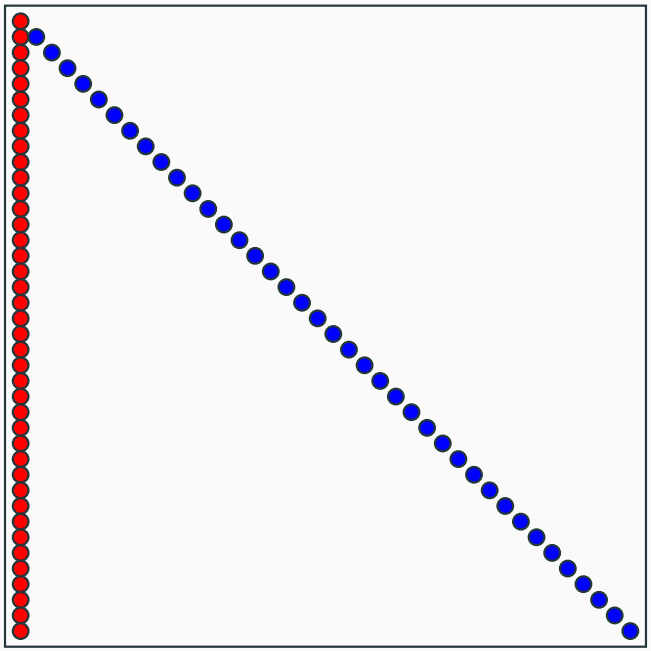
We will focus on first and last.

Scaling GPs: Factorization approach

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

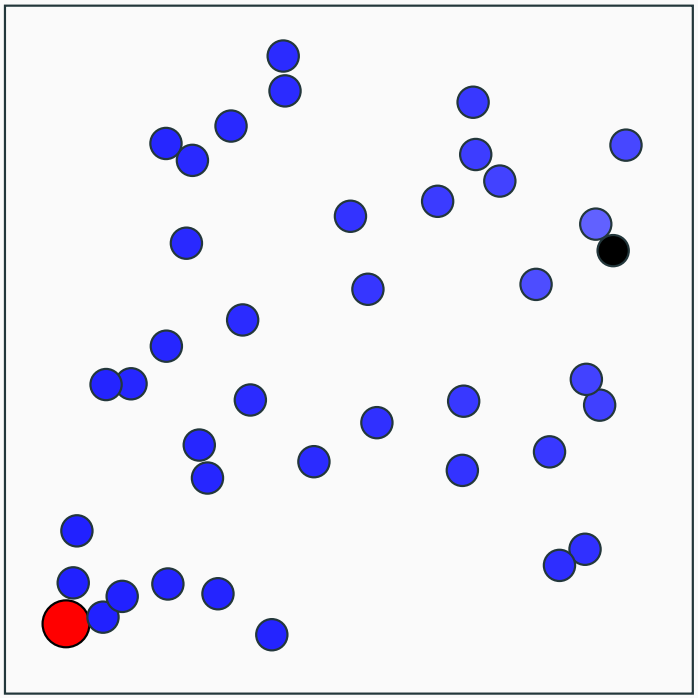


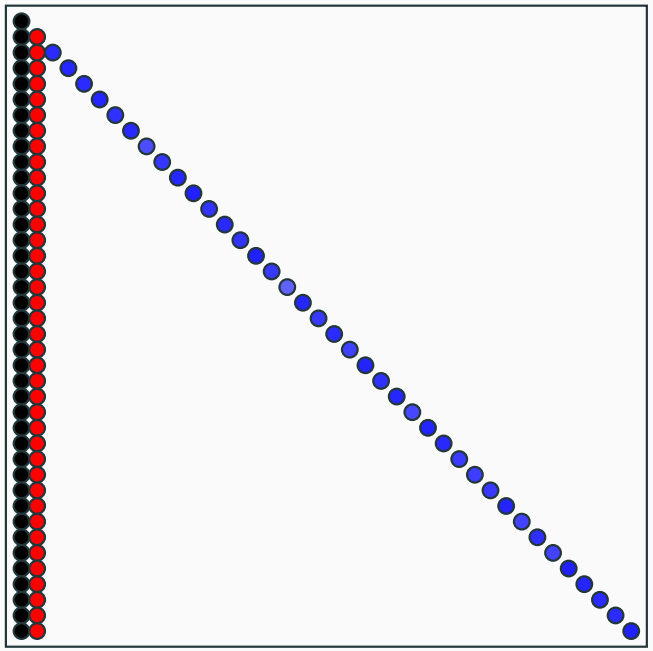


Diagonal element: 1.00e+00

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

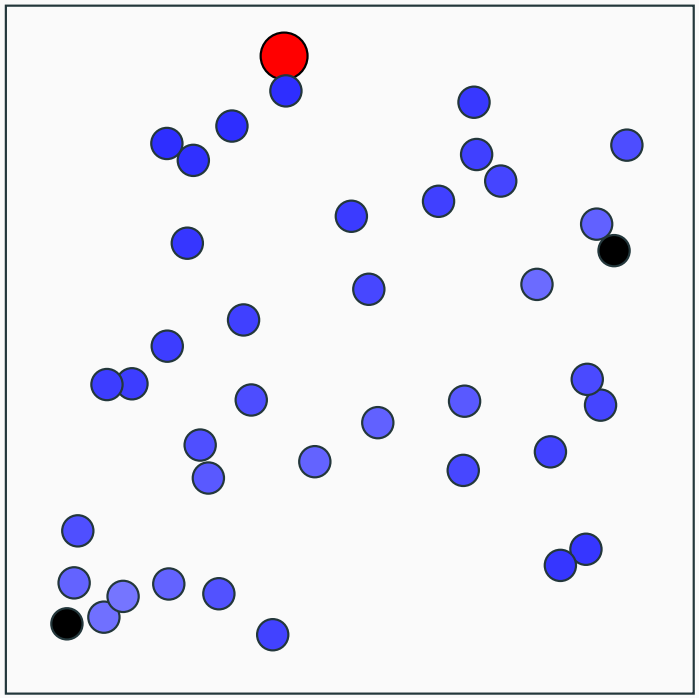


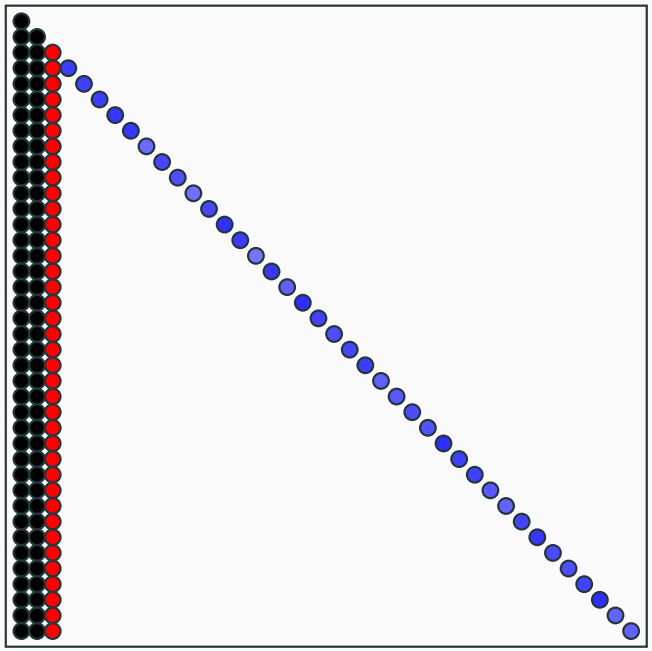


Diagonal element: 6.77e-02

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

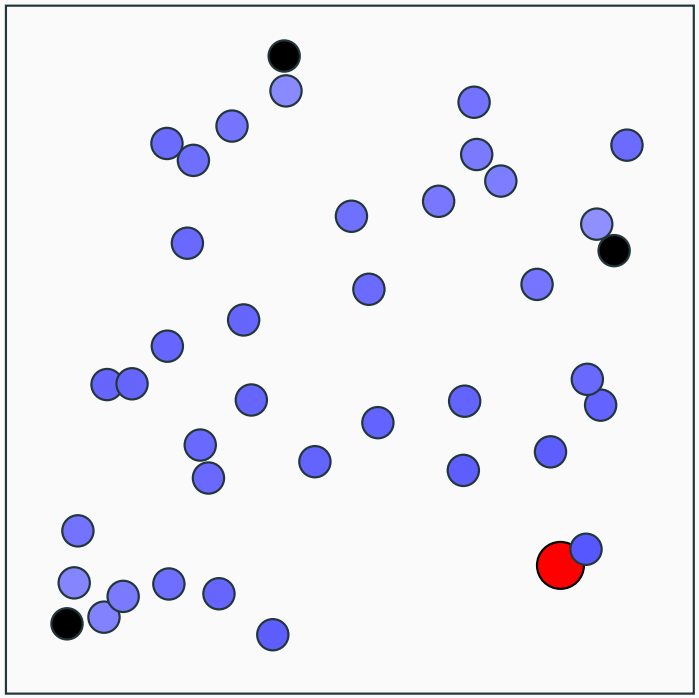


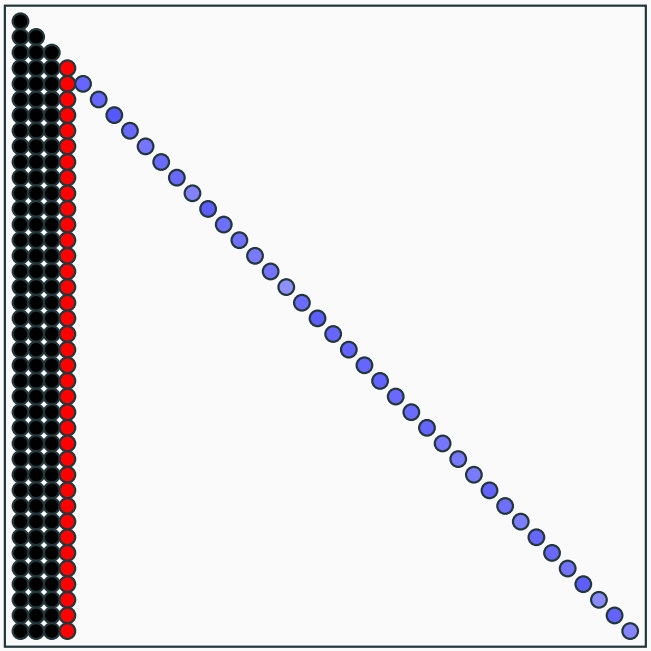


Diagonal element: 1.91e-02

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

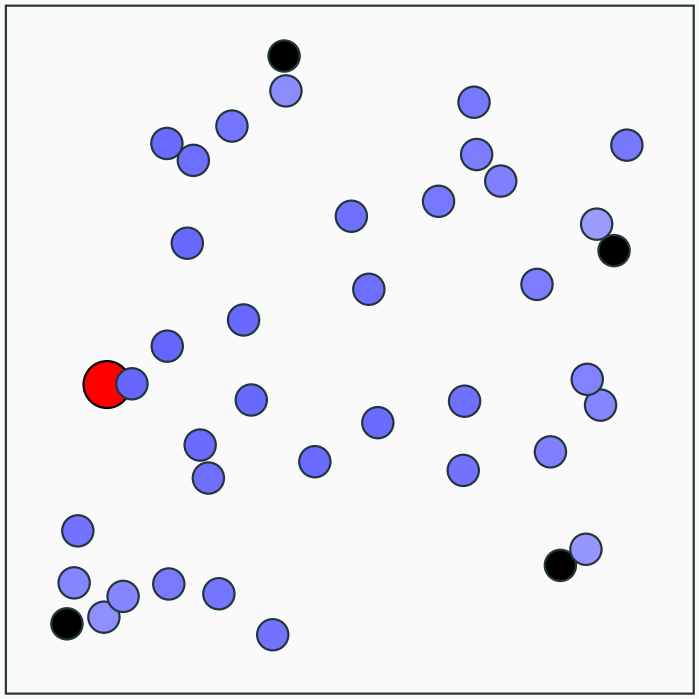


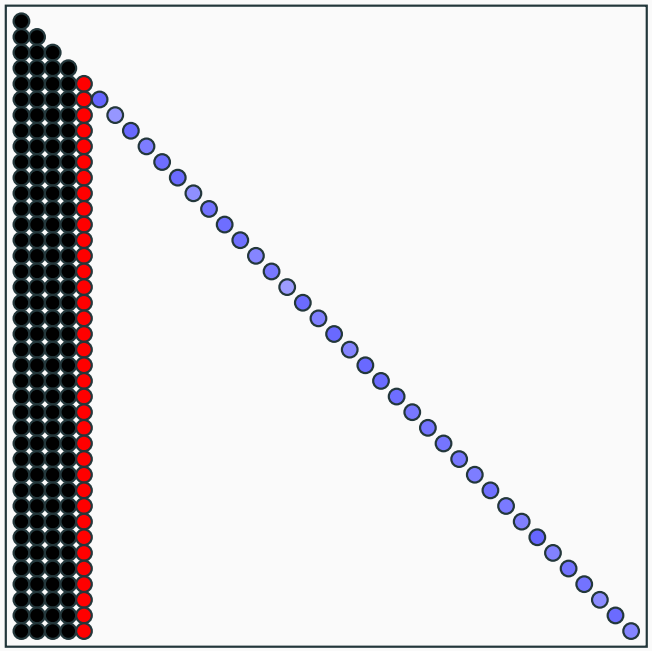


Diagonal element: 5.11e-04

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

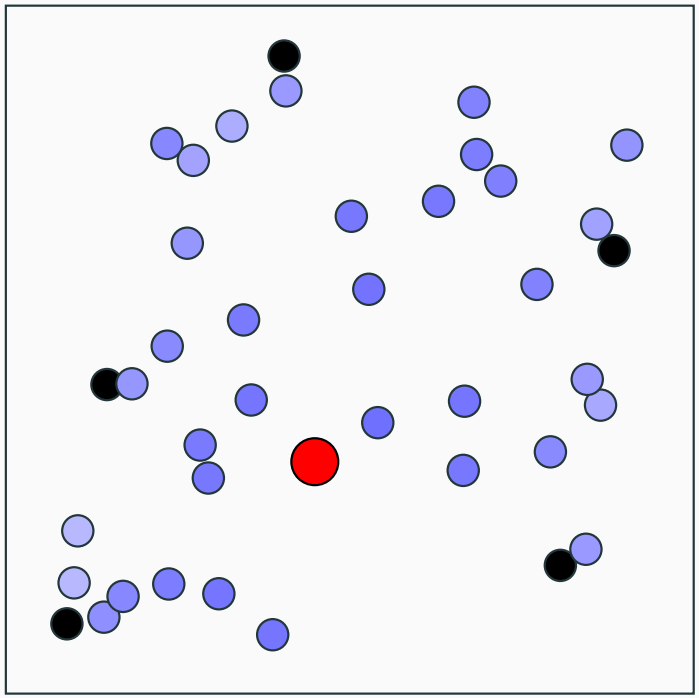


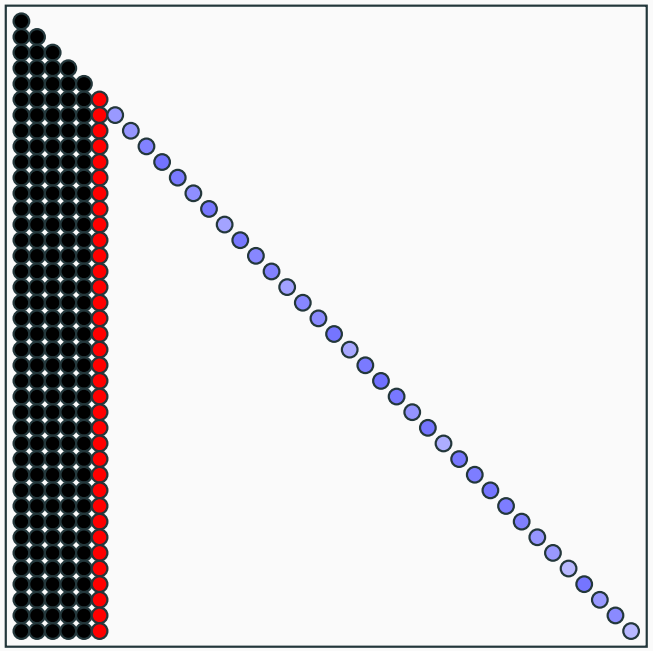


Diagonal element: 1.19e-04

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

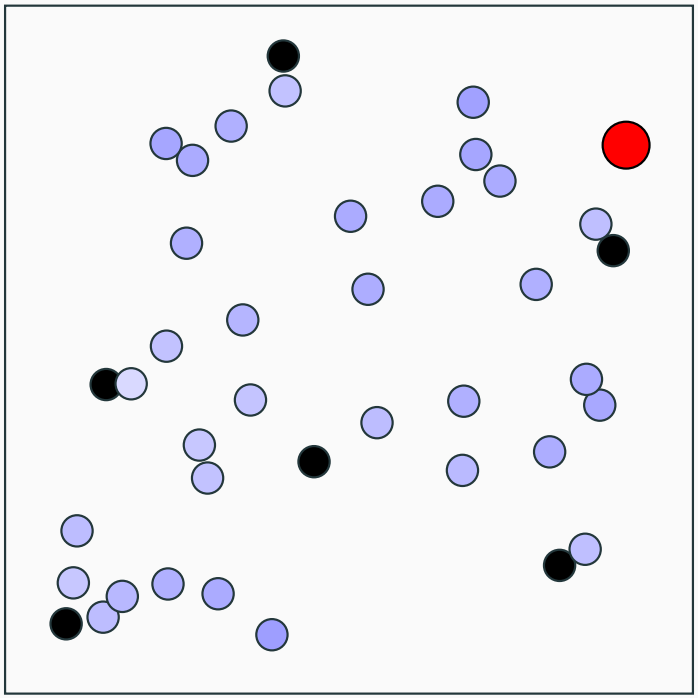


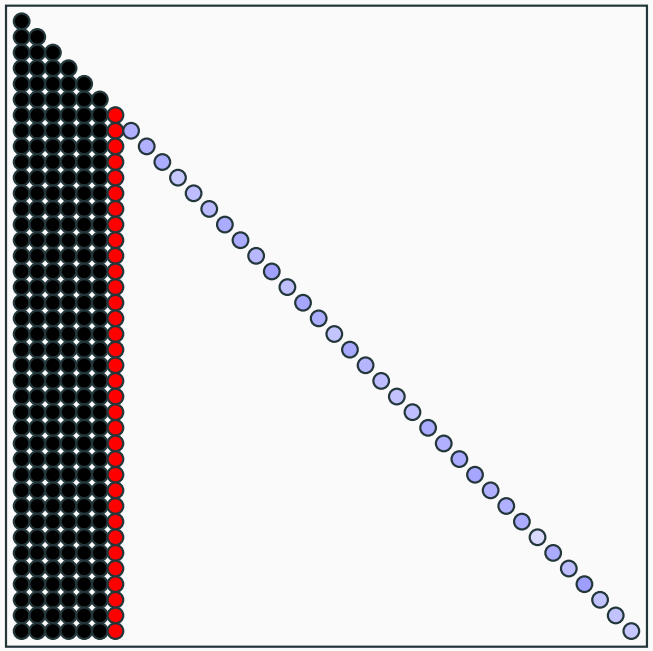


Diagonal element: 4.18e-05

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

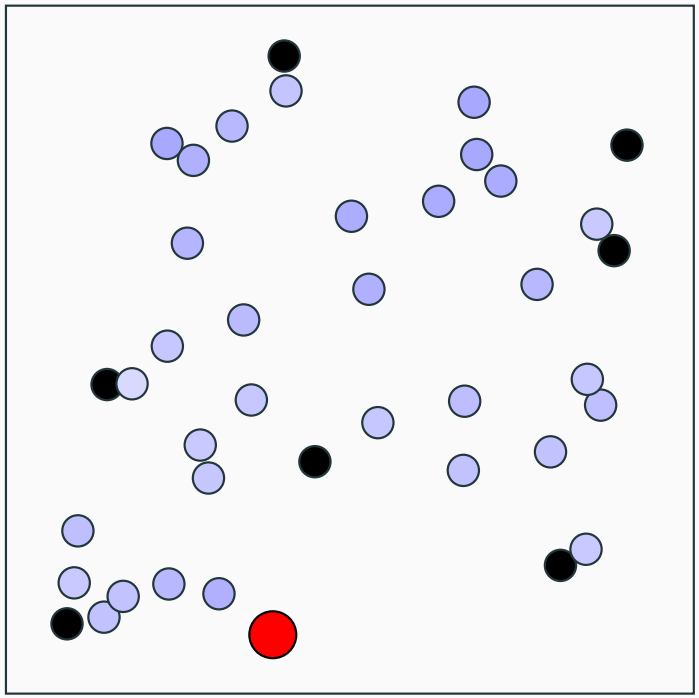


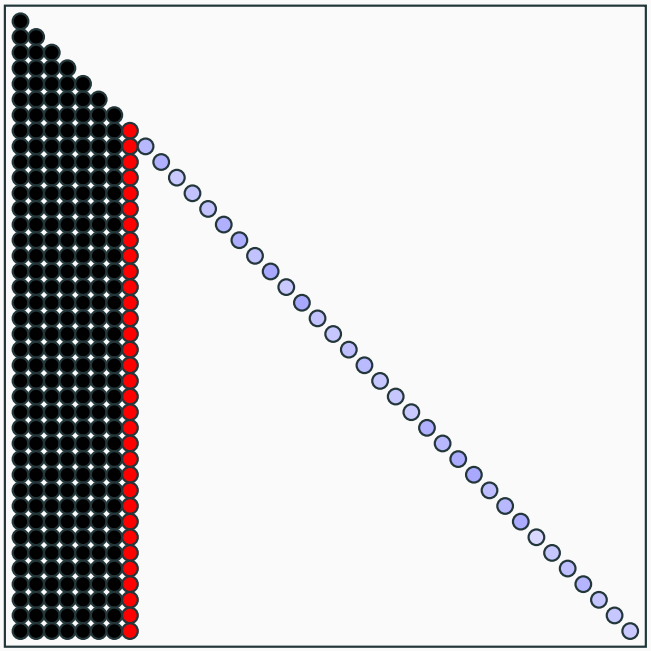


Diagonal element: 8.54e-07

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*

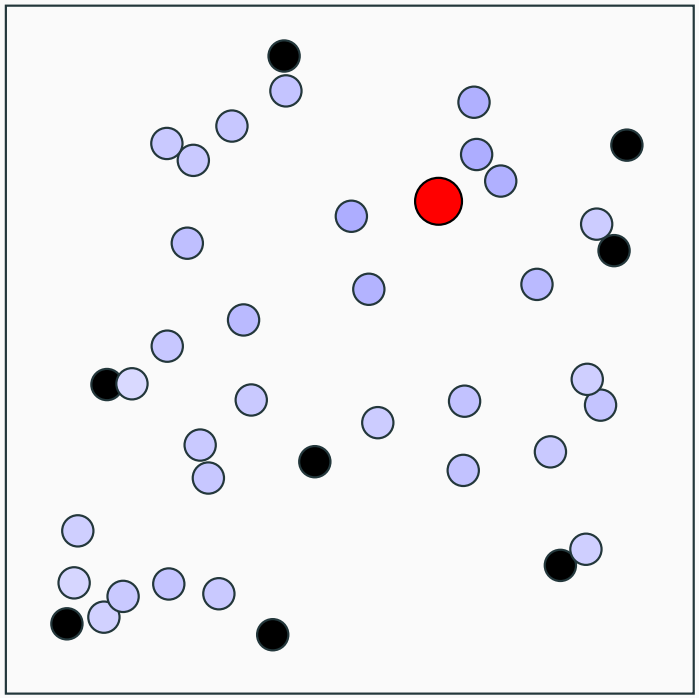


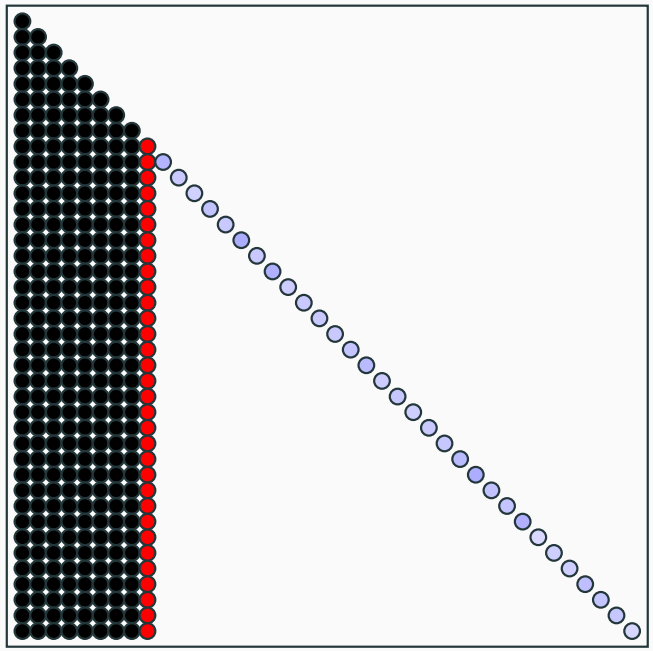


Diagonal element: 3.58e-07

Simplest Data-Sparse Approach

For K (nearly) low rank: *partial pivoted Cholesky*





Diagonal element: 1.92e-07

Simplest Data-Sparse Approach

Smooth kernel and long length scales: K nearly rank $m \ll n$

- $P(LL^T)P^T =$ partial pivoted Cholesky (select m *inducing points*); does *not* require forming K_{XX}
- Solve $P(LL^T + \sigma^2 I)P^T c = f_X$ stably by reformulating c as a scaled regularized least squares residual:

$$\text{minimize } \|f_X - Lw\|^2 + \sigma^2 \|w\|^2, \quad c = \sigma^{-2}(f_X - Lw)$$

- Compute $\log\det \tilde{K} = \log\det(L^T L + \sigma^2 I) + 2(n-m)\log\sigma$; similar cheap rearrangement for derivatives.
- Prediction and predictive variance are also cheap.

Notes on Pivoted Cholesky

- Don't have to be purely greedy (swaps as in CUR/ID)
- Can make it depend on data (LARS or Lasso approaches)
- Piv Chol good for *preconditioning* black-box approaches

Beyond Low Rank

If K is not low rank, can still use *rank-structured* factorization.

Example (1D):

$$\tilde{K} \approx \begin{bmatrix} \tilde{K}_{11} & U_1 U_2^T \\ U_2 U_1^T & \tilde{K}_{22} \end{bmatrix}$$

Can solve be recognizing "disguised sparsity":

$$\tilde{K}c = y \quad \Longrightarrow \quad \begin{bmatrix} \tilde{K}_{11} & 0 & U_1 & 0 \\ 0 & \tilde{K}_{22} & 0 & U_2 \\ U_1 & 0 & 0 & -I \\ 0 & U_2 & -I & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ 0 \\ 0 \end{bmatrix}$$

Apply recursively for an $O(n \log n)$ solver. More sophisticated options available, extensions to multi-dimensional, etc.

Scaling GPs: Black box approach

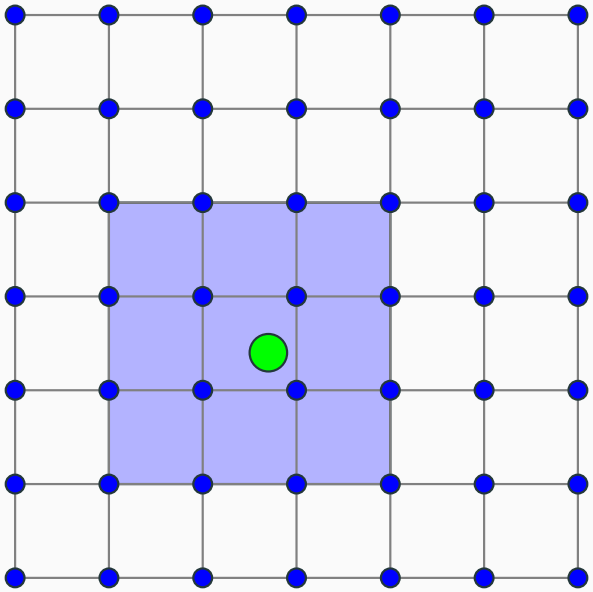
Basic ingredients for “black box” approach

- Fast MVMs with kernel matrices
- Krylov methods for linear solves and matrix functions
- Stochastic estimators: trace, diagonal, and other

Kernel approximations

- Low-rank approximation (via *inducing variables*)
  - Non-smooth kernels, small length scales $\implies$ large rank
  - Only semi-definite
- Sparse approximation
  - OK with SE kernels and short length scales
  - Less good with heavy tails or long length scales
  - May again lose definiteness
- More sophisticated: fast multipole, Fourier transforms
  - Same picture as in integral eq world (FMM, PFFT)
  - Main restriction: low dimensional spaces (2-3D)
- Kernel a model choice — how does approx affect results?

Example: Structured Kernel Interpolation (SKI)



Write $K_{XX} \approx W^T K_{UU} W$ where

- U is a uniform mesh of m points
- K_{UU} has Toeplitz or block Toeplitz structure
- Sparse W interpolates values from X to U

Apply K_{UU} via FFTs in $O(m \log m)$ time.

The power of fast MVMs

With MVMs alone, natural to explore nested *Krylov subspaces*:

$$\mathcal{K}_{d+1}(\tilde{K}, b) = \mathbf{span}\{b, \tilde{K}b, \tilde{K}^2 b, \ldots, \tilde{K}^d b\} = \{p(\tilde{K})b : p \in \mathcal{P}_k\}$$

Lanczos process: *expansion + Gram-Schmidt*

$$\beta_j q_{j+1} = \tilde{K} q_j - \alpha_j q_j - \beta_{j-1} q_{j-1}$$

Lanczos factorization: $\tilde{K} Q_k = Q_k \bar{T}_k$ where

$$Q_k = \begin{bmatrix} q_1 & q_2 & \ldots & q_k \end{bmatrix},$$

$$\bar{T}_k = \left[\begin{array}{ccccc} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \beta_3 & \\ & & \ddots & \ddots & \ddots \\ & & & \beta_{k-1} & \alpha_k \\ \hline & & & & \beta_k \end{array}\right] = \left[\begin{array}{c} T_k \\ \hline \beta_k e_k^T \end{array}\right]$$

The power of fast MVMs

Fast MVM with symmetric $\tilde{K} \implies$ try Lanczos!

- Incrementally computes $\tilde{K}Q = QT$ where
  - Q has orthonormal columns
  - Leading k columns span k-dim Krylov space
  - T is tridiagonal
- Building block for
  - Solving linear systems (CG)
  - Approximating eigenvalues
  - Approximating matrix functions: $f(\tilde{K})b$
  - Quadrature vs spectral measure for $\tilde{K}$
- Fast (three-term recurrence) and elegant...
- ... but not forward stable in finite precision

Function application via Lanczos

A computational kernel: $f(\tilde{K})b$

- Run Lanczos from starting vector $b/\|b\|$
- At n steps in exact arithmetic,

$$f(\tilde{K})b = Qf(T)Q^T b = \|b\|Qf(T)e_1$$

- Truncate at $k \ll n$ steps, use

$$f(\tilde{K})b \approx \|b\|Q_k f(T_k)e_1$$

- Error analysis hinges on quality of poly approx

$$\min_{f \in P_k} \max_{\lambda \in \Lambda(\tilde{K})} |f(\lambda) - \hat{f}(\lambda)|$$

- Compare: Chebyshev methods just use $[\lambda_{\min}, \lambda_{\max}]$

CG is a special case corresponding to $f(z) = z^{-1}$.

Tractable traces

CG solves systems with $\tilde{K}$; problem terms are

$$\mathcal{L}_{|K|} = -\frac{1}{2}\mathrm{tr}\left(\log \tilde{K}\right) \qquad \frac{\partial \mathcal{L}_{|K|}}{\partial \theta_i} = -\frac{1}{2}\mathrm{tr}\left(\tilde{K}^{-1}\frac{\partial \tilde{K}}{\partial \theta_i}\right)$$

Q: How do we parley fast MVMs into trace computations?

Tractable traces

Stochastic trace estimation trick:

- $z \in \mathbb{R}^n$ has independent random entries
- $\mathbb{E}[z_i] = 0$ and $\mathbb{E}[z_i^2] = 1$

Then

$$\mathbb{E}[z^T A z] = \sum_{i,j} a_{ij} \mathbb{E}[z_i z_j] = \mathrm{tr}(A).$$

NB: $\mathbb{E}[z \odot Az] = \mathbf{diag}(A)$.

Standard choices for the probe vector z:

- Hutchinson: $z_i = \pm 1$ with probability 0.5
- Gaussian: $z_i \sim N(0, 1)$

See Avron and Toledo review, JACM 2011.

Putting it together

For each probe vector z until error bars small enough:

- Run Lanczos from $z/\|z\|$
- Use Lanczos to estimate $\tilde{K}^{-1}z$ and $\log(\tilde{K})z$
- Dot products yield estimators:

$$\mathcal{L}_{|K|} = -\frac{1}{2}\mathbb{E}\left[z^T \log(\tilde{K})z\right]$$

$$\frac{\partial \mathcal{L}_{|K|}}{\partial \theta_i} = -\frac{1}{2}\mathbb{E}\left[(\tilde{K}^{-1}z)^T \left(\frac{\partial \tilde{K}}{\partial \theta_i} z\right)\right]$$

Cost per probe:

- One Lanczos process
- One matvec per parameter with derivative

Quite effective in practice! And amenable to preconditioning.

“There is No New Thing Under the Sun”

“Generalized Cross-Validation for Large-Scale Problems”
Golub and Von Matt (1997)

- Treats least squares and GCV (vs kernel methods and MLE)
- But the same Lanczos + stochastic trace estimator combo

Pivoted Cholesky preconditioning

Let $M = P(LL^T + \sigma^t I)P^T \approx \tilde{K}$ with $L \in \mathbb{R}^{n \times m}$, $m \ll n$:

- *Preconditioned* CG: works (implicitly) with $M^{-1}\tilde{K}$
- Note $\log\det \tilde{K} = \log\det M + \log\det M^{-1}\tilde{K}$
- Know how to do fast direct solves and log det with M
- All boils down to generalized Lanczos with $(\tilde{K}, M)$
- Smooth kernels (e.g. SE) and long length scales $\implies$ convergence in few steps

Blocking for performance

Generalized Lanczos per probe vector involves

- One matvec with $\tilde{K}$ per step
- One solve with M per step
- Barrier between steps
- Low *arithmetic intensity* (flops / memory access)
- Limited opportunities for parallelism

Idea: Lanczos for several probes in parallel

- Multiply $\tilde{K}$ or M^{-1} by *panel* of vectors / step
- Improves cache use and parallelism

The whole package

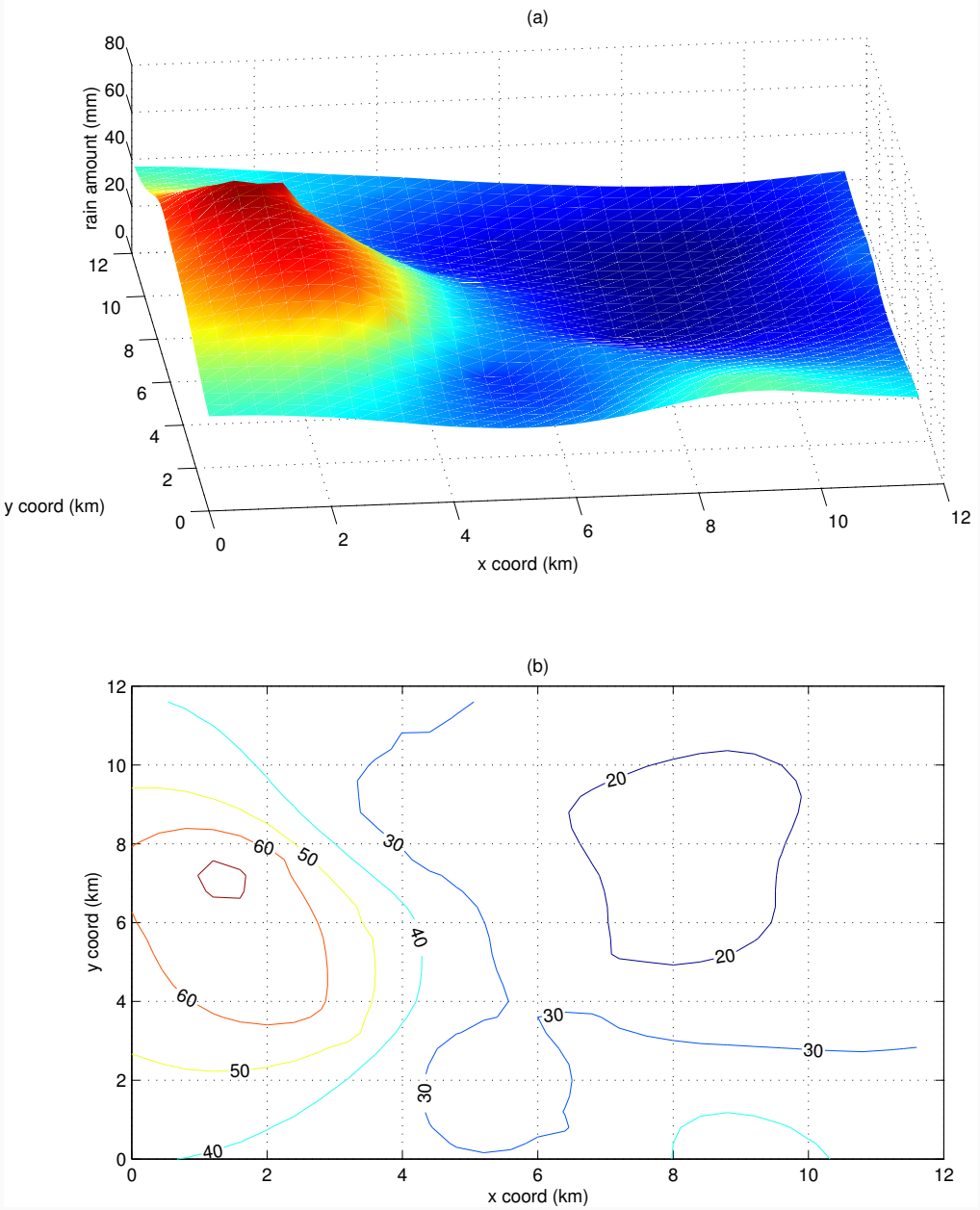
So we have

- Stochastic estimators + Krylov iterations
- Preconditioning to reduce steps to convergence
- Blocking to reduce time per step
- GPU acceleration speeds things up further

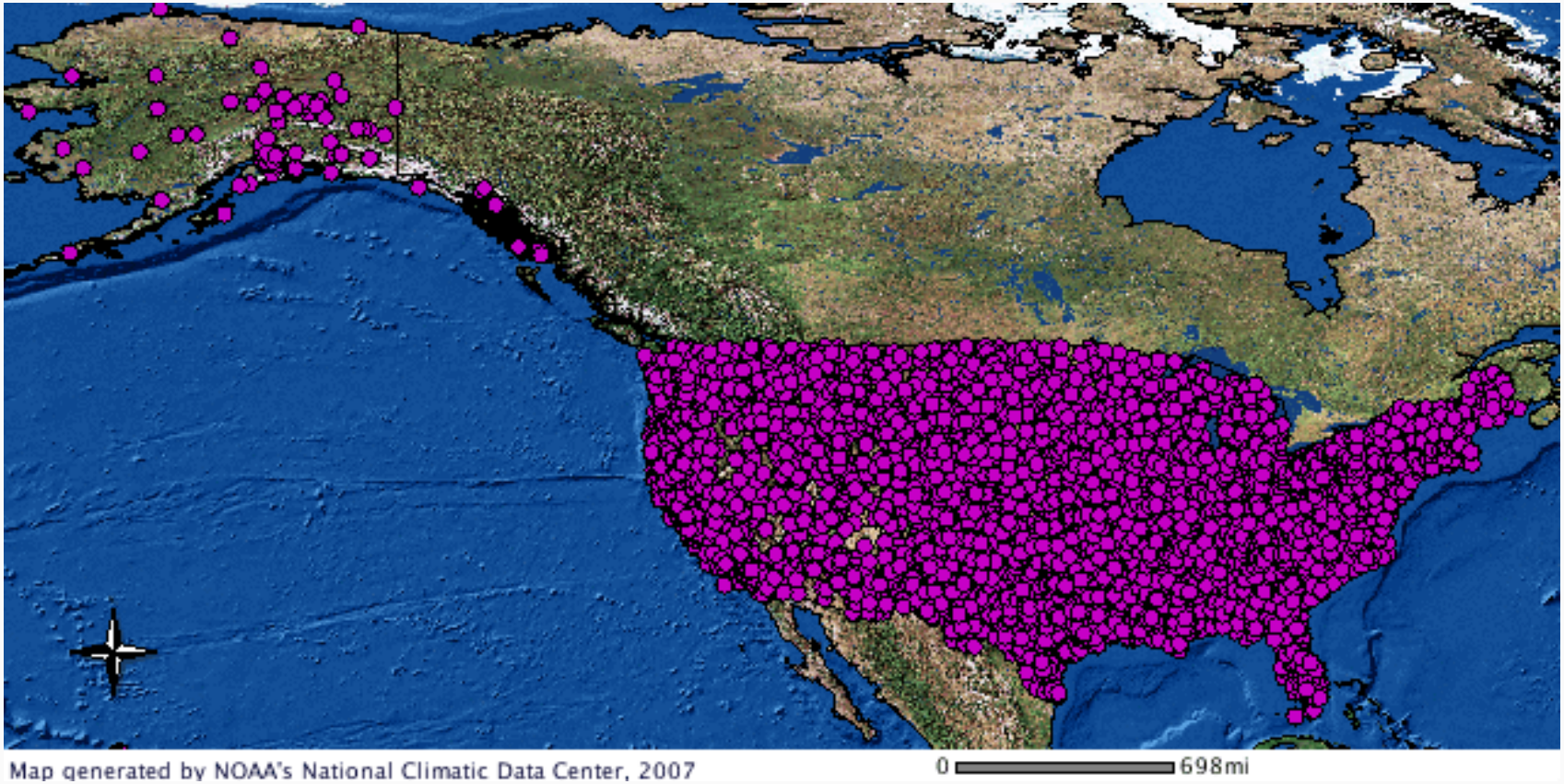
For all the tricks together: `https://gpytorch.ai`

Examples

Example: Rainfall



Example: Rainfall



Example: Rainfall

Method	n	m	MSE	Time [min]
Lanczos	528k	3M	0.613	14.3
Scaled eigenvalues	528k	3M	0.621	15.9
Exact	12k	-	0.903	11.8

- Data: Hourly precipitation data at 5500 weather stations
- Aggregate into daily precipitation
- Total data: $628K$ entries
- Train on $100K$ data points, test on remainder
- Use SKI with 100 points per spatial dim, 300 in time
- Comparison: scaled eigenvalues approx, exact solve

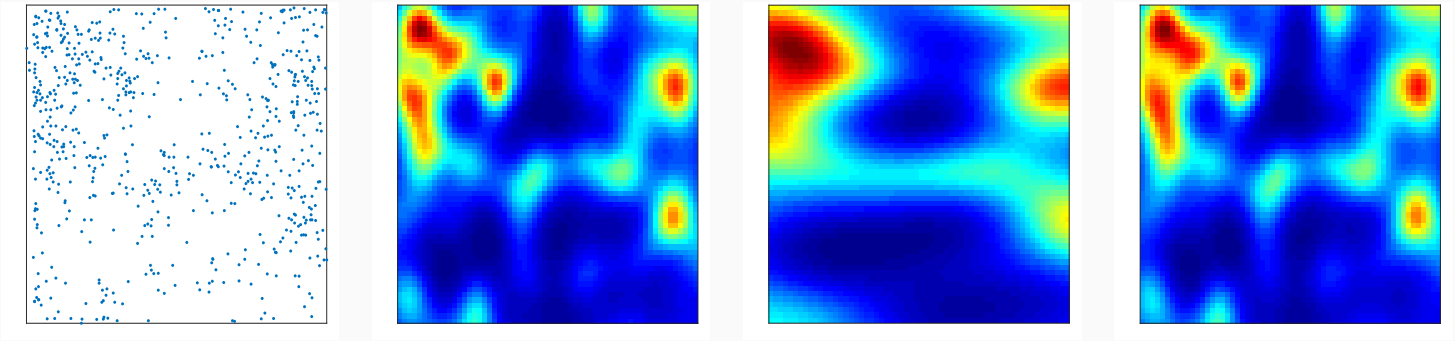
NB: This is with an older MATLAB code (GPML), not GPyTorch

Example: Hickory data

Can build other stochastic processes via GPs

- Example: Log-Gaussian Cox process model
  - Models count data (e.g. events in spatial bins)
  - Poisson conditional on intensity function
  - Log intensity drawn from a GP
- Laplace approximation for posterior
- Data set is point pattern of 703 hickory trees in Michigan

Example: Hickory data



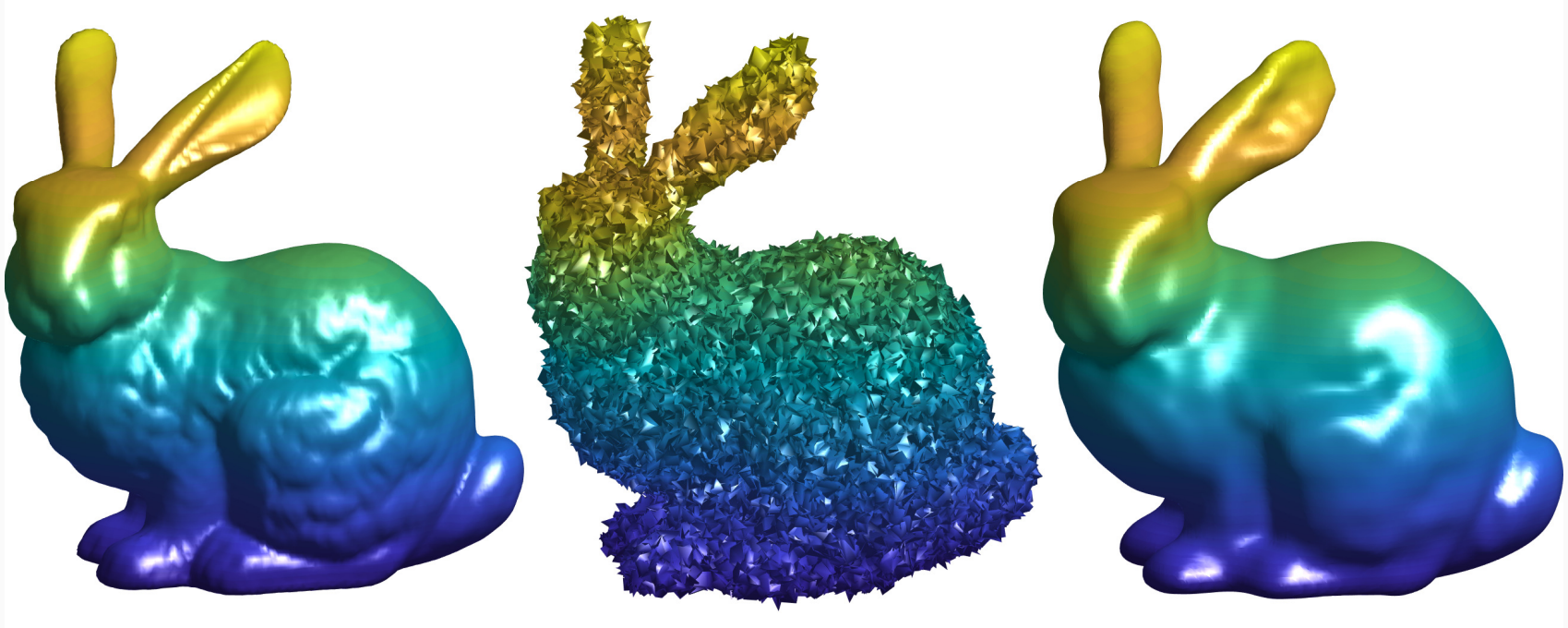
(a) Points (b) Exact (c) Scaled eigs (d) Lanczos

Figure 1: Prediction by different methods on the Hickory dataset.

Method	s_f	ℓ_1	ℓ_2	$-\log p(y\|\theta)$	Time [s]
Exact	0.696	0.063	0.085	1827.56	465.9
Lanczos	0.693	0.066	0.096	1828.07	21.4
Scaled eigs	0.543	0.237	0.112	1851.69	2.5

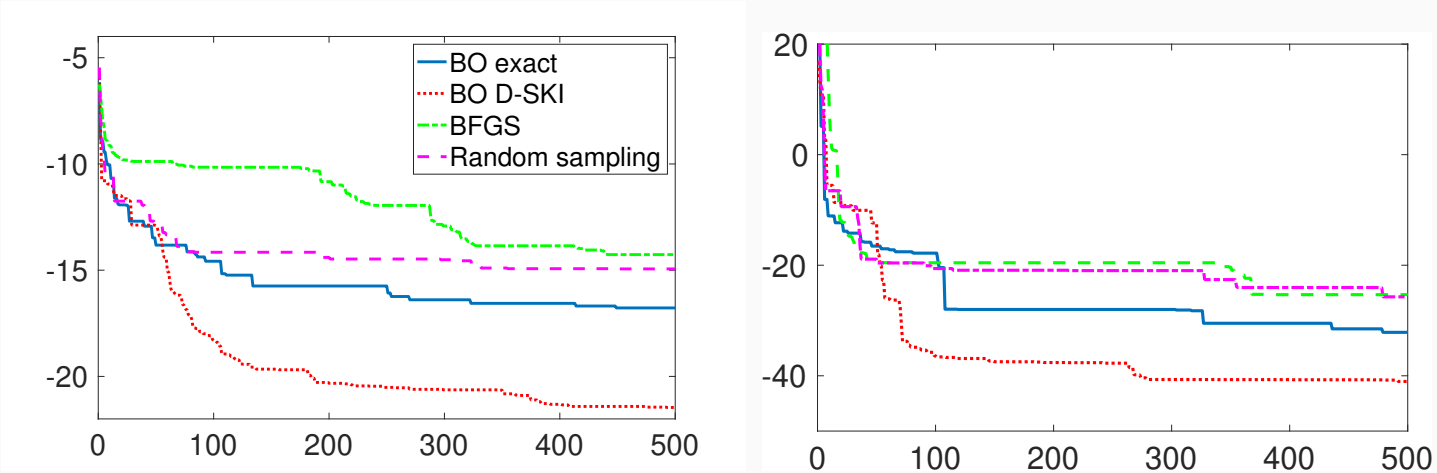
Table 1: Hyper-parameters recovered by different methods

Example: Surface reconstruction



Recovering the Stanford bunny model from 25K noisy normals.

Example: Bayesian optimization



- Learn active subspace of high-dimensional space
- Fit GP model with derivative in subspace
- Optimize expected improvement
  - Need predictive variance (use stochastic estimator)
  - Use low-rank approximation for control variate
- Works surprisingly well — still lots to do!

For more...

Eriksson, Dong, Lee, B., Wilson. Scaling Gaussian Process Regression with Derivatives. NeurIPS 2018.

Gardner, Pleiss, Weinberger, B., Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. NeurIPS 2018.

K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and A. G. Wilson, Scalable Log Determinants for Gaussian Process Kernel Learning. NIPS 2017.

GPyTorch: `https://gpytorch.ai`

Numerical Methods for Data Science: Latent Factor Models, Part I

David Bindel

17 June 2019

Department of Computer Science
Cornell University

Cornell CS 6241
Spring 2018

SJTU CS 258
June 2018 + May 2019

How do we teach numerical methods in a “data science” age?

- CS 4220 (S12): Data science projects in NA course
- CS 6241 (S18): “Numerical methods for data science”
- SJTU CS 258 (Jun 18+19): Undergrad version
- CS 3220 (F19): “Computational math for CS”

Lecture plan

Three threads from “lay of the land” to current research:

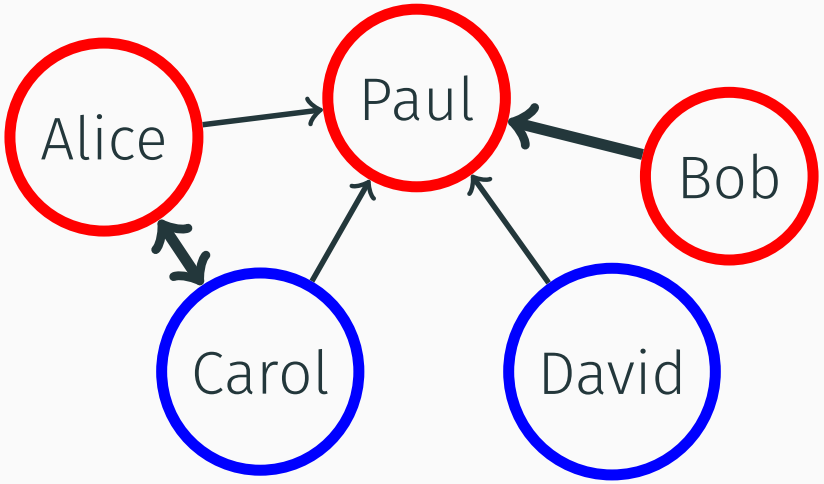
- Monday: Latent Factor Models
  - **Matrix data and decompositions**
  - NMF and topic models
- Wednesday: Scalable Kernel Methods
  - Structure and interpretation of kernels
  - Making kernel methods scale
- Friday: Spectral Network Analysis
  - Network spectra, optimization, and dynamics
  - Network densities of states

Matrix Data: Relations and Ranking

	Casa Blanca	Forest Gump	Rocky	The Matrix	...
Alice	5	5	1		
Bob	1		5	5	
Carol		2			
Dan		5		5	
...					

Matrix Data: Images and Functions

Matrix Data: Networks and Relations



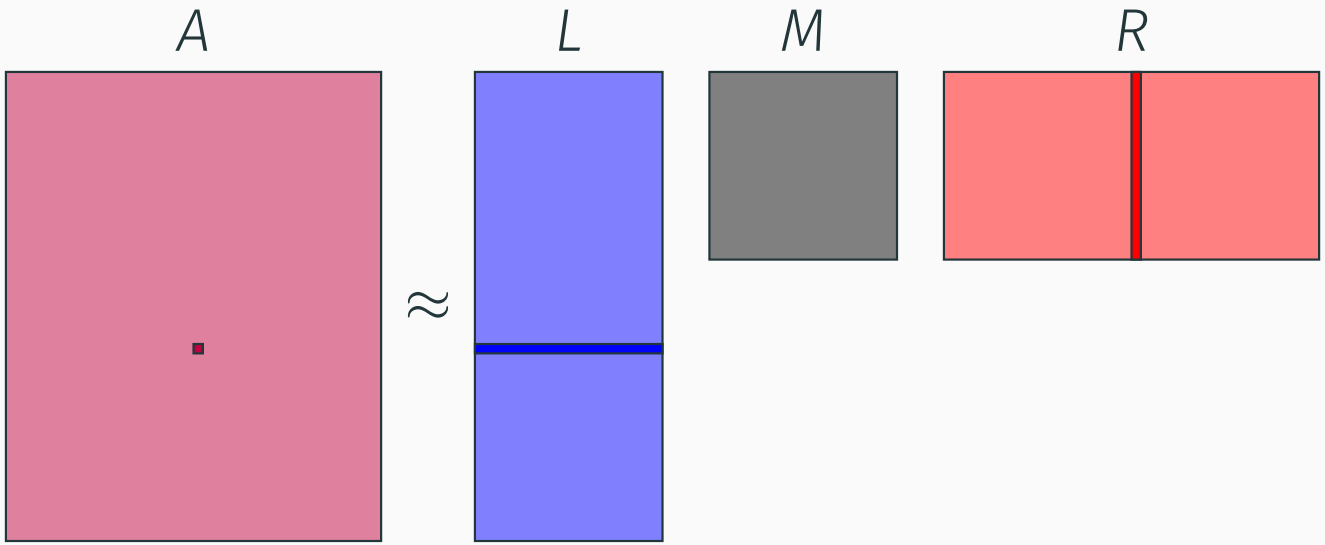
$$W = \begin{bmatrix} 0 & 0 & w_{AC} & 0 & 0 & w_{AP} \\ 0 & 0 & 0 & 0 & 0 & w_{BP} \\ w_{CA} & 0 & 0 & 0 & 0 & w_{CP} \\ 0 & 0 & 0 & 0 & 0 & w_{DP} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Matrix Data

- Relations between objects and features, e.g.
  - Documents and word frequencies (“bag of words” models)
  - User rankings of movies, songs, etc
  - Images and pixel values
  - Indicators for DNA markers in organisms
  - Treatments and outcome histograms
  - Snapshots of a state vector at different times
- Grid samples of bivariate functions
  - Image pixels indexed by row and column
  - PMF values for a discrete 2D random variable
- Relations between pairs of objects
  - Network relations (e.g. friendships)
  - Co-occurrence statistics, interaction frequencies, etc

Can generalize beyond pairwise data with *tensor* methods.

Latent Factor Modeling



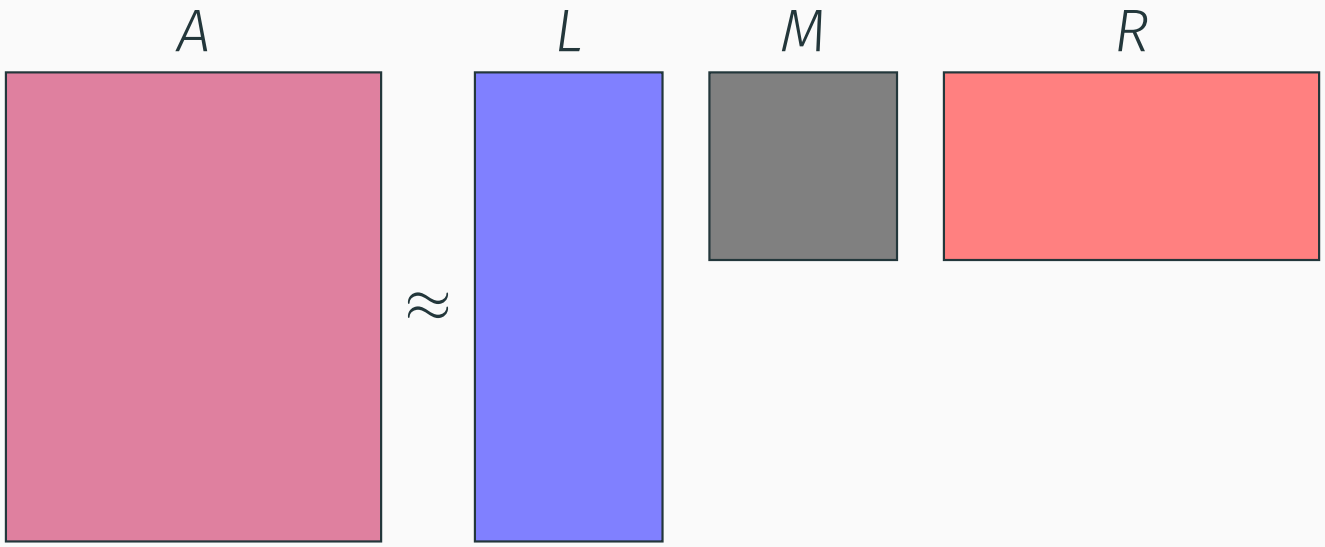
Map objects and features to *latent coordinates*, model data as bilinear function of coordinates:

$$a_{ij} \approx l_{i,:} \, M \, r_{:,j}.$$

Unknowns: latent coordinates for points, bilinear form.

This is underdetermined — need constraints on L, M, R for uniqueness.

Latent Factor Modeling



Model data arranged as a matrix $A \in \mathbb{R}^{m \times n}$ by

$$A \approx LMR, \quad L \in \mathbb{R}^{m \times r}, M \in \mathbb{R}^{r \times r}, R \in \mathbb{R}^{r \times n}$$

perhaps with constraints on L, M, and R.

Latent Factor Modeling

Model data arranged as a matrix $A \in \mathbb{R}^{m \times n}$ by

$$A \approx LMR, \quad L \in \mathbb{R}^{m \times r}, M \in \mathbb{R}^{r \times r}, R \in \mathbb{R}^{r \times n}$$

perhaps with constraints on L, M, and R.

From this, we would like to

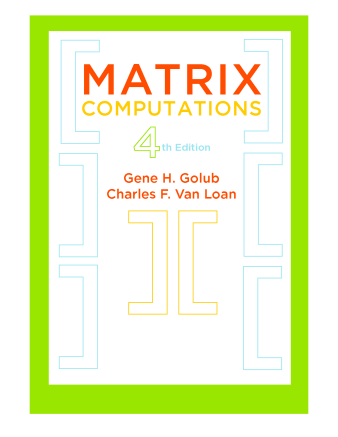
- Compress data
- Remove "noise" (including outliers)
- Fill in missing data
- Cluster and classify objects
- Find meaningful "parts" to data

with the Magic of Matrices.

The Magic of Matrices (or Machine Learning)

https://xkcd.com/1838/

What is The Matrix?



$A \in \mathbb{R}^{m \times n}$ is an m-by-n array of real numbers:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

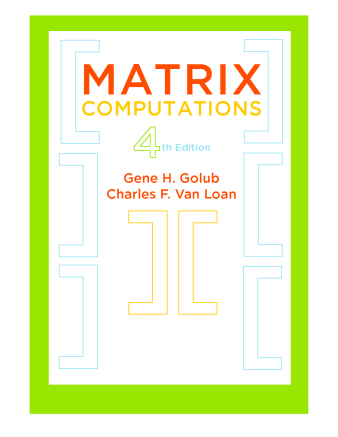
Is this right? Consider some a_{ij} meanings:

- Image pixel at row i and column j
- Departure of train j from station i
- Frequency of word j in document i

None seem all that linear algebraic!

Seek useful puns between *matrices* and *arrays*.

What is The Matrix?



In LA, a matrix represents (w.r.t. basis choice):

Linear map	$L : \mathcal{V} \to \mathcal{W}$	$w = Av$
Linear operator	$L : \mathcal{V} \to \mathcal{V}$	$w = Av$
Sesquilinear form	$b : \mathcal{V} \times \mathcal{W} \to \mathbb{R}$	$b = w^* A v$
Quadratic form	$\phi : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$	$\phi = v^* A v$

Different possible attitudes toward bases

- Pure LA: The LA object is the thing
  - Think basis independent (except canonical choices)
- Numerical LA: Sometimes bases matter!
  - Sparsity, shape, non-negativity
- Data: The matrix (array) is the thing

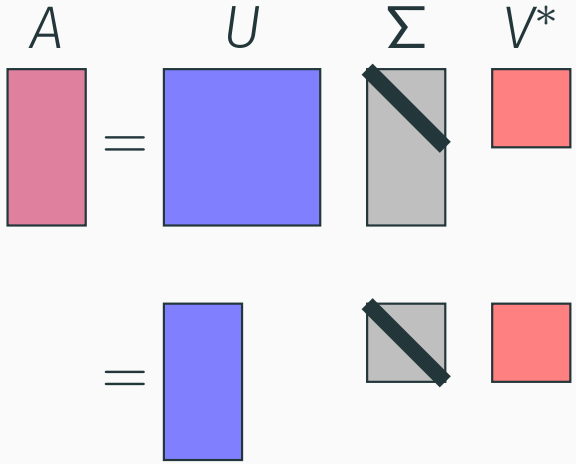
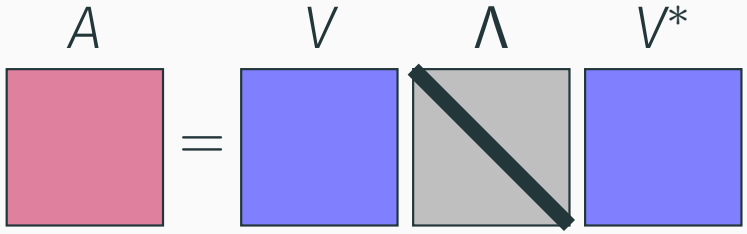
Matrices in Linear Algebra: Canonical Forms

	General bases	Orthonormal bases
Linear map (or bilinear form)	Rank/nullity $A = XI_kY^*$	**SVD** $A = U\Sigma V^*$
Linear operator	Jordan form $A = VJV^{-1}$	Schur form $A = UTU^*$
Quadratic form	Sylvester inertia $A = X_1X_1^* - X_2X_2^*$	**Symm eigendecomp** $A = V\Lambda V^*$

What if basis choice (identity of rows and columns) matters?

- Natural transformations are basis permutations.
- Permutation matrices $\subset$ orthogonal matrices.
- Orthogonal decompositions make good building blocks!

The SVD and the Symmetric Eigenvalue Problem

SVD	SEP
	
• U, V orthonormal • $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots)$ • σ_j descending, nonneg • Full: Σ rectangular • Economy: U rectangular	• V orthonormal • $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots)$ • λ_j descending • $A = A^*$ • Agrees with SVD if SPD

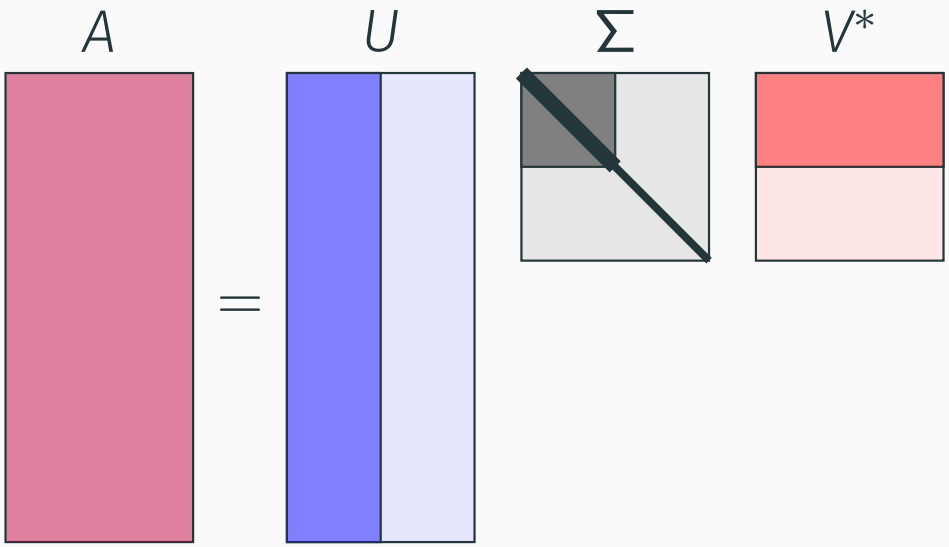
Construction

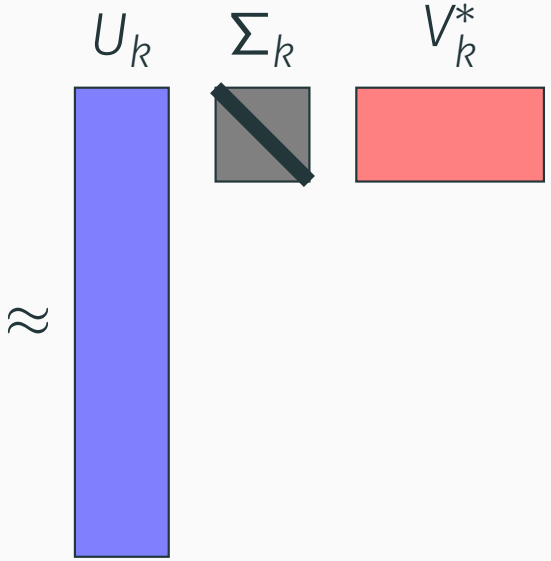
Variational construction of SVD (similar for SEP):

- Maximize $\|Av\|$ over $\|v\| = 1$
  - Unique maximum value, no local maxima!
  - Result: $Av_1 = \sigma_1 u_1$ where $u_1 = Av_1/\|Av_1\|$
- Maximize $\|Av\|$ over $\|v\| = 1$ and $v \perp v_1$
  - Again a "nice" optimization problem
  - Result: $Av_2 = \sigma_2 u_2$
- Continue in this fashion

Completely greedy is OK in principle! No need to backtrack.

Matrices and Data: Low-Rank Approximation by SVD





Matrices and Data: Low-Rank Approximation by SVD

Consider the (economy) SVD of $A \in \mathbb{R}^{m \times n}$ for $m \geq n$

$$A = U\Sigma V^*, \quad U \in \mathbb{R}^{m \times n}, \Sigma \in \mathbb{R}^{n \times n}, V \in \mathbb{R}^{n \times n}$$

where $U^*U = I$, $V^*V = I$ and Σ is diagonal with

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$$

Eckart-Young: Best rank k approximation is the truncated SVD

$$\hat{A} = U_k \Sigma_k V_k^*.$$

True in Froebenius norm, spectral norm. Error is

$$\|A - \hat{A}\|_F^2 = \sum_{j=k+1}^{n} \sigma_j^2, \qquad \|A - \hat{A}\|_2 = \sigma_{k+1}$$

Actual Computation

SVD and SEP admit similar computational schemes

- Small and dense (up to a couple 1000, MATLAB eig/svd):
  - Orthogonal reduction to bidiagonal / tridiagonal ($O(n^3)$)
  - Cheaper reduction rest of the way to the decomposition
- Subspace iteration
  - Start with random orthonormal columns
  - Multiply by A (and maybe A^*) a couple times
  - RandNLA: Few steps for OK accuracy (with oversample)
  - Good for cache use, great for modest accuracy
- Krylov methods (Matlab eigs and svds)
  - Repeatedly multiply a few vectors by A (and maybe A^*)
  - Orthogonalize at each step (parallel bottleneck)
  - Various restarting and acceleration tricks
  - High accuracy for a few extreme pairs

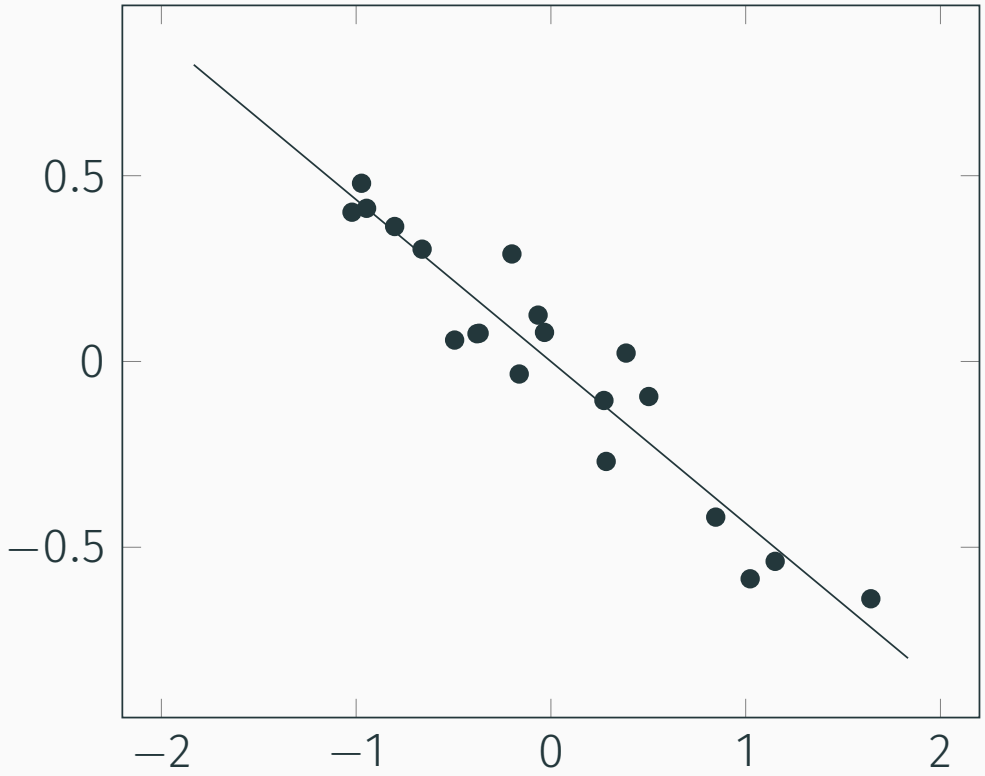
Some Computational Aspects

Big takeaways:

- We have good codes that (mostly) “just work”
- People have thought through how to make them run fast
- Have good backward stability properties
- No misconvergence or sensitivity to starting point

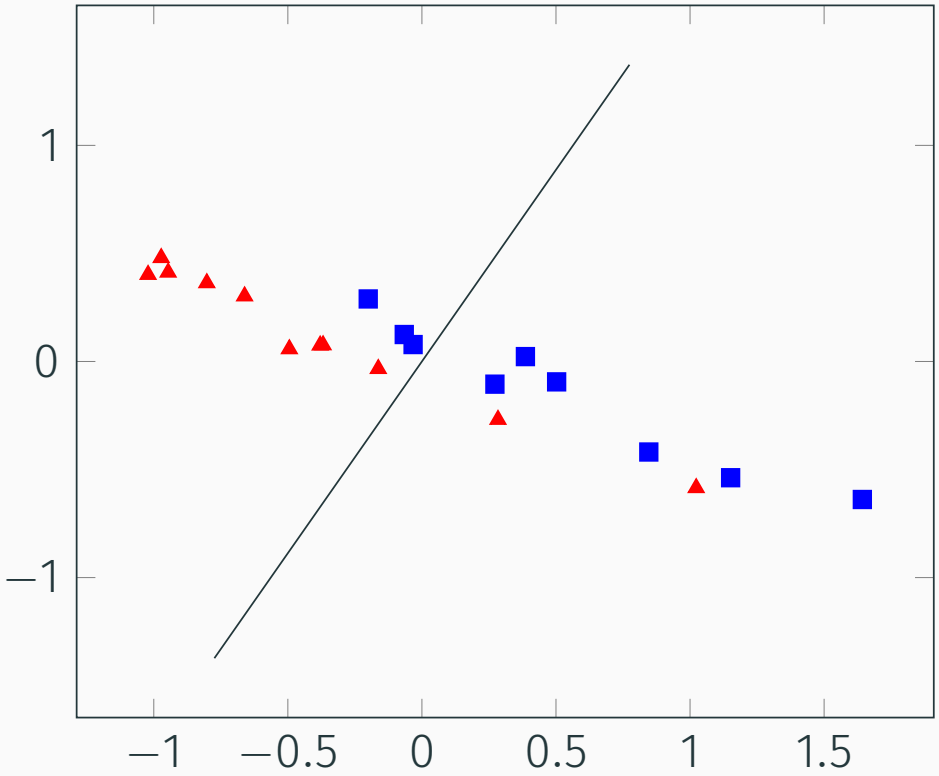
These are *great* building blocks

Example: PCA



- Matrix rows represent different object properties
- Typically center (subtract column means); may scale
- Run SVD on resulting matrix
- Dominant singular values are “principal components”

Example: LDA



- Matrix rows represent different object properties
- Rows are labeled, want to discriminate these labels
- Solution involves a generalized SEP
- May be very different from PCA directions

Example: Latent semantic analysis

Vector space model (Salton and Yang, 1975)

- Columns are documents, row entries are word frequency
- Scale raw data (tf-idf)
- Take SVD of resulting matrix
- Rows of $U \approx$ latent word representations
- Columns of $V \approx$ latent document representations
- Useful for comparing words to words, docs to docs
- Also useful for searching for docs by keywords

Problem: latent coordinates are hard to interpret!

Example: Spectral Clustering

Goal: Cluster objects (rows) according to features

- Compute a truncated SVD: $A \approx U_r \Sigma_r V_r^*$
- Treat rows of $U_r \Sigma_r$ as latent coordinates
- Run k-means to cluster points

Essentially gives

$$A \approx LC^*$$

where

$$l_{ij} = \begin{cases} 1, & \text{item } i \text{ in class } j \\ 0, & \text{otherwise} \end{cases} \qquad C_{:,j} = \text{centroid of class } j$$

Several types of spectral clustering – will discuss again later!

The Story So Far

SVD and SEP provide useful dimension reduction

- Lower-dimensional “latent spaces” for further analysis
- Latent space clarifies object similarity / dissimilarity
- But interpreting the coordinate system is hard!

So what do we want beyond SVD and SEP?

- Linear dimension with interpretable structure (today)
- Nonlinear dimension reduction (a little coming up)

Recall: The Latent Factor Framework

Factor data matrix $A \in \mathbb{R}^{m \times n}$ as

$$A \approx LMR, \quad L \in \mathbb{R}^{m \times r}, M \in \mathbb{R}^{r \times r}, R \in \mathbb{R}^{r \times n}$$

with different structures on L, M, and R.

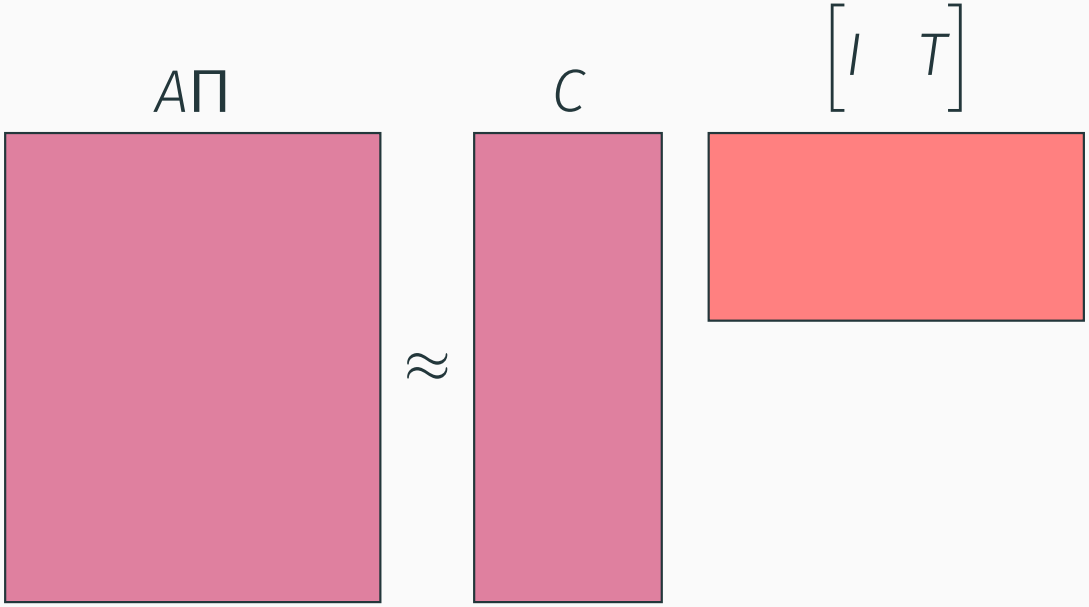
- R (or MR) as maps original features to "latent" features
- Can impose constraints on L, M, and R
- Orthogonality constraints gets us to SVD (easy)
- Other types of constraints are harder
- ...but other constraints improve interpretability

Factor Selection and Skeletonization

Idea: Express latent factors via rows/columns of A

- Improves interpretability (maybe — more tricks help)
- May improve cost of working with matrix
  - Can form part of a row/column without forming all
  - Useful in both experiments and computation
- *But* how do we choose good representative rows/cols?
  - Want things to be as linearly independent as possible
  - Also want good approximation quality
  - SVD provides a “speed of light” bound

Interpolative Decomposition / Skeletonization



- C consists of the leading k columns of A
- $T = C^{\dagger}(A\Pi)_{:,k+1:n}$ chosen to minimize Frobenius norm error
- Exists some Π — not easily computed — such that
  - Entries of T are at most 2
  - Singular values of $\begin{bmatrix} I & T \end{bmatrix}$ in $[1, 1 + \sqrt{k(n-k)}]$
  - Approximation error within $1 + \sqrt{k(n-k)}$ of optimal

CUR Decomposition

Idea: $A \approx CUR$ where

- C consists of columns of A
- R consists of rows of A
- $U = C^{\dagger} A R^{\dagger}$ is optimal given C, R
- Selection of good C and R is again the challenge

Can also consider symmetric variants where $R = C^*$.

Pivoted QR

Pivoted QR decomposition is

$$A\Pi = QR$$

where R is upper triangular, r_{ii} are positive and decreasing.

Idea: Be greedy (as in SVD), but choose among columns of A

- Choose column a_i with maximum norm
  - Set $r_{11} = \|a_i\|$ and $q_1 = a_i/r_{11}$ (so $a_1 = q_1 r_1$)
  - Orthogonalize vs q_1: $a'_j = a_j - q_1 r_{j1}$ with $r_{j1} = q_1^T a_j$
- Choose modified column a'_i with maximum norm
  - Set $r_{22} = \|a'_i\|$ and $q_2 = a'_i/r_{22}$
  - Orthogonalize vs q_2 to get new A' matrix
- Keep repeating the process (re-ordered Gram-Schmidt)

Unlike SVD this is *not* optimal — but often pretty good.

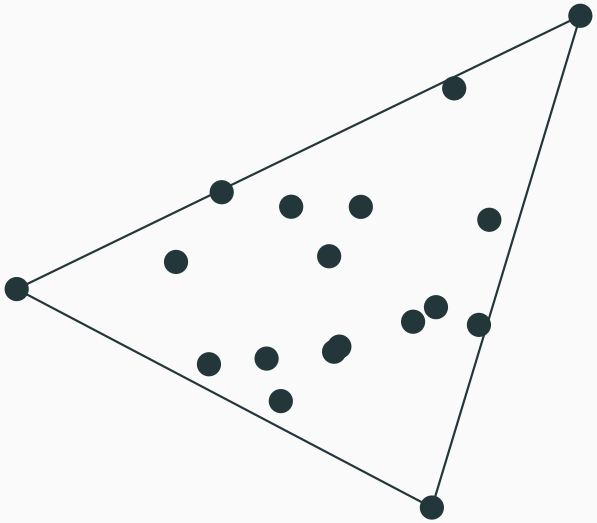
Computing Pivoted QR

As with SVD, constructive definition $\neq$ usual computation.

- Dense case is a standard building block (qr in MATLAB)
- Ongoing work to improve parallelism and cache efficiency
- Clever tournament pivoted (TSQR) for $m \gg n$

Again: A great building block to borrow from someone else!

The Geometry of Pivoted QR



Chosen column is on the convex hull of a 1D projection

$\implies$ it is a point on the convex hull of the original columns.

Pivoted QR to Pivoted Cholesky

Symmetric positive definite matrices $\equiv$ Gram matrices:

$$H = A^T A$$

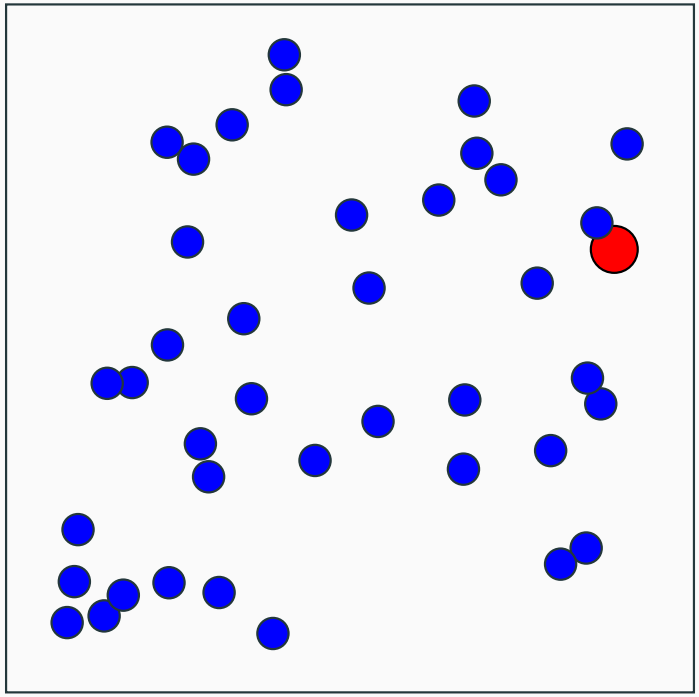
Decompose:

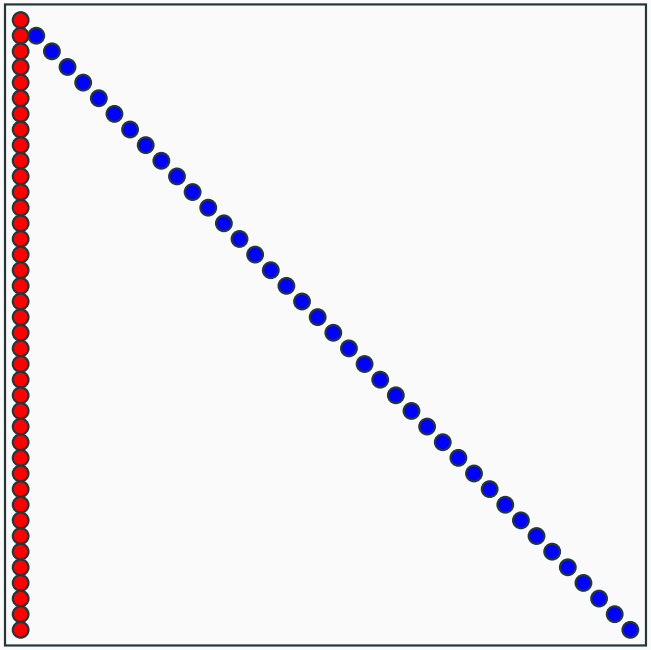
$$A\Pi = QR \implies$$
$$\Pi^T H \Pi = R^T Q^T Q R = R^T R$$

Running pivoted Cholesky is equivalent to pivoted QR.

Esp useful when A is implicit (only access via H).
Let's look briefly at an example we'll see again on Wednesday.

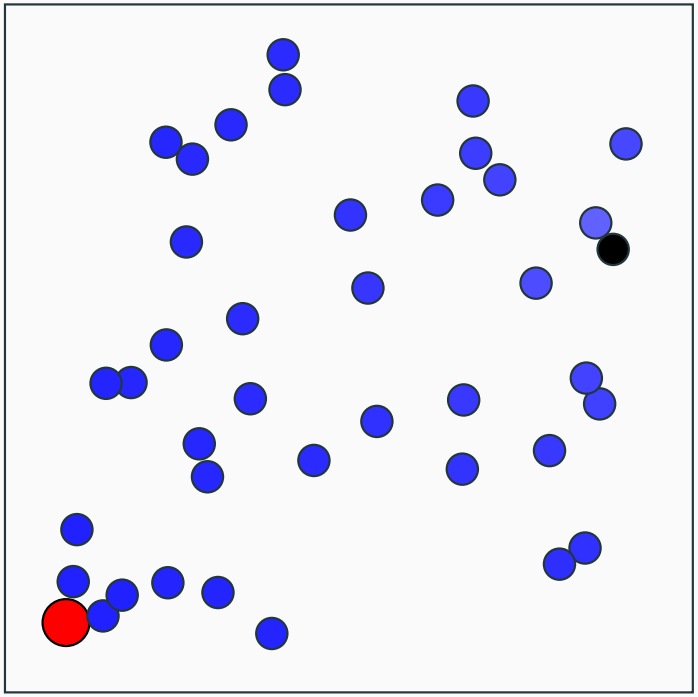
Example: Pivoted Cholesky on a Kernel Matrix

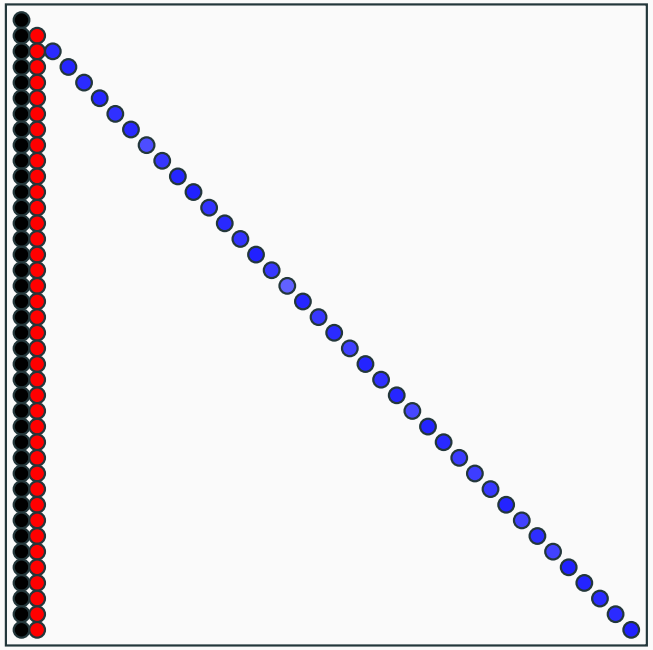




Diagonal element: 1.00e+00

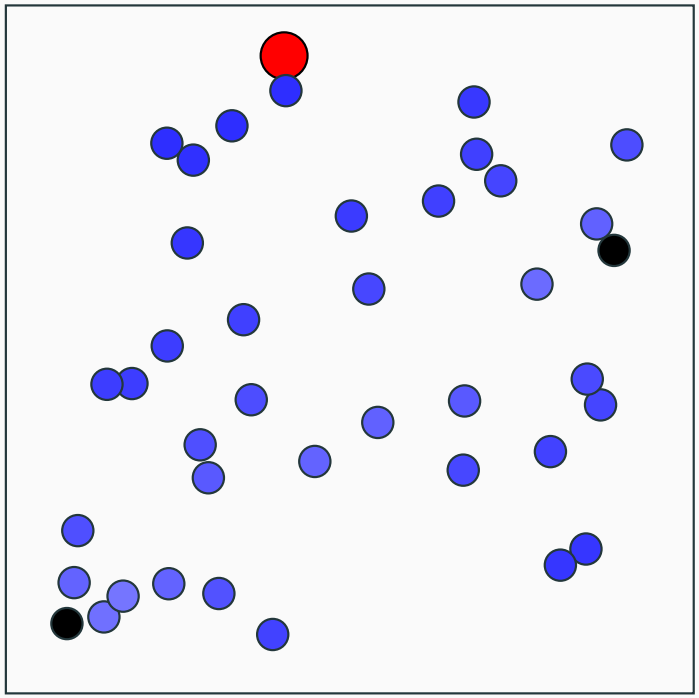
Example: Pivoted Cholesky on a Kernel Matrix

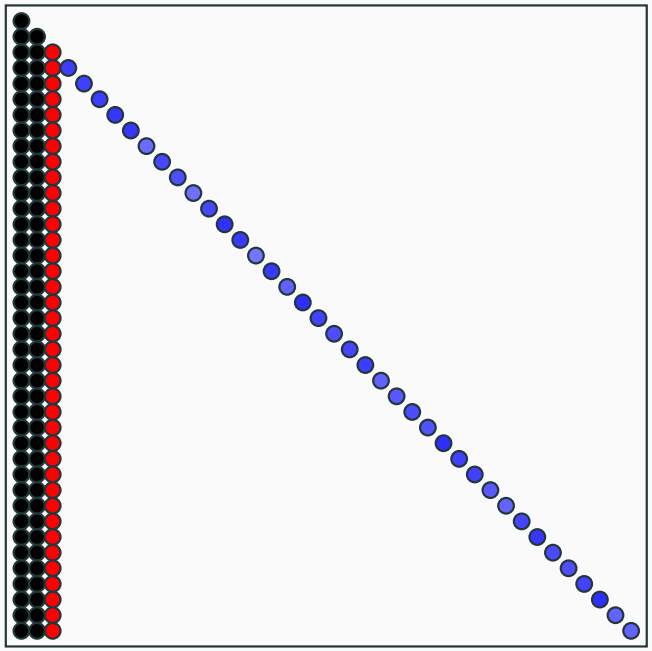




Diagonal element: 6.77e-02

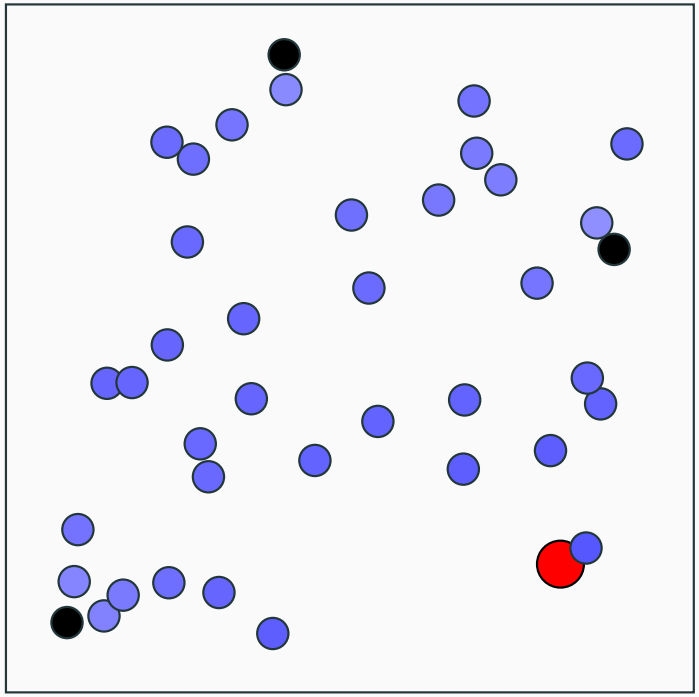
Example: Pivoted Cholesky on a Kernel Matrix

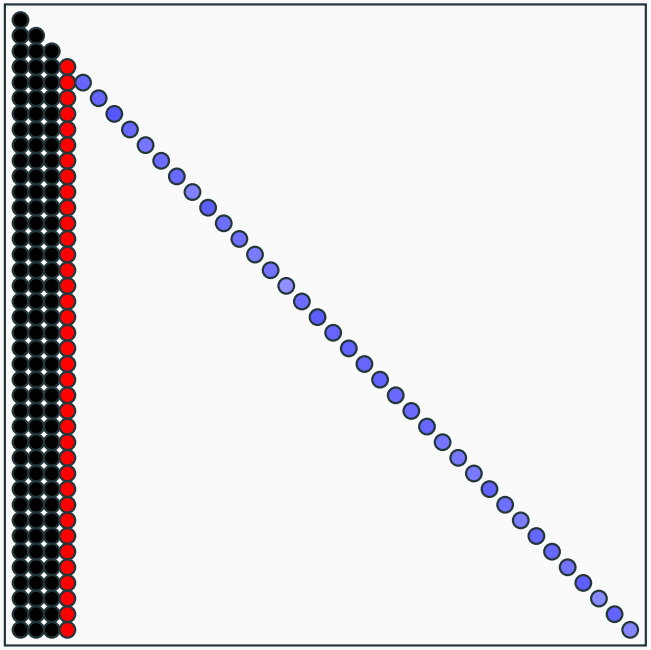




Diagonal element: 1.91e-02

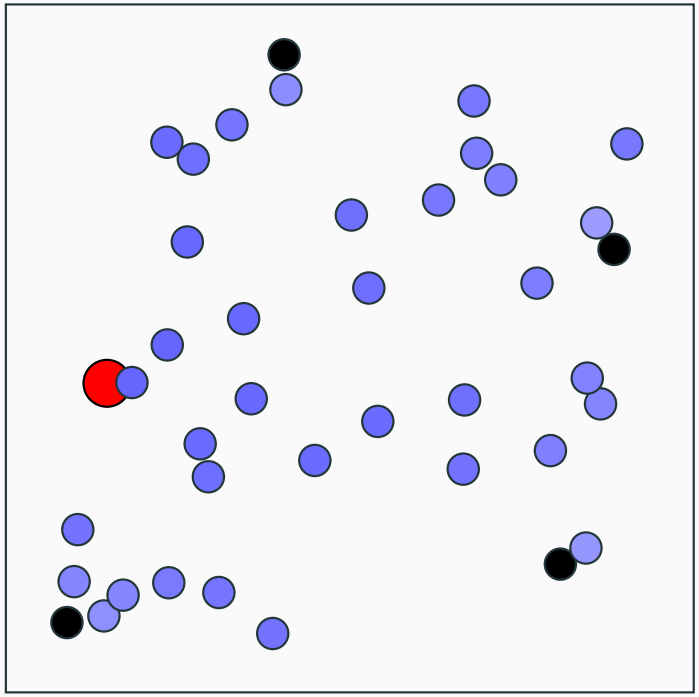
Example: Pivoted Cholesky on a Kernel Matrix

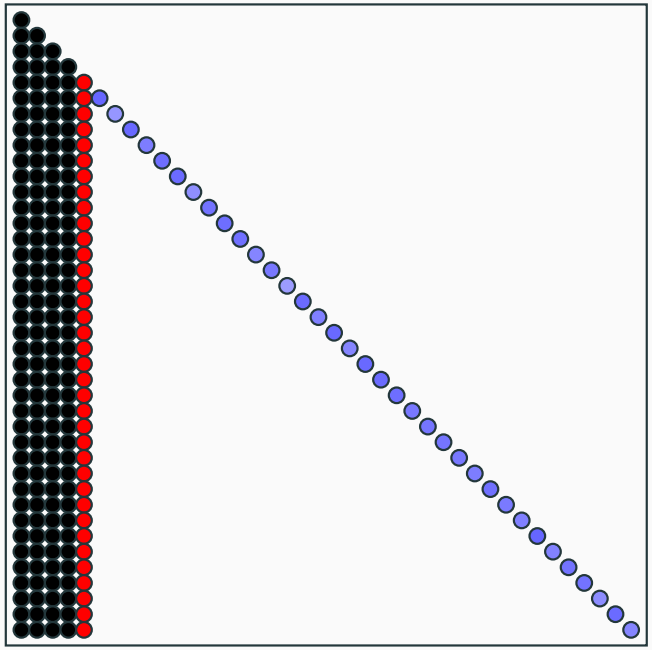




Diagonal element: 5.11e-04

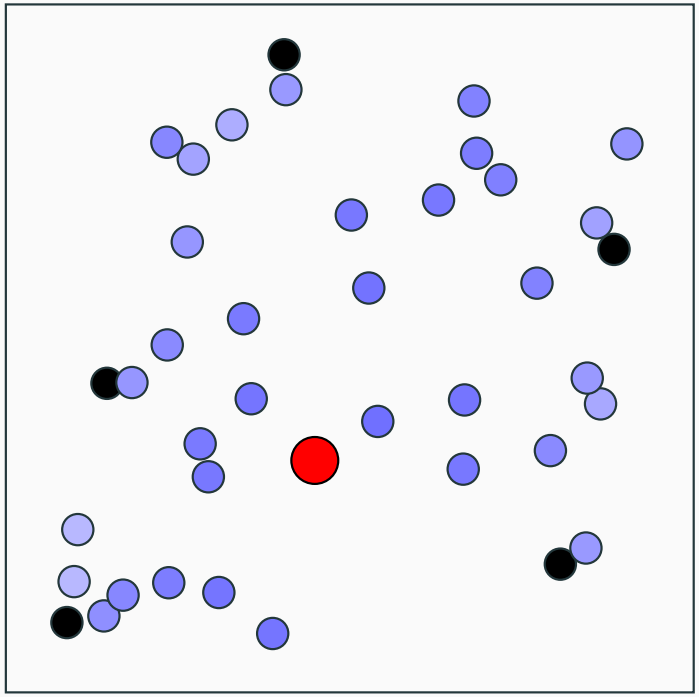
Example: Pivoted Cholesky on a Kernel Matrix

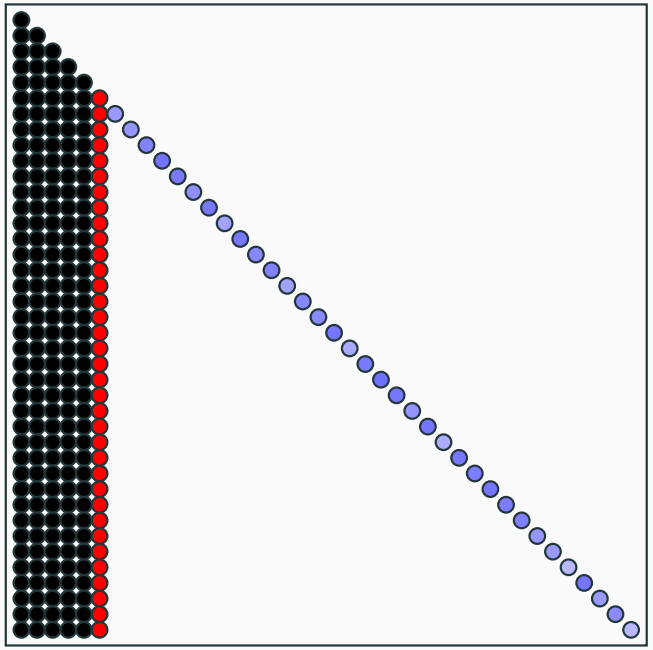




Diagonal element: 1.19e-04

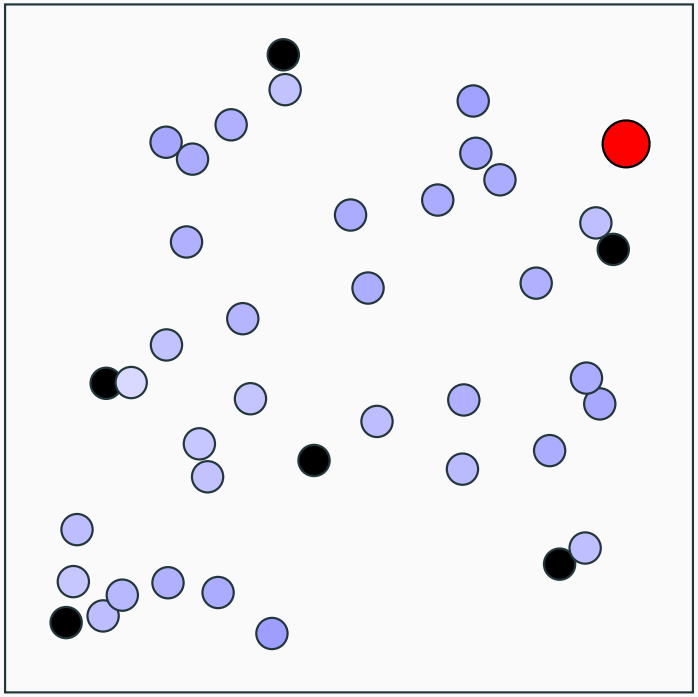
Example: Pivoted Cholesky on a Kernel Matrix

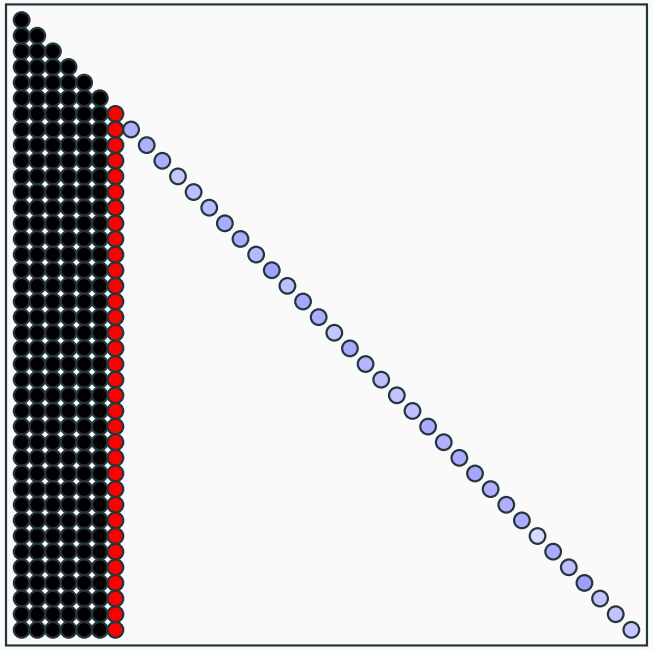




Diagonal element: 4.18e-05

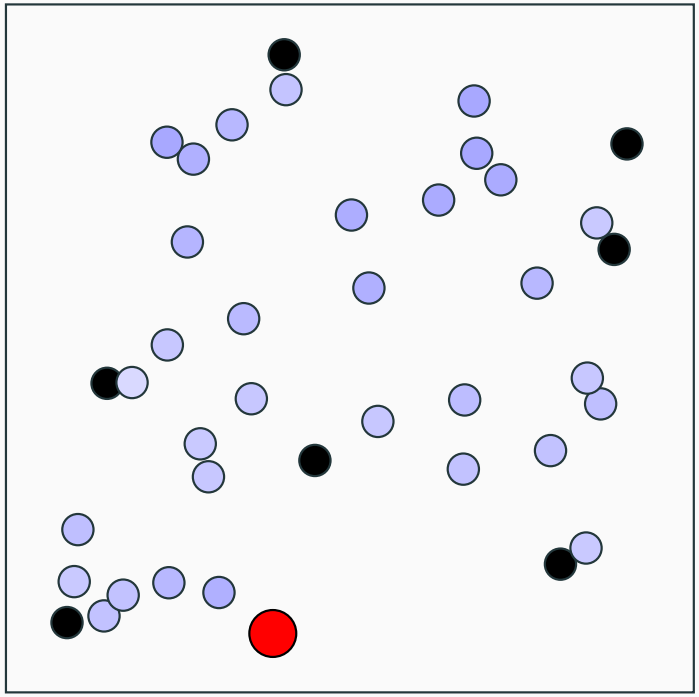
Example: Pivoted Cholesky on a Kernel Matrix

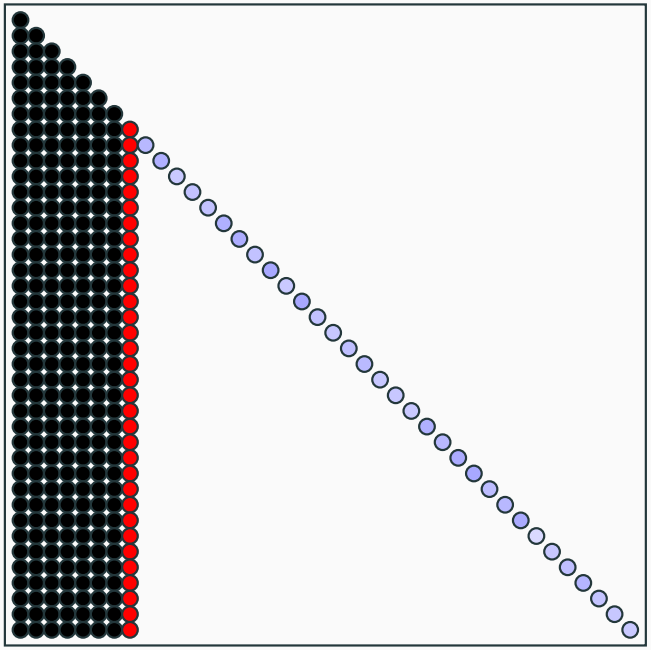




Diagonal element: 8.54e-07

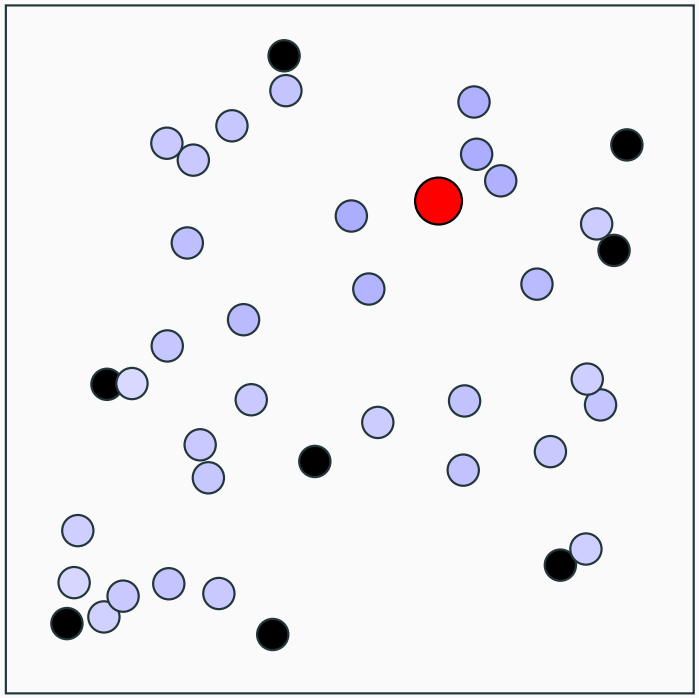
Example: Pivoted Cholesky on a Kernel Matrix

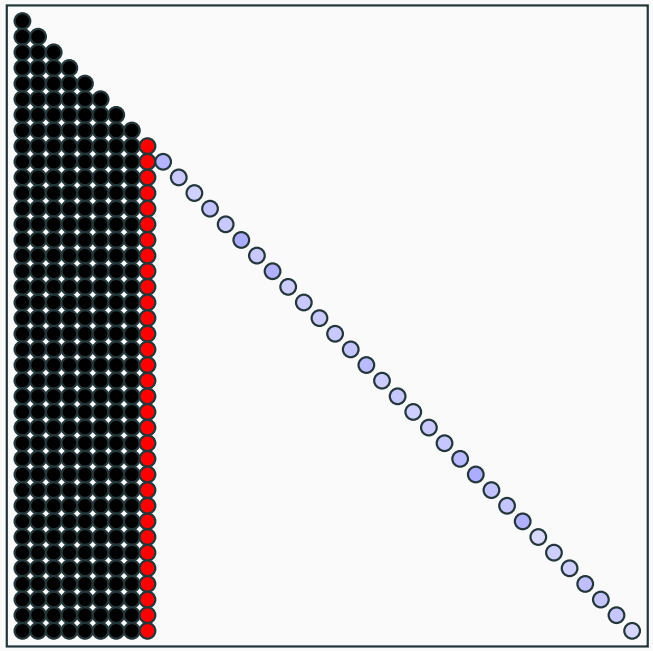




Diagonal element: 3.58e-07

Example: Pivoted Cholesky on a Kernel Matrix





Diagonal element: 1.92e-07

Improving the Decomposition

Recall interpolative decomposition:

$$A\Pi \approx C \begin{bmatrix} I & T \end{bmatrix}$$

and can keep all $|t_{ij}| \leq 2$ with right Π.

Idea: Start from pivoted QR and refine

- Compute truncated pivoted QR

$$A\Pi = QR = \begin{bmatrix} Q_1 & Q_1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \approx Q_1 \begin{bmatrix} R_{11} & R_{12} \end{bmatrix}$$

- Set $C = Q_1 R_{11}$ and $T = R_{11}^{-1} R_{12}$
- For large entries ($|t_{ij}| > 2$), swap columns π_i and π_{r+j}.
- Recompute T; repeat swaps until happy.

Next Time: Topics, Parts, and NMF

What if we want a low-rank factorization with more structure?

- Non-negativity?
- Sparsity of factors?
- Normalization for a probabilistic interpretation?

Hard in general, but there are some effective approaches.

After the break: From non-negative matrix factorization to spectral topic modeling

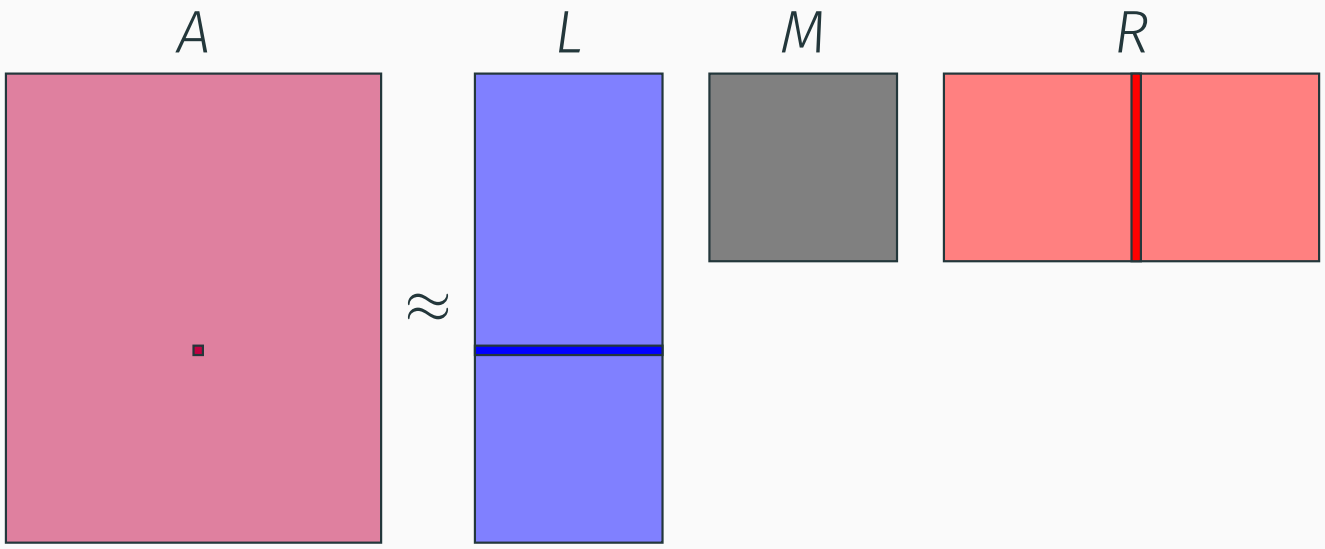
Numerical Methods for Data Science: Latent Factor Models, Part II

David Bindel

17 June 2019

Department of Computer Science
Cornell University

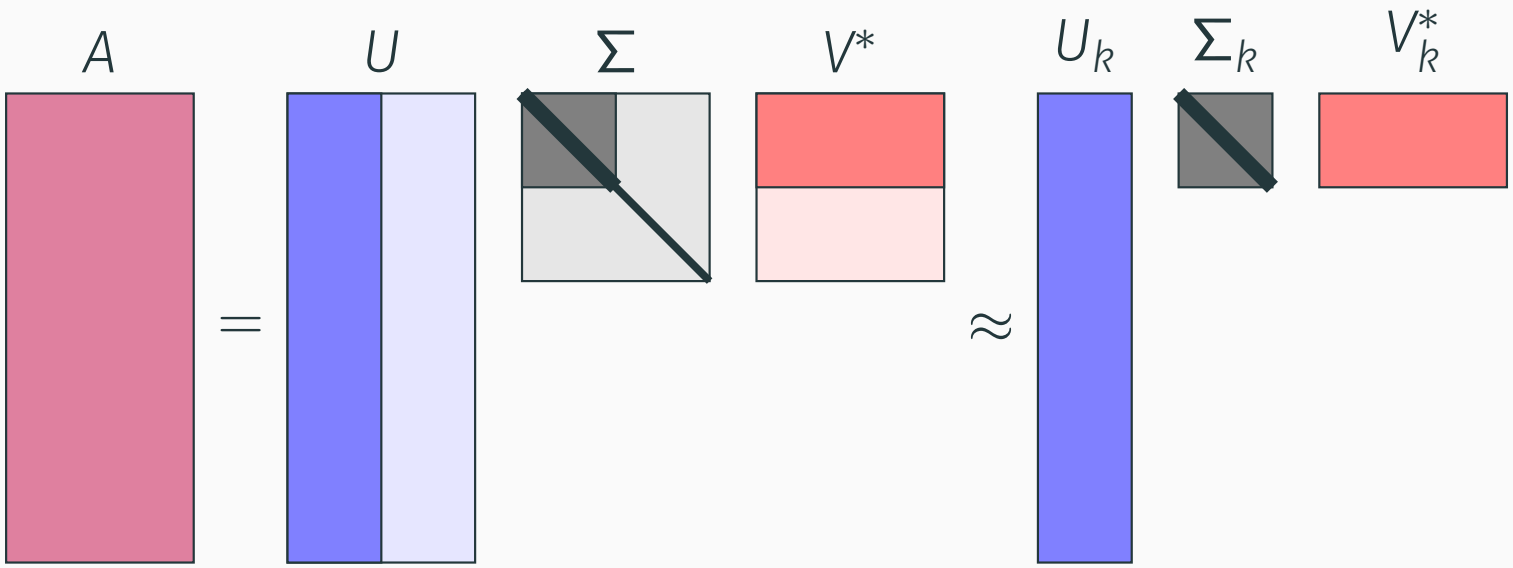
Reminder: Latent Factor Modeling



Underdetermined without constraints on L, M, and R.

- Simplest: Orthonormality constraints (SVD)
- Harder: L or R drawn from A (ID, CUR)
- Hardest: **Non-negativity and sparsity constraints**

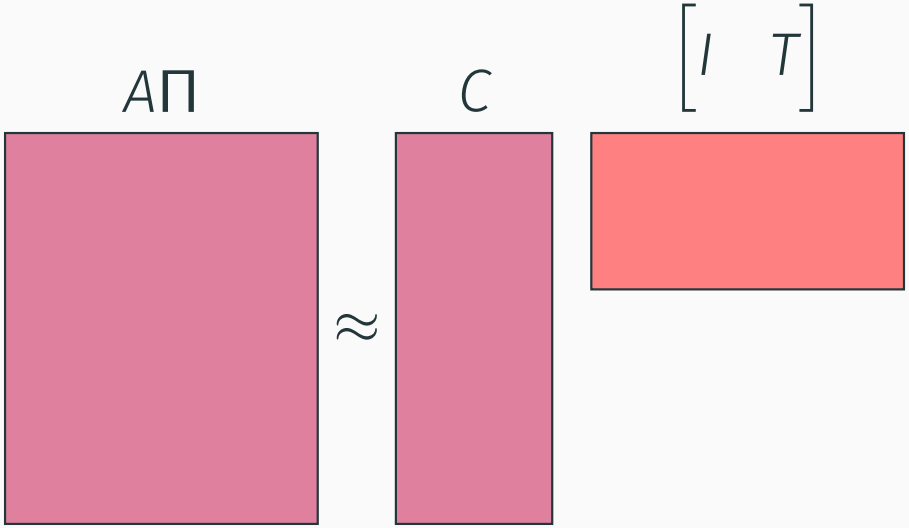
Reminder: Latent Factor Modeling and SVD



Simplest: Orthonormality constraints (SVD and the like)

- Straightforward to compute
  - Lots of good codes to use (sparse or dense case)
  - Existence proof is a greedy algorithm!
- Provides useful comparisons for indexing, clustering, etc
- Factors (latent coordinates) are not easy to interpret

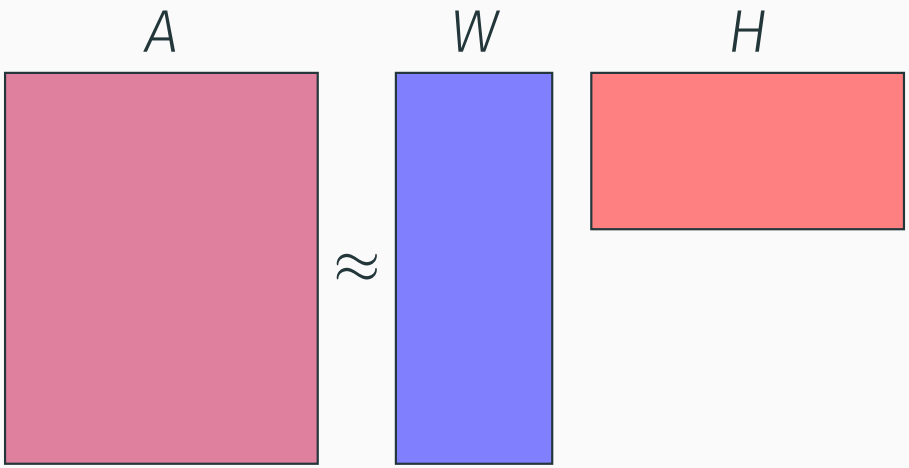
Reminder: Latent Factor Modeling and ID/CUR



Harder: Subset constraints (factors from rows/cols of A)

- Good selection of rows/cols gives
  - Nice factors (e.g. $|t_{ij}| \leq 2$)
  - Approximation error bounds
- Good building block: pivoted QR (greedy approach)
  - Greedy and suboptimal (though quality bounds exist)
  - Provides a good initial guess (refine by swapping)

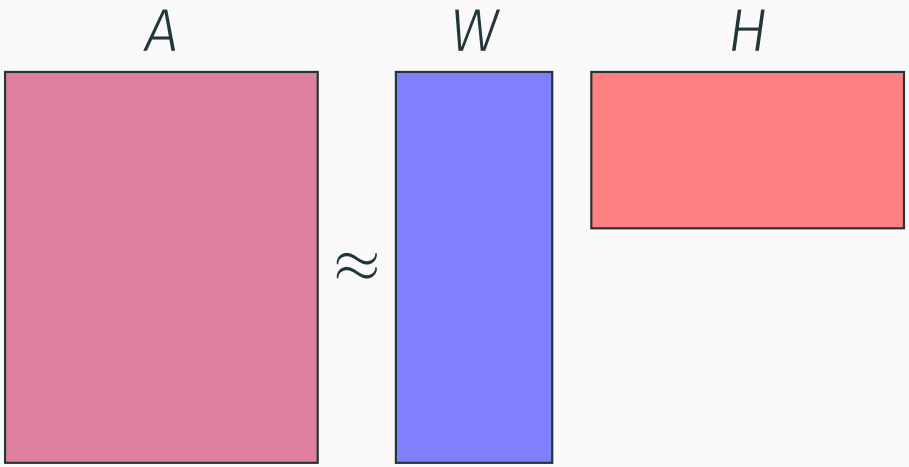
Latent Factor Modeling and Non-Negativity



Hardest: Non-negativity constraints ($W \in \mathbb{R}_+^{m \times k}, H \in \mathbb{R}_+^{k \times n}$)

- A hard optimization problem!
  - Standard algorithms converge to local minima
  - Few nontrivial approximation error bounds
- But this is great for interpretability!

Interpreting Non-Negative Matrix Factorizations (NMF)



Domain	Object ($a_{:,j}$)	Cluster ($w_{:,k}$)
Document	Word	Topic
Image	Pixel	Segment
Network	User	Community
Legislature	Member	Party/Group
Playlist	Song	Genre
Chemical spectra	Mixture	Molecules

Why is it Hard?

Non-negative matrix factorization is hard!

- The role of k is very unclear
- The problem is nonconvex with many local minima
- Greedy strategies are very suboptimal

But: sometimes it seems easier in practice!

Can deal with this in two ways

- Use locally-convergent iterations
  (and maybe think about initialization)
- Find structural reasons for non-hardness

We will tackle both.

The Optimization Problem

Goal:

$$A \approx WH$$

Several options to quantify "$\approx$"; we use Frobenius norm:

$$\text{minimize } \frac{1}{2}\|A - WH\|_F^2 \text{ where } W \in \mathbb{R}_+^{m \times k}, H \in \mathbb{R}_+^{k \times n}$$

Getting the Gradient

Note $\|\cdot\|_F$ is a Euclidean norm for the inner product

$$\langle X, Y\rangle_F = \mathrm{tr}(Y^T X)$$

Compute variations (directional derivatives) of

$$\phi(W, H) = \|R\|_F^2 = \|A - WH\|_F^2$$

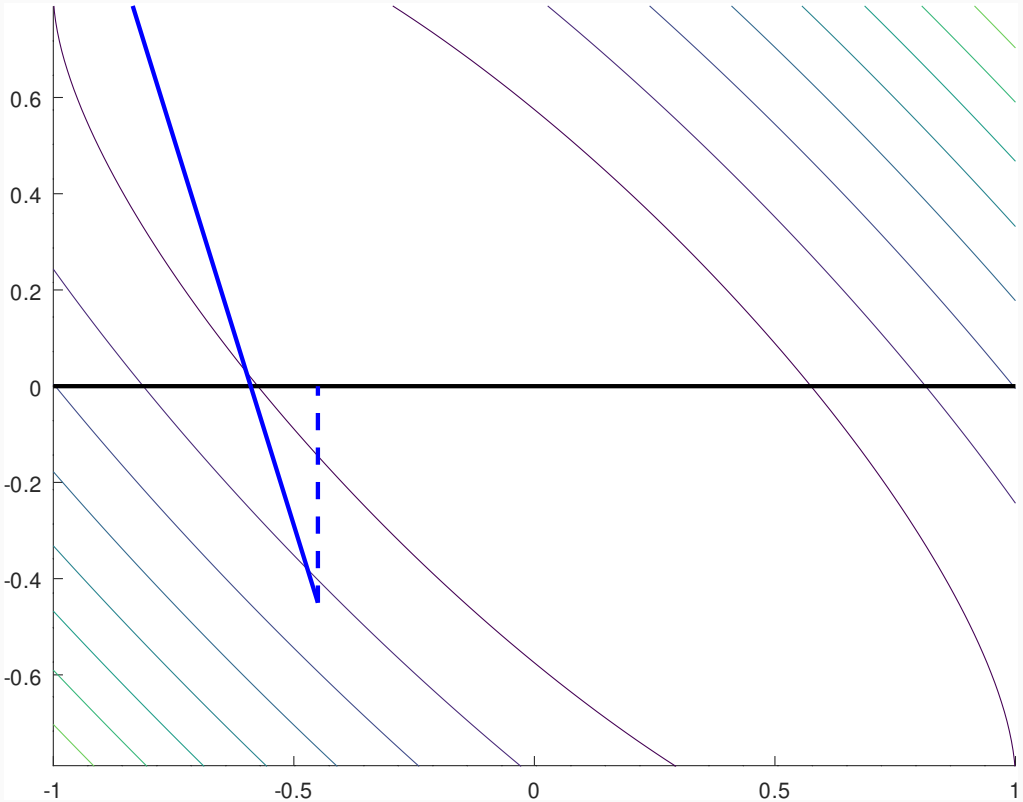
as follows:

$$\delta\phi = \delta\left[\frac{1}{2}\langle R, R\rangle_F\right] = \langle \delta R, R\rangle_F = -\langle (\delta W)H, R\rangle_F - \langle W(\delta H), R\rangle_F.$$

Do some algebra (cyclic invariance of traces) to rearrange as

$$\delta\phi = -\langle \delta W, RH^T\rangle_F - \langle \delta H, W^T R\rangle_F$$

Projected Gradient Descent



$$x^{k+1} = \mathcal{P}\left(x^{k+1} - \alpha_k \nabla \phi(x^k)\right)$$

Projected Gradient Descent for NMF

Given an objective ϕ and a projection $\mathcal{P}$ onto a convex constraint set, projected gradient descent iteration is

$$x^{k+1} = \mathcal{P}\left(x^{k+1} - \alpha_k \nabla\phi(x^k)\right)$$

In our case: $\mathcal{P}(X) = [X]_+$ is elementwise max with zero, and

$$\delta\phi = -\langle \delta W, RH^T\rangle_F - \langle \delta H, W^T R\rangle_F$$

so

$$W^{\text{new}} = [W + \alpha RH^T]_+$$
$$H^{\text{new}} = [H + \alpha W^T R]_+$$

This converges for appropriate choices of α.

Multiplicative Updates

Can also choose more than one step size in PGD:

$$W^{\text{new}} = [W + S \odot (RH^T)]_+$$
$$W^{\text{new}} = [H + S' \odot (W^T R)]_+$$

Choose non-negative step sizes

$$S = W \oslash (WHH^T), \quad S' = H \oslash (W^T WH).$$

With these choices, terms cancel, so that

$$W^{\text{new}} = S \odot (AH^T) = W \oslash (WHH^T) \odot (AH^T)$$
$$H^{\text{new}} = S' \odot (W^T A) = H \oslash (W^T WH) \odot (W^T A).$$

Scaling is such that we no longer have to project.
Guaranteed non-increasing – but no convergence guarantee.

This is the Lee and Seung scheme.

Death by a Million Steps

Pete Stewart on "long run" convergence of gradient descent:

> *But this* **long run** *is a misleading guide to current affairs.* **In the long run** *we are all dead.*
>
> — *John Maynard Keynes (1923)*

Issues:

- Each step is expensive (compute with full matrices)
- We may need a lot of steps
- We need some smart way to choose α (for PGD)

Can we do better? Maybe with cheaper steps?

One Approach: Stochastic Gradient Descent

Projected gradient descent iteration:

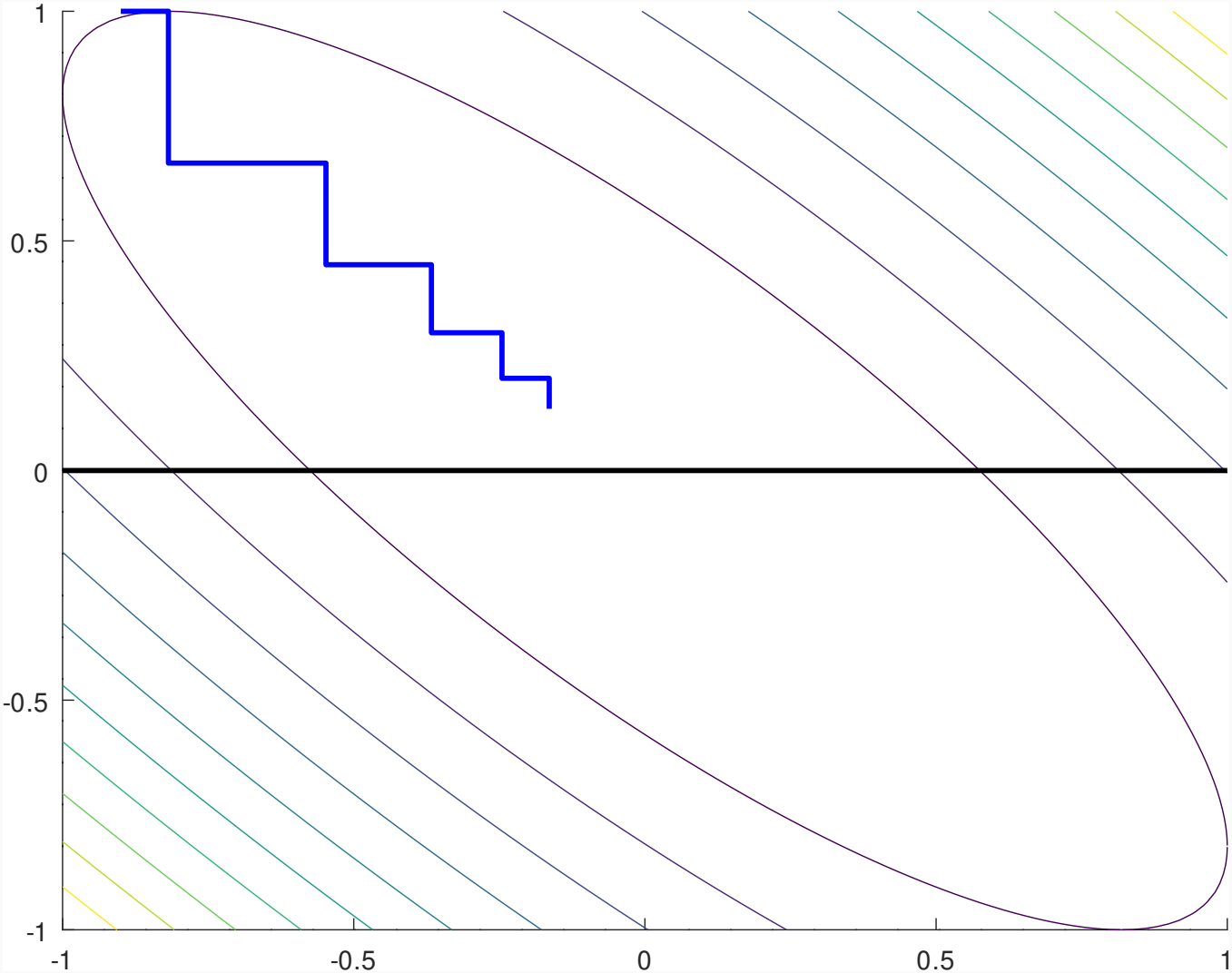
$$x^{k+1} = \mathcal{P}\left(x^{k+1} - \alpha_k \nabla\phi(x^k)\right)$$

Projected SGD:

$$x^{k+1} = \mathcal{P}\left(x^{k+1} - \alpha_k g(x^k)\right)$$
$$\mathbb{E}[g(x)] = \nabla\phi(x)$$

Converges, if slowly, for appropriate α_k (hard to choose).
And each step is cheap.

Coordinate Descent



(Block) Coordinate Descent

Idea: Partition variables and sweep over them; at each step

- Fix all but the current (block) of variables
- Take an optimization step on that variable (block)

Repeat until convergence.

Convergence[1] if each subproblem has a unique solution.
Q: What should the blocks / subproblems be?

[1] to a stationary point

Three Variants on Coordinate Descent

- Simple coordinate descent
  - Very cheap updates
  - Looks like PGD, but less parallel (GS vs Jacobi)
- $2k$ vector blocks (columns of W and rows of H)
  - Hierarchical Alternating Least Squares (HALS) / Rank-One Residual Iteration (RRI)
  - Subsolves are single-variable non-negative LS problems
  - Equivalent to simple coordinate descent (for one order)
- Alternately minimize on W and H
  - Alternating Non-Negative Least Squares (ANLS)
  - Yields one NNLS per row of W or column of H
  - Simple active-set solvers can work pretty well here!

Can accelerate any of these (e.g. Anderson acceleration).

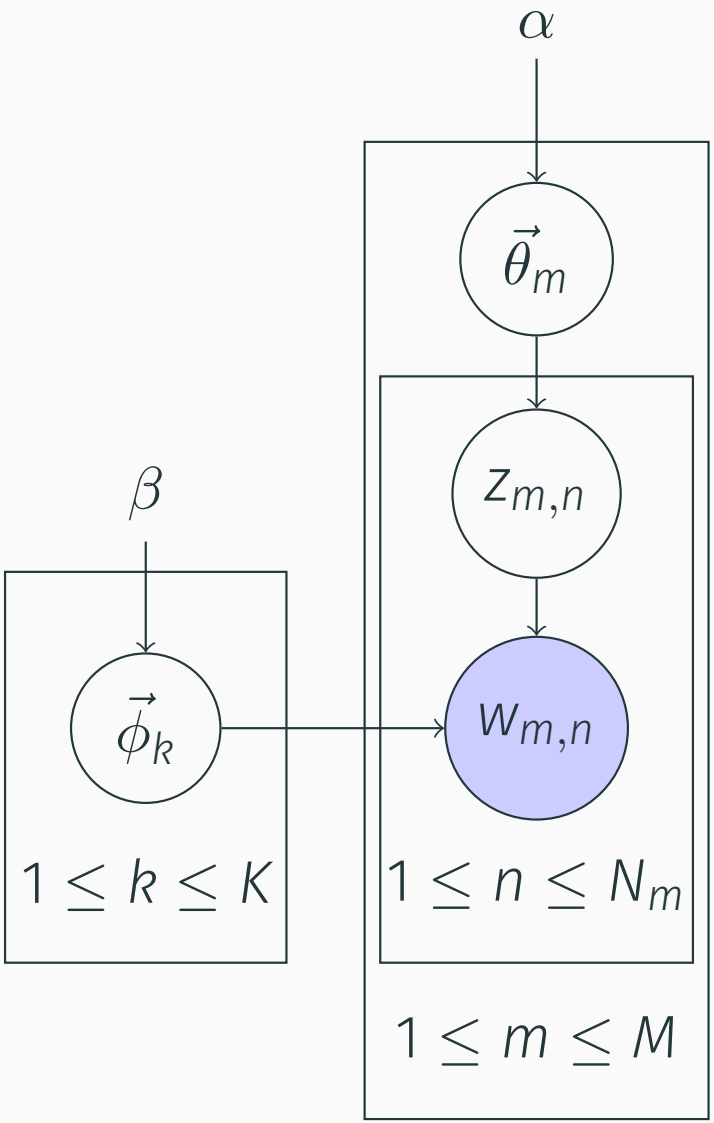
See example in Walker and Ni, SINUM 2011.

The Bayesian Approach: LDA

Latent Dirichlet Allocation (LDA) is a generative model:

- For each topic, choose word distribution $\vec{\phi}_k \sim \text{Dir}(\beta)$
- For each doc, choose topic distribution $\vec{\theta}_m \sim \text{Dir}(\alpha)$
- For word n in document m
  - Choose topic $z_{m,n} \sim \text{Cat}(\vec{\theta}_m)$
  - Choose word $w_{m,n} \sim \text{Cat}(\vec{\phi}_{z_{m,n}})$

The Bayesian Approach: LDA

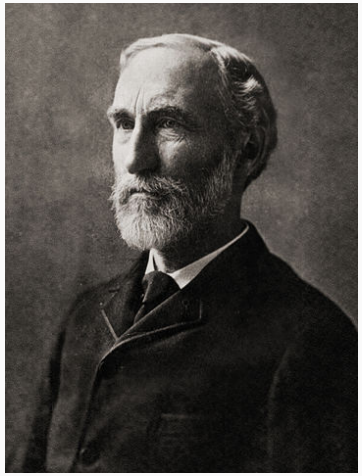


Graphical model notation:

- Circles = random variables
- Boxes = repetition
- Arrows = dependency
- Hyperparameters α, β

Observe $w_{m,n}$; infer the rest.

(Collapsed) Gibbs sampling for LDA



- Start with random assignment of topic labels $z_{m,n}$
- Compute counts (words per topic, topics per doc)
- Repeatedly loop over data; for each word $w_{m,n}$
  - Re-sample $z_{m,n}$ conditioned on all other $z_{m',n'}$
  - Update counts
- Infer $\vec{\phi}_k$ and $\vec{\theta}_m$ from samples of z

This looks a lot like coordinate descent!

The Picture So Far

- Lots of *locally* convergent iterations
- Slow convergence may be an issue?
  - Can still get useful results without guarantees!
  - Lee and Seung may not converge to stationary point
  - SGD methods are often terminated pretty early
  - Gibbs is useful because it sticks to a metastable state (otherwise permutation invariance blurs topic identity)
- All involve many passes over the data

One Stone, Two Birds

What makes us unhappy?

- Limited theory + frequent global convergence in practice
- Algorithms that require many passes over the data

A common solution: *separability*

Separable NMF

Separable structure:

$$\Pi^T A = \begin{bmatrix} I \\ W_2 \end{bmatrix} H$$

Happens in many domains!

Domain	Object ($a_{:,j}$)	Cluster ($w_{:,k}$)	Basis
Document	Word	Topic	Anchor word
Image	Pixel	Segment	Pure pixel
Network	User	Community	Representative
Legislature	Member	Party/Group	Partisan
Playlist	Song	Genre	Signature song

Finding Anchors

Assume separable structure:

$$\Pi^T A = \begin{bmatrix} I \\ W_2 \end{bmatrix} H$$

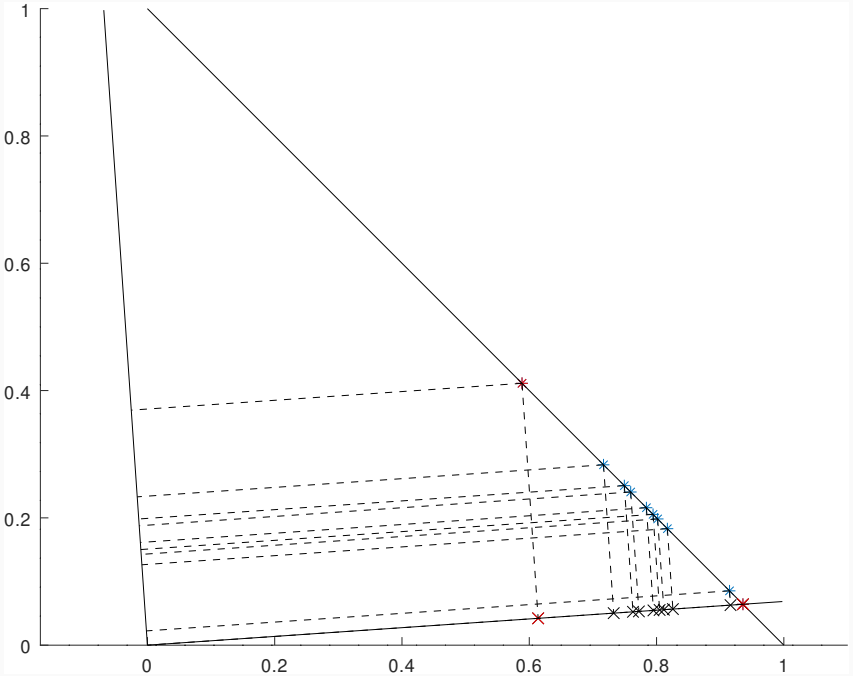
Normalize rows of A (rows of W, cols of H) to sum to 1:

$$\Pi^T \bar{A} = \begin{bmatrix} I \\ \bar{W}_2 \end{bmatrix} \bar{H}$$

Leading rows of $\Pi^T \bar{A}$ are the *convex hull* of all rows.

We have already seen how to find convex hulls!

Finding Anchors

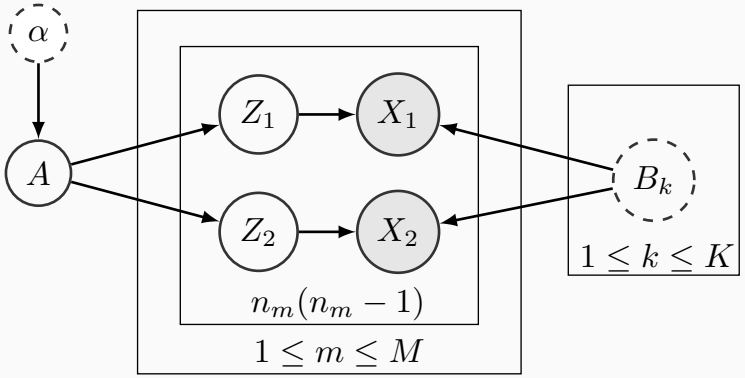


This is exactly the construction of ID via pivoted QR!

$$\bar{A}^T \Pi = QR = \bar{H}^T \begin{bmatrix} I & \bar{W}_2^T \end{bmatrix}$$

Recognizing pivoted QR lets us use better implementations.

Concentrate on Co-occurrence



Rest of the talk: focus on normalized co-occurrence matrix

$C_{ij} \propto$ frequency with which words i, j appear together

Note:

- Smaller (if less sparse) than word-document matrix
- Separability of word-document matrix inherited by C
- Topics tend to be easier to pull out
- Can produce a graphical model for C directly

Factoring the Co-occurrence

$$C \approx BAB^T$$

- Columns of B correspond to word distributions for topics
- Matrix A corresponds to topic correlation
- Entries of C are non-negative sum to 1 (joint stochastic)
- Entries of A are non-negative and sum to 1
- Columns of B are non-negative and sum to 1

The Anchor Words Algorithm

$$\Pi^T C \Pi \approx \begin{bmatrix} I \\ B_2 \end{bmatrix} A \begin{bmatrix} I & B_2^T \end{bmatrix}$$

- Find anchor words by pivoted QR on $\bar{C}$
- Simplex-constrained least squares to compute B_2
- Choose A as submatrix of C (can also do least squares)

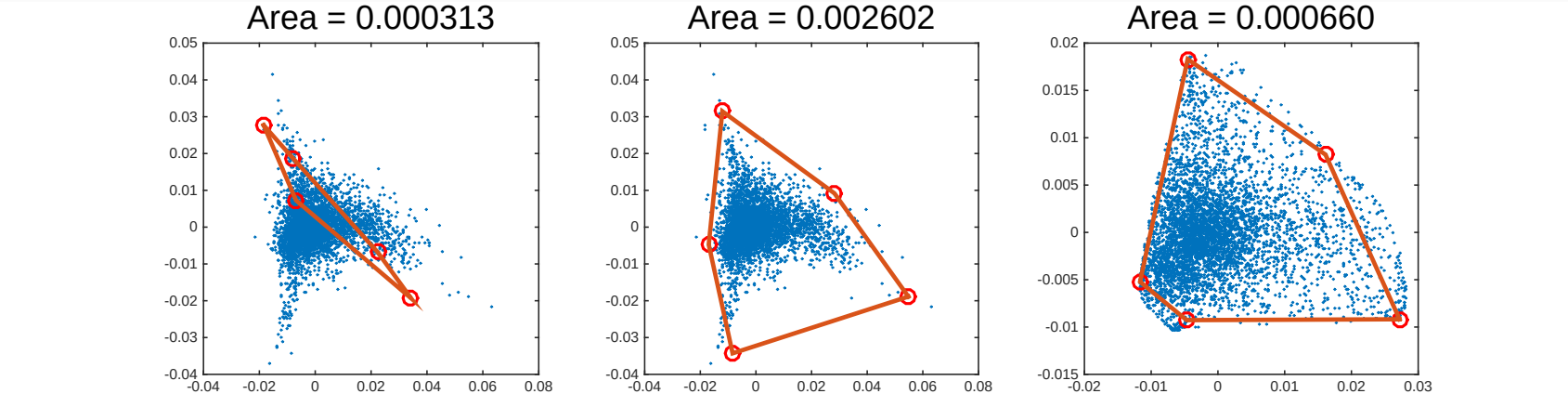
How Well Does It Work? (NeurIPS document collection)

Arora et al. 2013 (Baseline)
neuron layer hidden recognition signal cell noise
neuron layer hidden cell signal representation noise
neuron layer cell hidden signal noise dynamic
neuron layer cell hidden control signal noise
neuron layer hidden cell signal recognition noise
Probabilistic LDA (Gibbs)
neuron cell visual signal response field activity
control action policy optimal reinforcement dynamic robot
recognition image object feature word speech features
hidden net layer dynamic neuron recurrent noise
gaussian approximation matrix bound component variables

What Goes Wrong?

Not a conventional NeurIPS author.

What Goes Wrong?



- Naive anchor words algorithm finds bad anchors!
- Issue: Separability may only be approximate
- Data does not even look very low rank

Finding a Fix

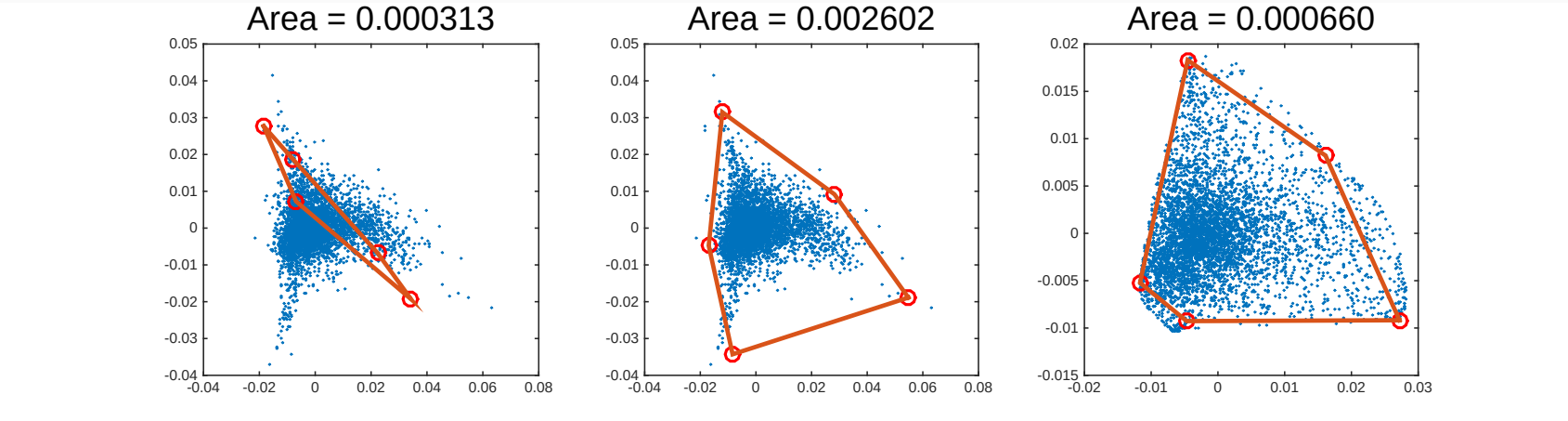
Model implicitly assumes C is

- Joint stochastic
- Positive semi-definite
- Low rank

Cannot fit outside this structure.

So: what if we enforce the structure?

Rectification



Alternating projection based on:

- Projection onto low-rank PSD (via eigensolve)
- Projection onto joint stochastic (elementwise)

Anchor selection based on rectified C works *much* better.

How Well Does It Work?

Lee et al. 2015 (AP)
neuron circuit cell synaptic signal layer activity
control action dynamic optimal policy controller reinforcement
recognition layer hidden word speech image net
cell field visual direction image motion object orientation
gaussian noise hidden approximation matrix bound examples
Probabilistic LDA (Gibbs)
neuron cell visual signal response field activity
control action policy optimal reinforcement dynamic robot
recognition image object feature word speech features
hidden net layer dynamic neuron recurrent noise
gaussian approximation matrix bound component variables

Summary

Three flavors of latent factor model:

- Latent factors via SVD are easy to compute
- Factors drawn from data are more interpretable
- NMF and LDA give natural parts decomposition

NMF is tricky in general, but

- Non-convexity sometimes seems a non-issue
- *Separability* hypothesis suggests why
- Assuming separability gives fast algorithms
- May need to enforce structure for good results

Recent approaches involve all three ideas: SVD/SEP, QRP, NMF.

Numerical Methods for Data Science: Spectral Network Analysis, Part I

David Bindel

21 June 2019

Department of Computer Science
Cornell University

Lecture plan

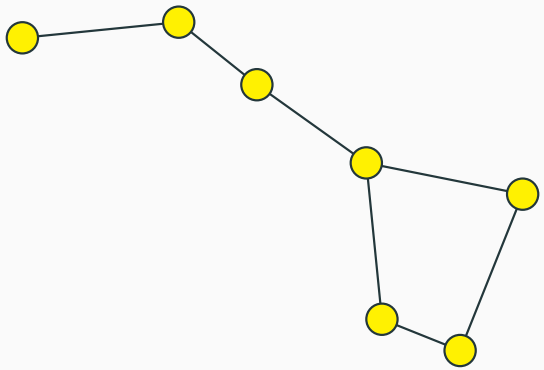
Three threads from “lay of the land” to current research:

- Monday: Latent Factor Models
- Wednesday: Scalable Kernel Methods
- Friday: Spectral Network Analysis
  - 1:30-2:30: Network spectra, optimization, and dynamics
  - 3:00-4:00: Network densities of states

Slides posted on web page (linked from my Cornell page).

Basics of Networks

Networks and Graphs

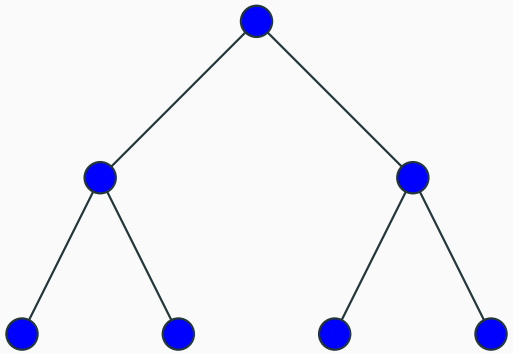


A graph (network) consists of

- Node (or vertex) set $\mathcal{V}$
- Edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$
  - *Undirected* if $(u, v) \in \mathcal{E} \implies (v, u) \in \mathcal{E}$
- Optional *edge weights* $\mathcal{E} \mapsto \mathbb{R}$

Can also add node weights or edge/node attributes.

Example Networks: Classic CS



Often small and/or highly structured:

- Finite state automata
- Search trees and DAGs
- Graphical models (correlated random variables)

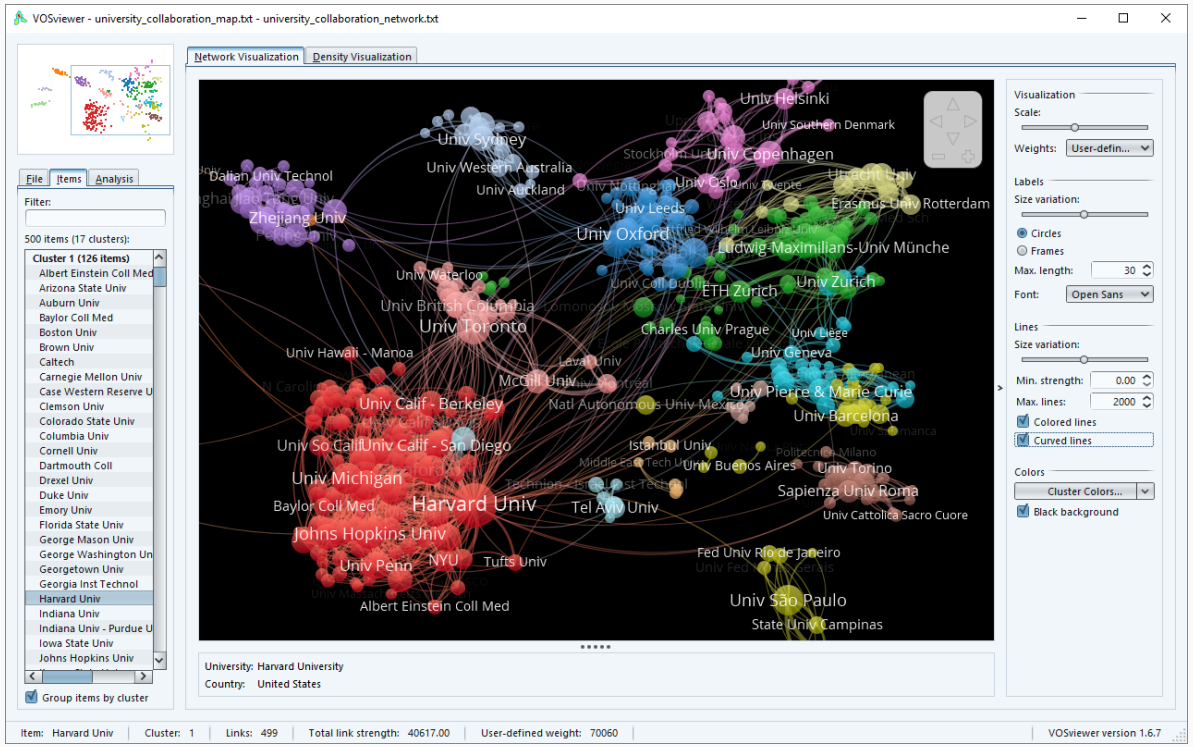
Mostly not the topic for today.

Example Networks: Physical

Connected to physical objects in 2D/3D:

- Rivers, roads, transportation
- Circuits and other electrical networks
- Pipe flow networks
- Computer networks

Example Networks: Citations



http://www.vosviewer.com/

Often directed, some very high-degree nodes, “small world”:

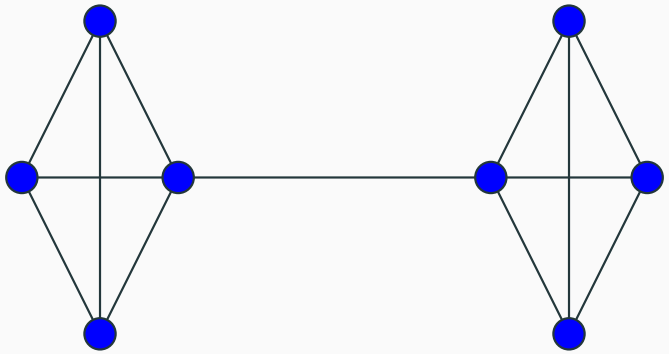
- Web pages, citation networks
- Purchase networks

Other Networks

Lots of others as well!

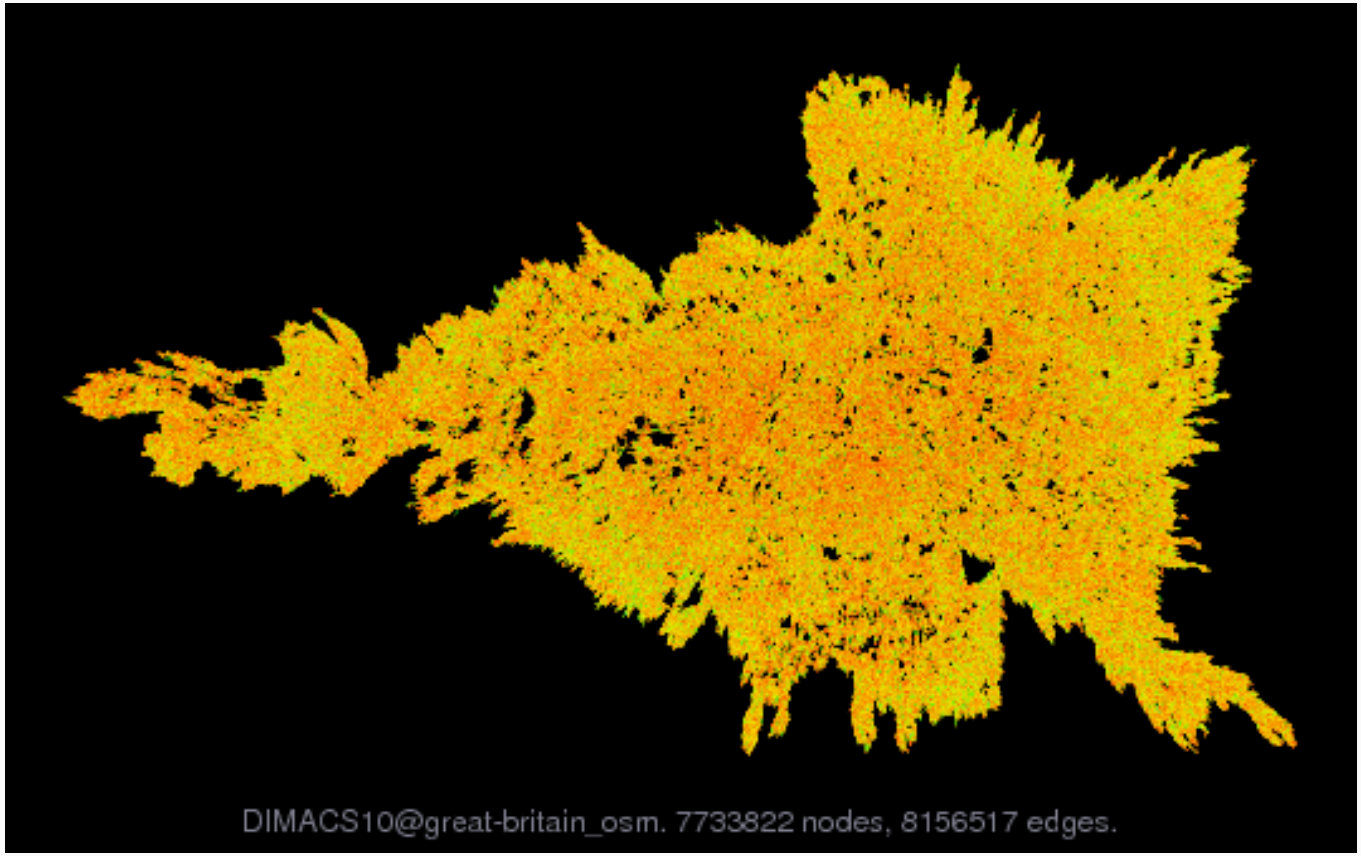
- Friendship networks
- Interaction networks (phone calls, etc)
- Food webs
- Protein interactions
- ...

The Big Questions: Connectedness



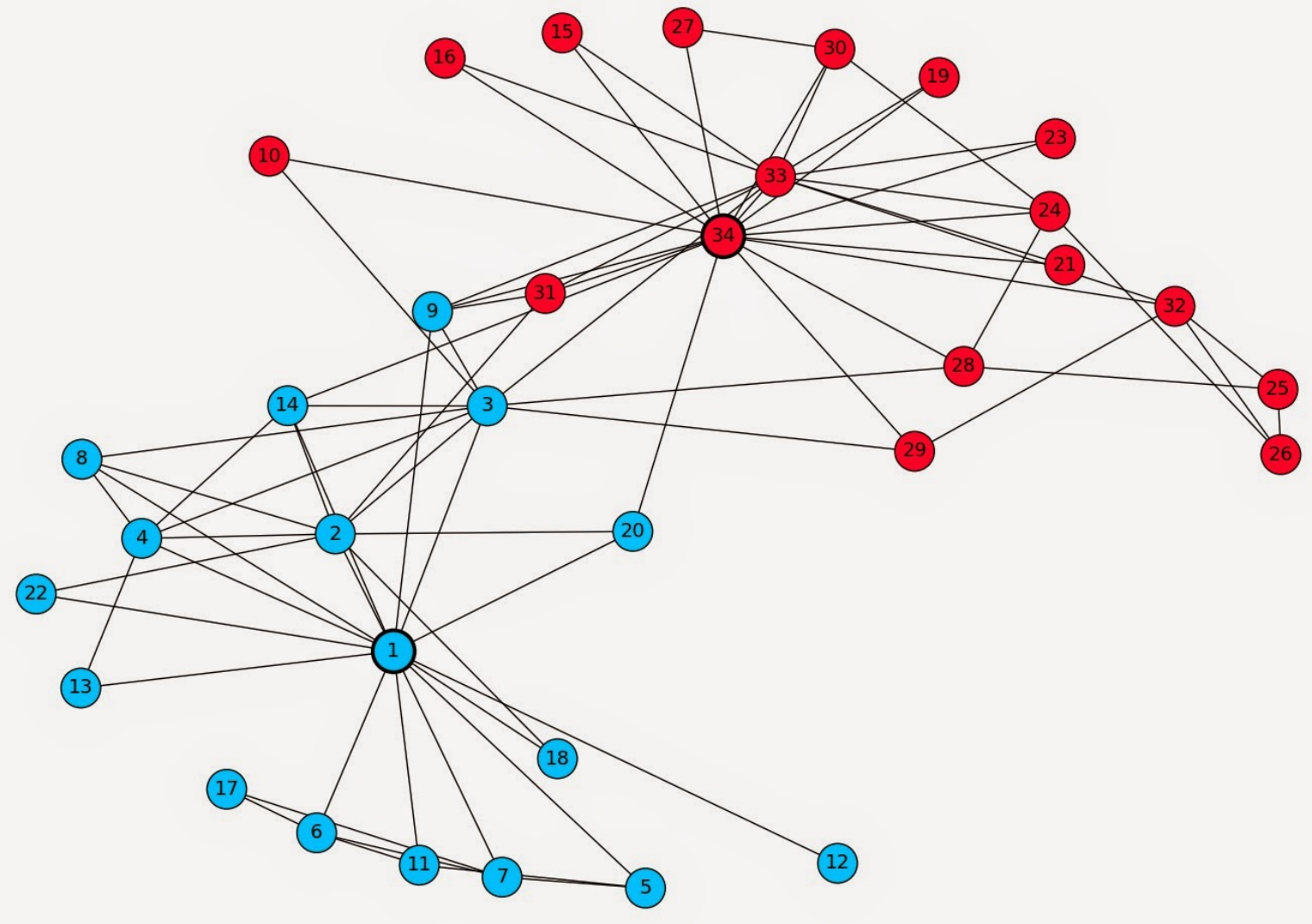
How “well-connected” is the network?

The Big Questions: Geometric Embedding



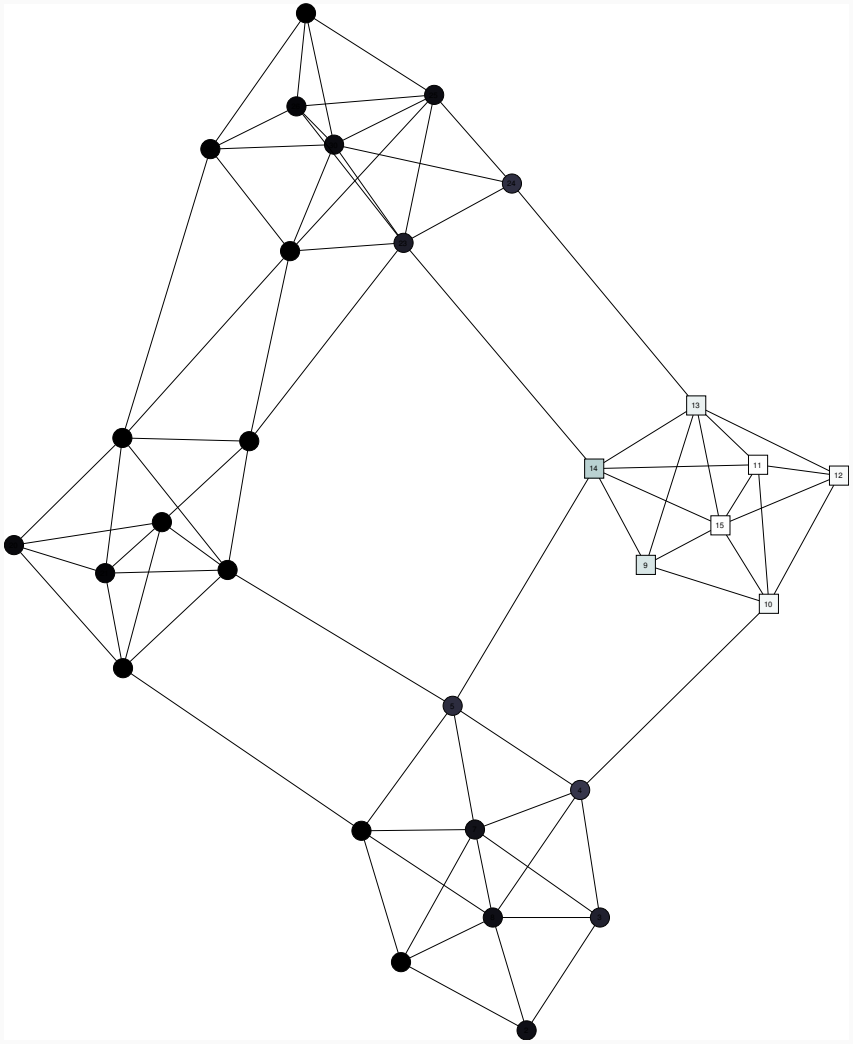
Is there an underlying geometry to the network?

The Big Questions: Centrality and Ranking



Who are important players?

The Big Questions: Clustering and Communities



What are the natural clusters or communities?

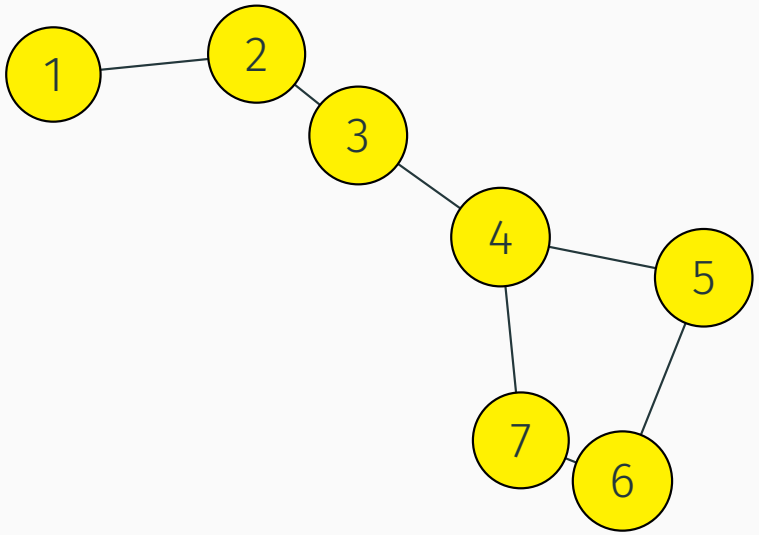
The Big Questions

One might ask many more questions:

- Graph alignment: Can we map between similar structures?
- Link prediction: Can we extrapolate the pattern?
- Cascade analysis: How does information spread?
- ...

Common approach: map to a linear algebra problem!

From Networks to Matrices



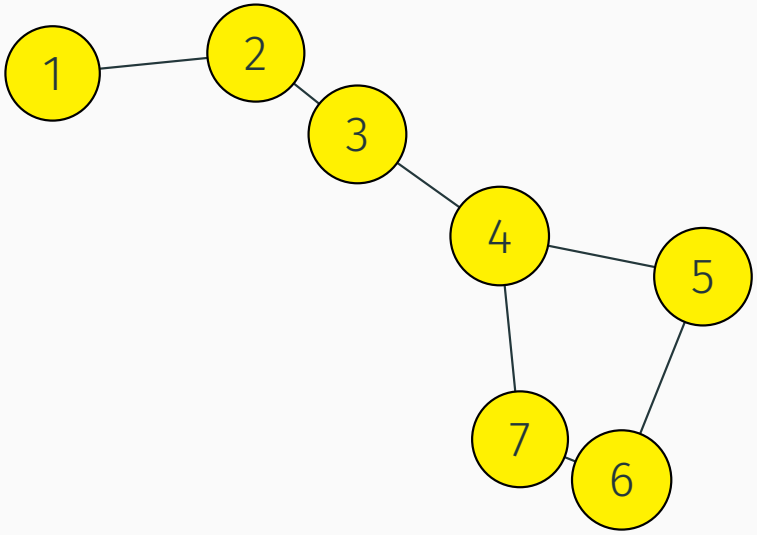
$$\begin{bmatrix} & 1 & & & & & \\ 1 & & 1 & & & & \\ & 1 & & 1 & & & \\ & & 1 & & 1 & & 1 \\ & & & 1 & & 1 & \\ & & & & 1 & & 1 \\ & & & 1 & & 1 & \end{bmatrix}$$

Adjacency A; unweighted is

$$a_{uv} = \begin{cases} 1, & (u, v) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}$$

Degree $d_u = \sum_v a_{uv}$ is total adjacent edges (edge weight).
Distinguish in/out in directed case.

From Networks to Matrices

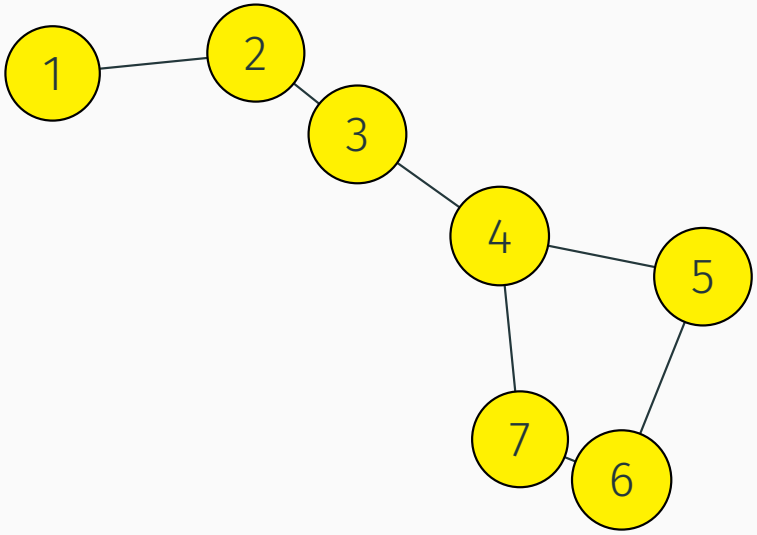


$$\begin{bmatrix} 1 & -1 & & & & & \\ & 1 & -1 & & & & \\ & & 1 & -1 & & & \\ & & & 1 & -1 & & \\ & & & & 1 & -1 & \\ & & & & & 1 & -1 \\ & & & -1 & & & 1 \end{bmatrix}$$

Differencing matrix (unweighted case)

$$g_{ew} = \begin{cases} 1, & e = (w, v) \\ -1, & e = (u, w) \\ 0, & \text{otherwise} \end{cases}$$

From Networks to Matrices



$$\begin{bmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 3 & -1 & & -1 \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & -1 & & -1 & 2 \end{bmatrix}$$

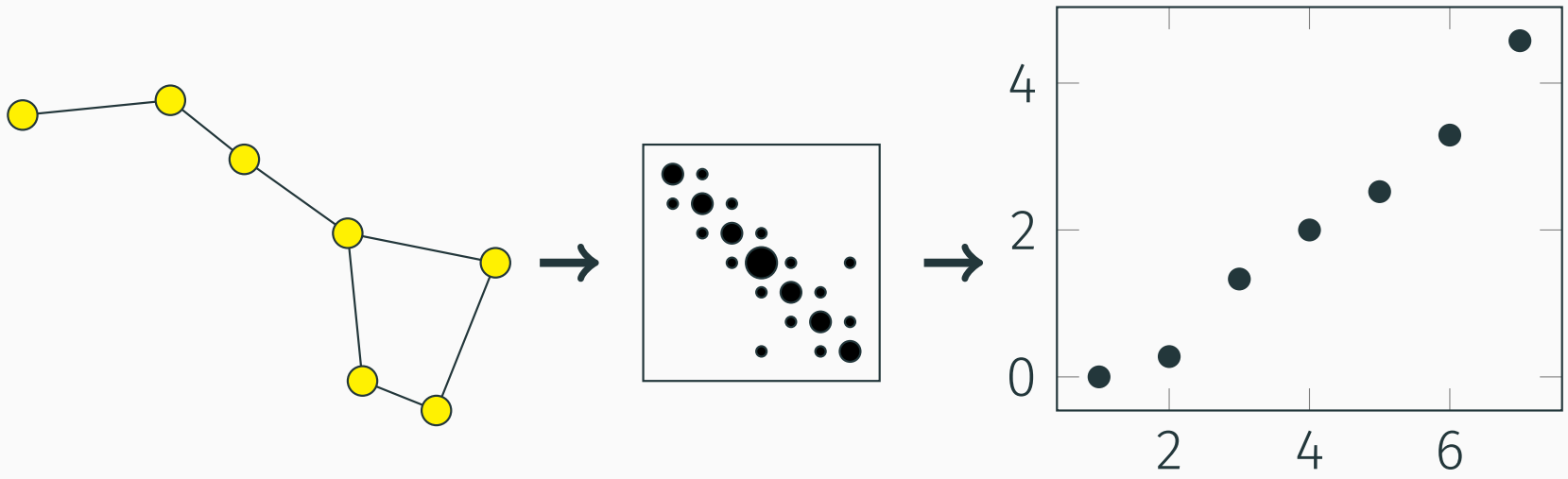
Laplacian $L = G^T G = D - A$; unweighted is

$$l_{uv} = \begin{cases} \text{degree } d_u, & u = v \\ -1, & (u, v) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}$$

A Bestiary of Matrices

- Adjacency matrix: A
- Laplacian matrix: $L = D - A$
- Unsigned Laplacian: $L = D + A$
- Random walk matrix: $P = AD^{-1}$ (or $D^{-1}A$)
- Normalized adjacency: $\bar{A} = D^{-1/2}AD^{-1/2}$
- Normalized Laplacian: $\bar{L} = I - \bar{A} = D^{-1/2}LD^{-1/2}$
- Modularity matrix: $B = A - \frac{dd^T}{2n}$
- Motif adjacency: $W = A^2 \odot A$

Spectral Network Analysis



Three stories of why eigenstuff matters:

- Dynamics and diffusion
- Measure and counting
- Kernels and geometry

Story 1: Dynamics and Diffusion

The Random Walker

Graph with adjacency A (a_{ij} denotes edge j to i),

$$p_{ij} = a_{ij}/d_j = \text{probability of step } j \to i$$

Let $w_j(t)$ denote probability a walker is at node j at time t; then

$$w(t+1) = Pw(t) = P^t w(0)$$

Equations for a discrete time Markov chain $\equiv$ power iteration

The Random Walker

Suppose P diagonalizable and the walk is ergodic. Then

$$P = V\Lambda V^{-1}, \quad P^t = V\Lambda^t V^{-1}$$

with $\lambda_1 = 1$ and $|\lambda_j| < 1$ for $j \neq 1$.

Let w^∞ denote the stationary distribution; then

$$\|P^t - (w^\infty)e/n^T\| \leq C|\lambda_2|^t$$

where e is the vector of all ones. Rate of convergence determined by second-largest eigenvalue modulus $|\lambda_2|$.

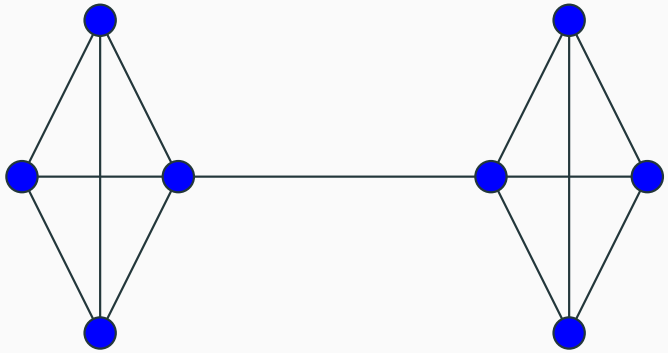
Random Walker to Random Surfer

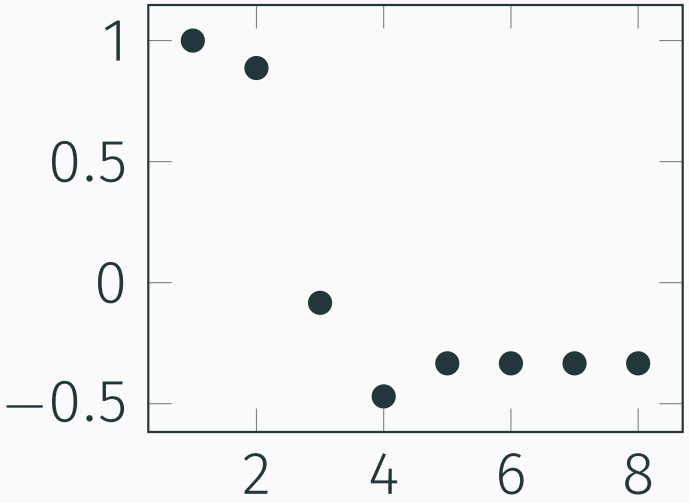
PageRank: Random walk + "teleport" with probability α

$$w(t+1) = (1-\alpha)Pw(t) + \alpha w^{\text{ref}}$$

Get a nice spectral gap $(1-\alpha)$, fast convergence to stationarity.

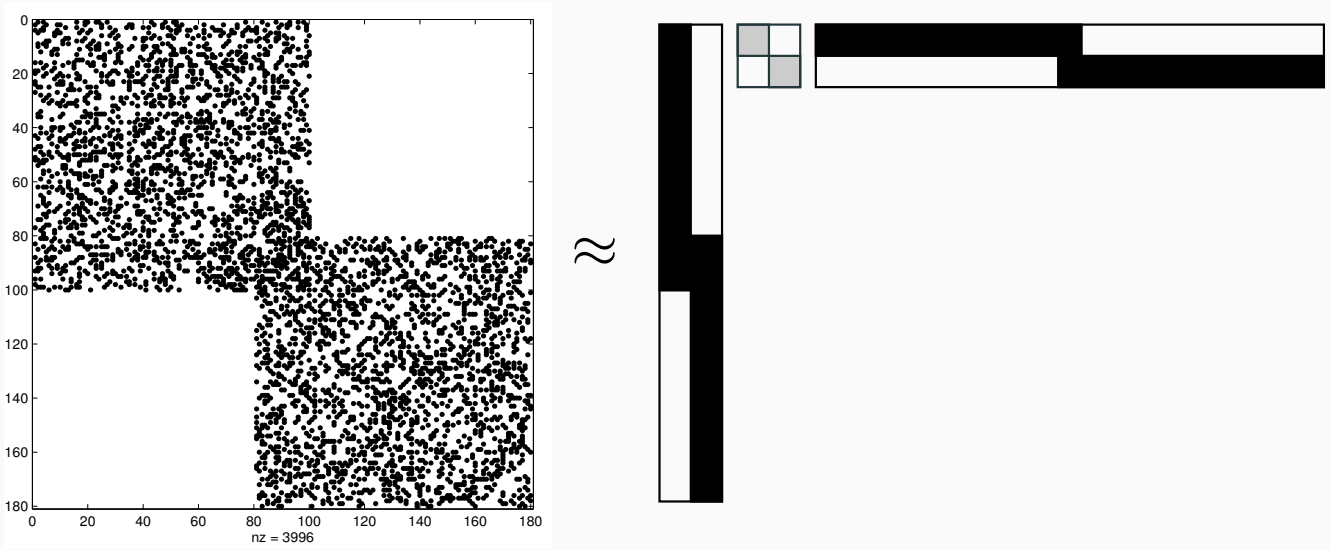
Convergence Fast and Slow





- Eigenvalue at 1: stationary state
- Size of $|\lambda_2|$ determines convergence
- What do the other eigenvalues mean?
- Interpreting two eigenvalues near 1 in dumbbell:
  - Distribution rapidly approaches 2D subspace initially
  - Then slower relaxation to equilibrium
- Corresponds to *metastable states* (c.f. Simon-Ando theory)

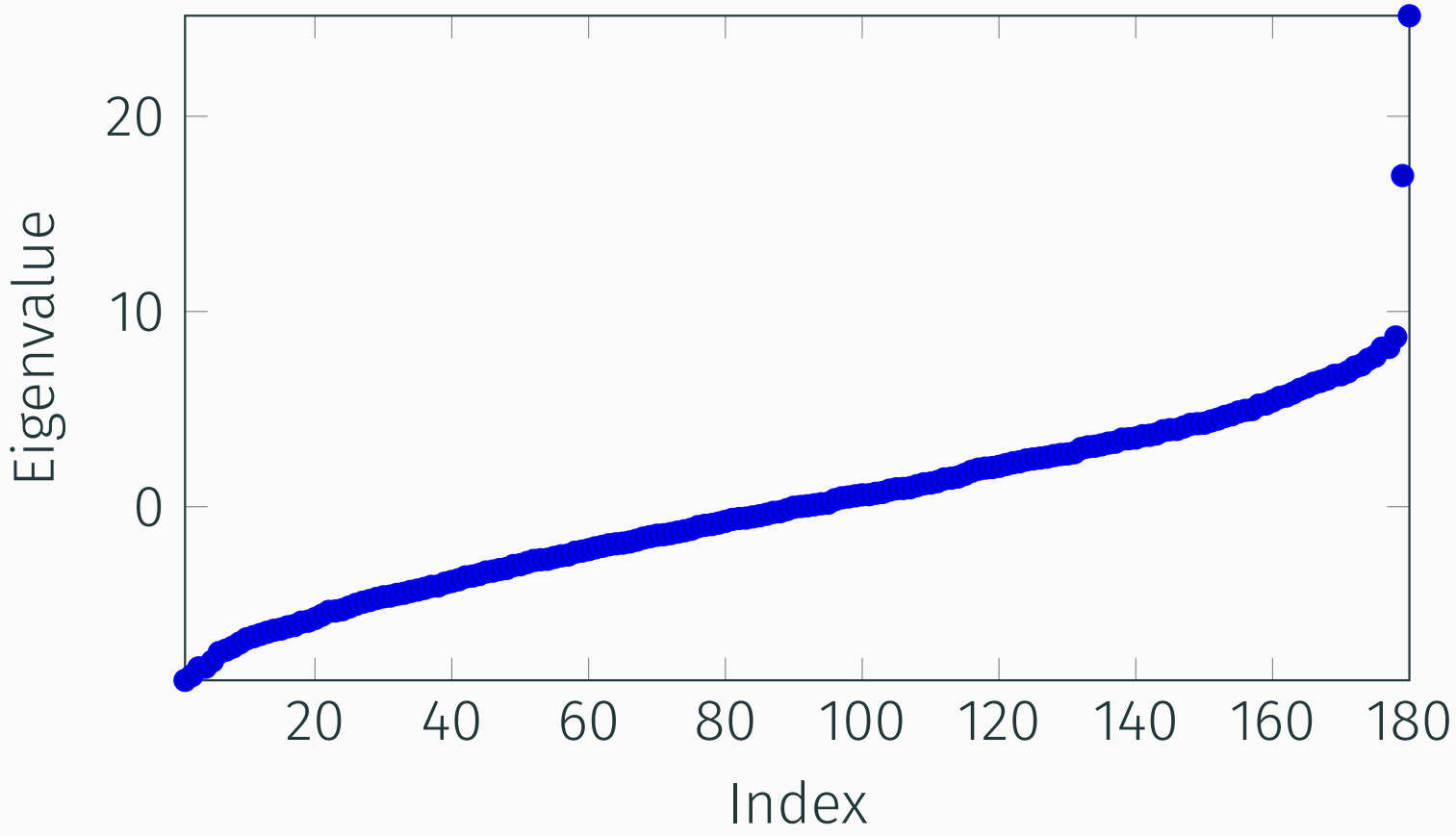
Dynamics in a Block Model



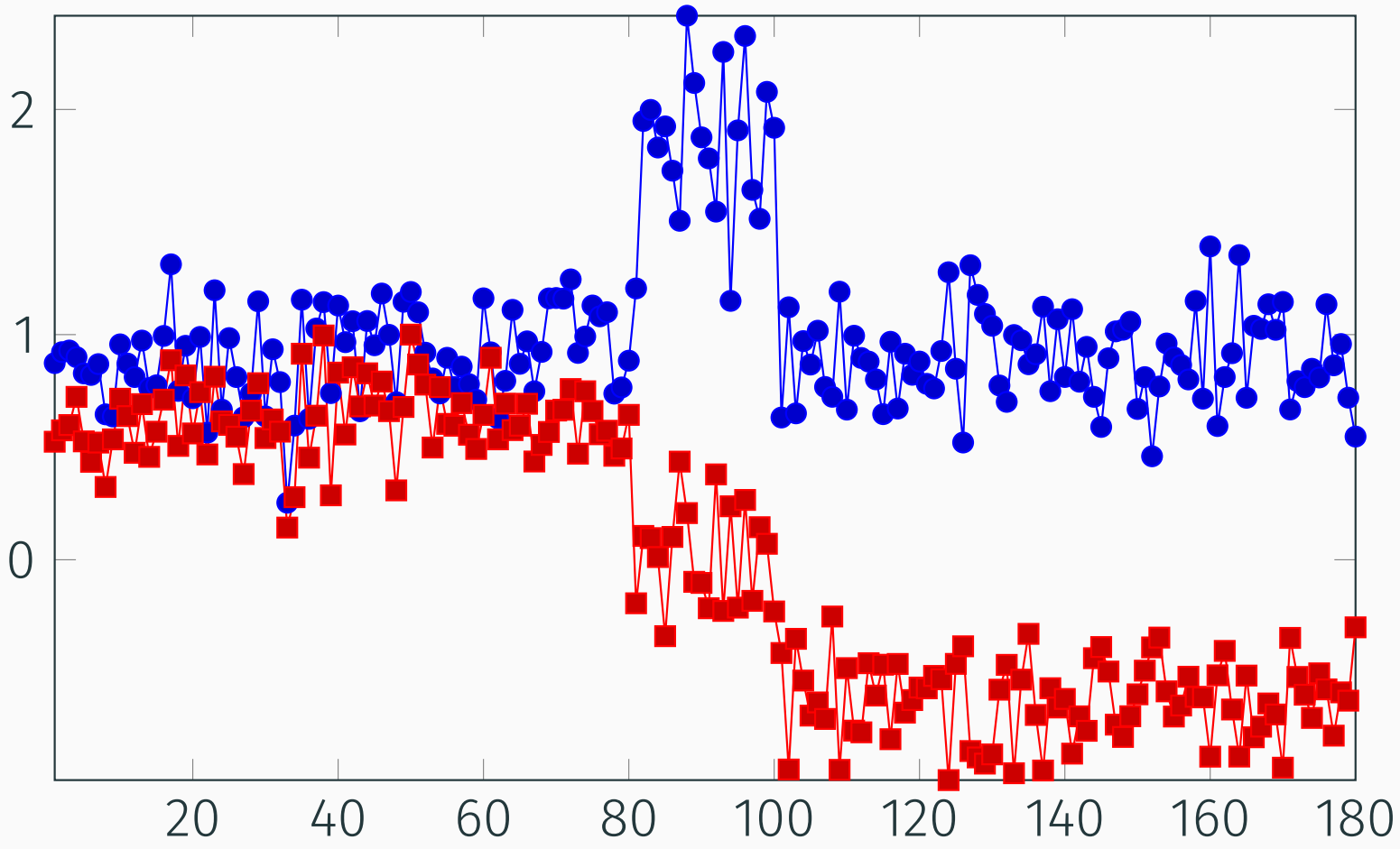
Composite model: $A \approx S\,\mathbf{diag}(\beta)S^T$, $S \in \{0,1\}^{n\times c}$

- Motivation: possibly-overlapping random graphs
- Columns of S are one basis for range space
- Want to go from some general basis back to S

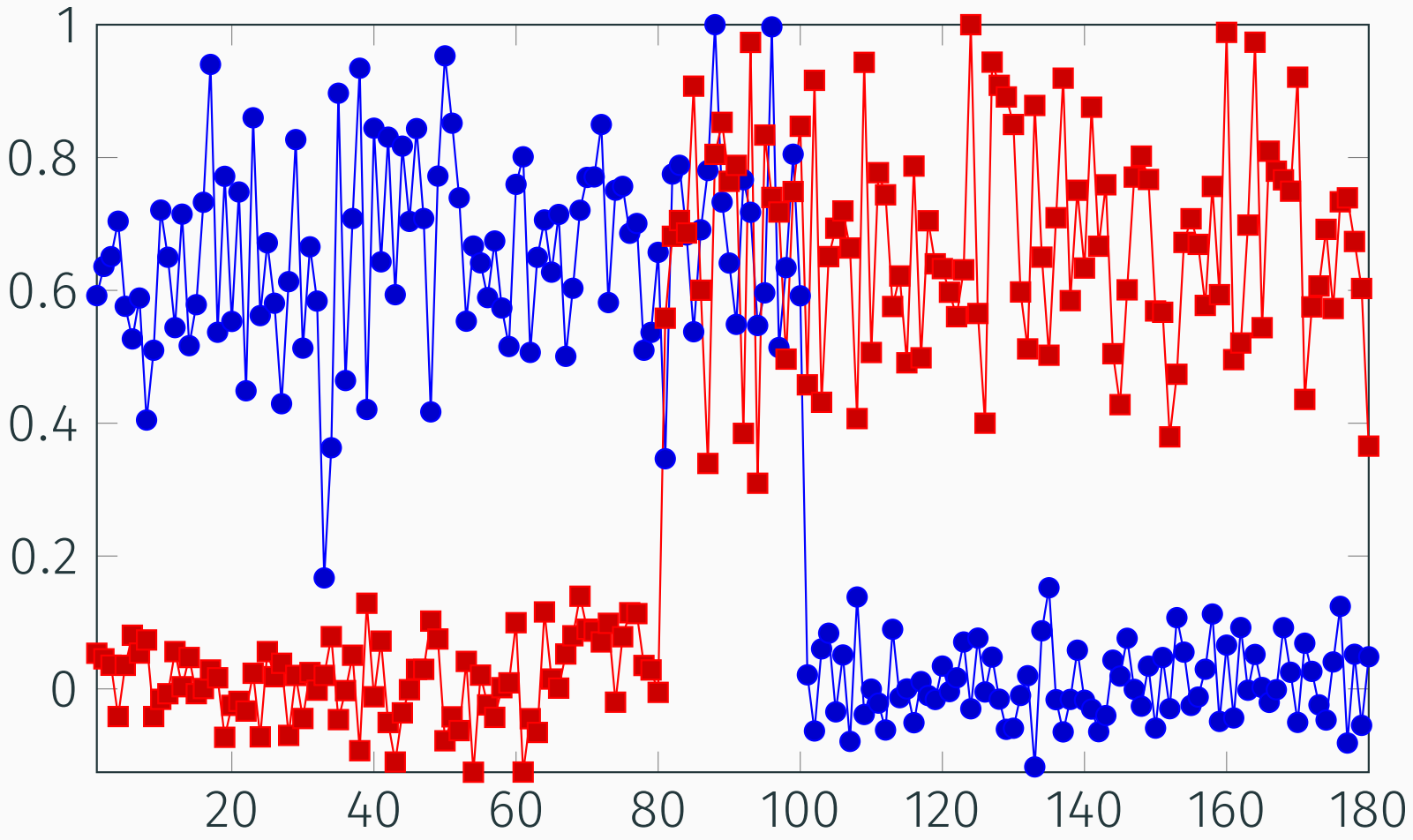
Spectrum for a Block Model Sample



Dominant Vectors



Same Space, Different Basis



And Beyond!

Many variations:

- Path counting and power of A
- Continuous time walks and $\exp(-tL)$
- Hub/authority importance, HITS, and SVD power iteration
- ...

All involve linear time-invariant systems.

Story 2: Measure and Counting

Measurement by Quadratic Forms

Indicate $V' \subseteq V$ by $s \in \{0,1\}^n$. Measure subgraph:

$$s^T A s = |E'| = \text{internal edges}$$
$$s^T D s = \text{edges incident on subgraph}$$
$$s^T L s = \text{edges between } V' \text{ and } \bar{V}'$$
$$s^T B s = \text{“surprising” internal edges}$$

Modularity matrix is $B = A - \frac{dd^T}{2n}$

Graph Bisection

Idea: Find $s \in \{0,1\}^n$ such that $e^T s = n/2$ to

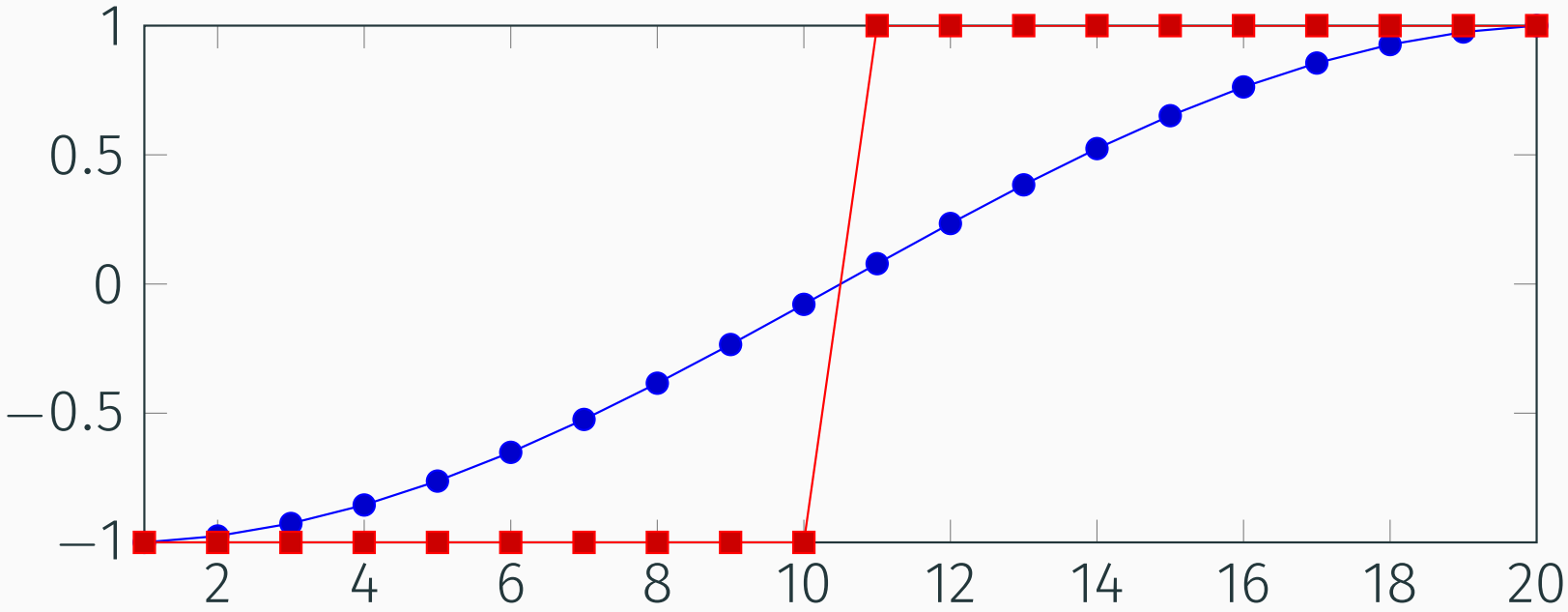
- minimize $s^T L s$ (min cut)
- maximize $s^T B s$ (max modularity)

Equivalently: Find $\bar{s} \in \{\pm 1\}^n$ such that $e^T \bar{s} = 0$ to

- minimize $\bar{s}^T L \bar{s} = s^T L s$ or
- maximize $\bar{s}^T B \bar{s} = s^T B s$

Oops — NP hard!

Relax!



Hard: $\min \bar{s}^T L \bar{s} \quad \text{s.t.} \quad e^T \bar{s} = 0, \quad \bar{s} \in \{\pm 1\}^n.$

Easy: $\min v^T L v \quad \text{s.t.} \quad e^T v = 0, \quad v \in \mathbb{R}^n, \|v\|^2 = n.$

Cheeger Inequality

Relaxation gives half of *Cheeger inequality*

$$h(G) \geq \lambda_2 \geq \frac{h^2(G)}{2}$$

where

$$h(G) = \min \left\{ \frac{|\partial A|}{|A|} : A \subset \mathcal{V}, 0 < |A| \leq \frac{1}{2}|\mathcal{V}| \right\}$$

Relation between bottlenecks and λ_2.

Also relevant to mixing picture from dynamics story!

Rayleigh Quotients

$$\frac{s^T A s}{s^T s} = \text{mean internal degree in subgraph}$$

$$\frac{s^T L s}{s^T s} = \text{edges cut between } V' \text{ and } \bar{V}'$$

$$\frac{s^T A s}{s^T D s} = \text{fraction of incident edges internal to } V'$$

$$\frac{s^T L s}{s^T D s} = \text{fraction of incident edges cut}$$

$$\frac{s^T B s}{s^T s} = \text{mean “surprising” internal degree in subgraph}$$

$$\frac{s^T B s}{s^T D s} = \text{mean fraction of internal degree that is surprising}$$

Rayleigh Quotients and Eigenvalues

Basic connection (M spd):

$$\frac{x^T K x}{x^T M x} \text{ stationary at x} \qquad \Longleftrightarrow \qquad Kx = \lambda M x$$

Easy despite lack of convexity.

Limits of Rayleigh Quotients

But small variations kill us:

$$\max_{x \neq 0} \frac{x^T A x}{\|x\|_2^2} = \lambda_{\max}(A), \text{ but}$$

$$\max_{x \neq 0} \frac{x^T A x}{\|x\|_1^2} = 1 - \omega^{-1}$$

where ω is the max clique size (Motzkin-Strauss).

Rayleigh Quotients and Eigenproblems

Decompose:

$$W^T M W = I \text{ and } W^T K W = \Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_n).$$

For any $x \neq 0$,

$$\frac{x^T K x}{x^T M x} = \sum_{j=1}^{n} \lambda_j z_j^2, \text{ where } z = \frac{W^{-1} x}{\|W^{-1} x\|_2}.$$

So

$$\frac{s^T K s}{s^T M s} \approx \lambda_{\max} \implies s \approx \sum_{\lambda_j \approx \lambda_{\max}} w_j z_j.$$

So look at invariant subspaces for extreme eigenvalues.

Story 3: Kernels and Geometry

Classical Multi-Dimensional Scaling

Data: square distances between points:

$$
\begin{aligned}
[D^{(2)}]_{ij} &= d_{ij}^2 = \|u_i - u_j\|^2 \\
&= \|u_i\|^2 - 2\langle u_i, u_j\rangle + \|u_j\|^2 \\
r_i &\equiv \|u_i\|^2 \\
D^{(2)} &= r^{(2)} e^T - 2UU^T + e(r^{(2)})^T
\end{aligned}
$$

Double centering:

$$
B = -\frac{1}{2} J D^{(2)} J = UU^T, \quad J \equiv I - \frac{1}{n} ee^T
$$

Decompose $B = XX^T$ (e.g. via eigs) to get coordinates for data.
Note that B looks like a kernel matrix / Gram matrix.

Resistance Distance

Think of resistors on each edge, consider net flow between source at node i and sink at node j:

$$d_{ij}^2 = (e_i - e_j)L^\dagger(e_i - e_j)$$

This acts like a squared distance — find coordinates!

$$L = V\Lambda V^T \quad \Longrightarrow \quad L^\dagger = V \begin{bmatrix} 0 & & & \\ & \lambda_2^{-1} & & \\ & & \ddots & \\ & & & \lambda_n^{-1} \end{bmatrix} V^T$$

Truncate to get a low-dimensional embedding.

Leveraging Geometry

General idea: M a kernel matrix / Gram matrix over data

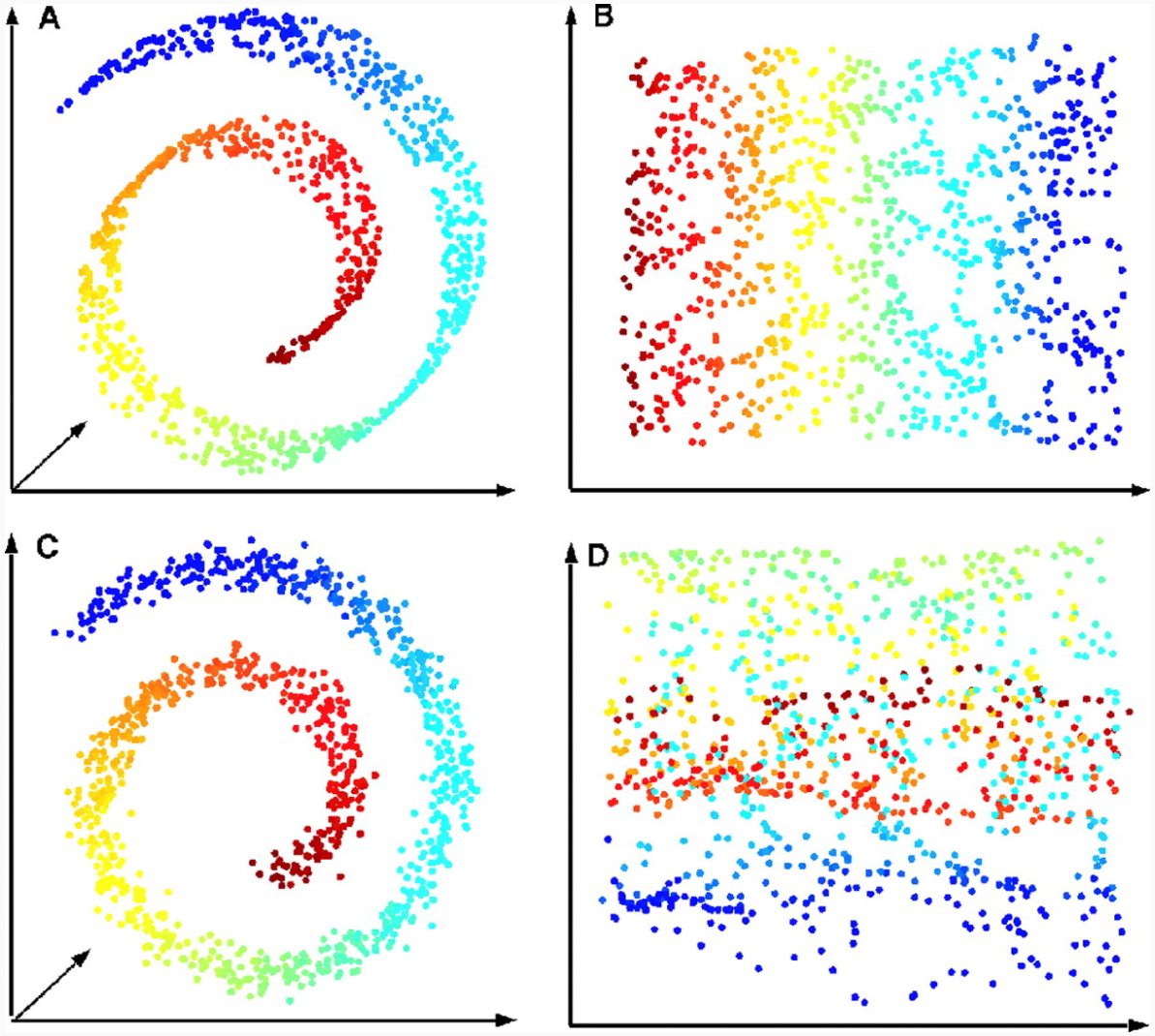
- Eigendecomposition: $M = V\Lambda V^T$
- Coordinates for i: $(v_{ij}\sqrt{\lambda_j})_{j=1}^n$
- Truncate to get low-dimensional embedding

Then do processing over geometry

- Graph layout (use as node coordinates)
- Geometric partitioning (e.g. inertial methods)
- Geometric clustering (k-means)

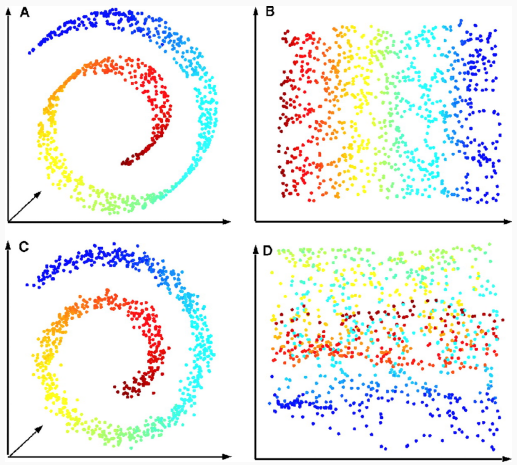
Other than resistance distance, useful metrics?

Beyond Resistance



What if we don't believe global geometry?

Isomap Idea



Distance matrix for points:

- Ordinary pairwise distance nearby
- Graph distance far away

And then apply MDS.

Summary and Preview

The Three Stories

Three stories of why eigenstuff matters:

- Dynamics and diffusion
- Measure and counting
- Kernels and geometry

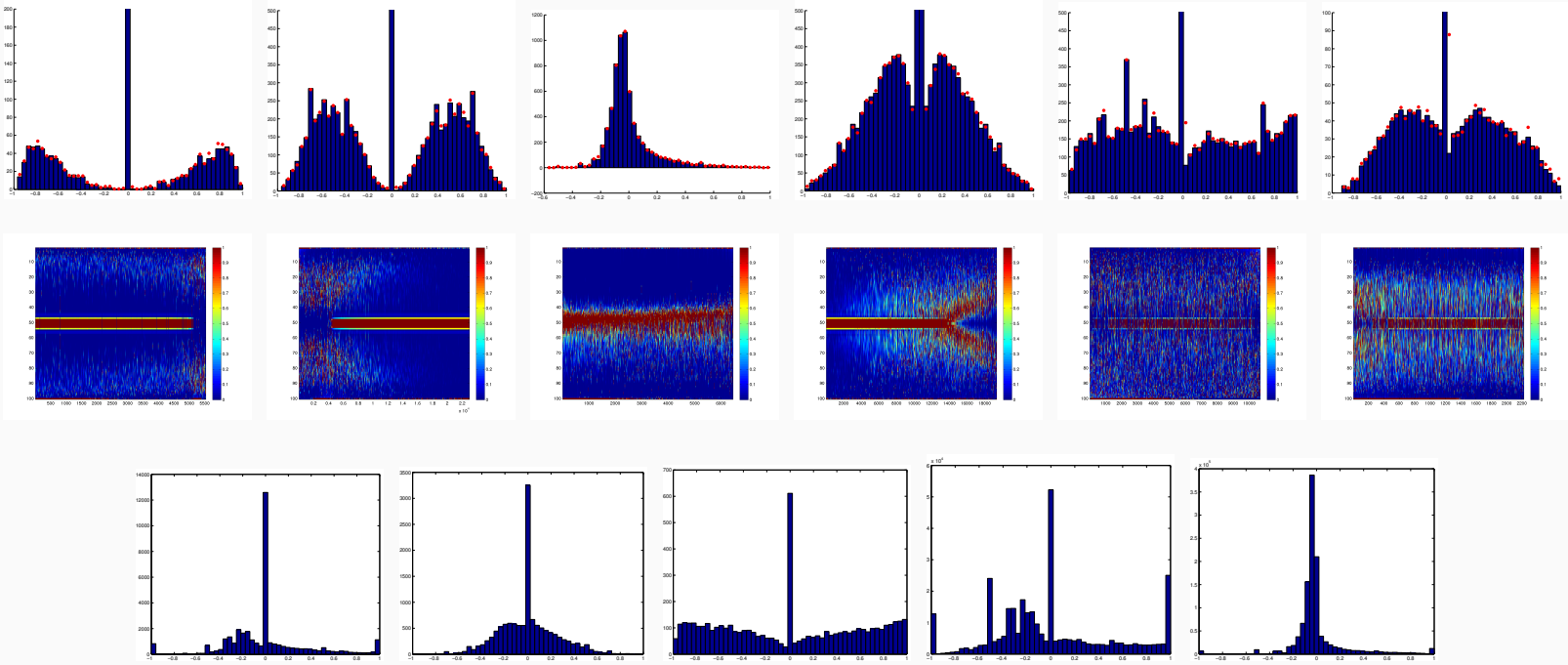
The Missing Link

So far, focus is the *extreme* eigenpairs:

- Metastable states in dynamics story
- Relaxed solutions in measure/counting story
- Reconstructed coordinates in geometric story

What about the interior of the spectrum?

The Missing Link



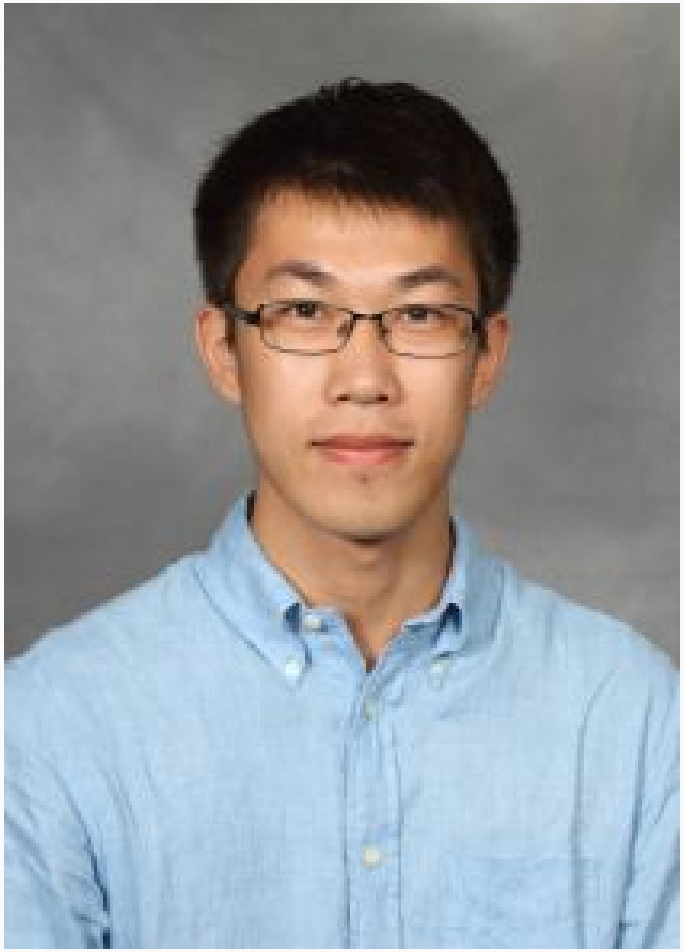
Numerical Methods for Data Science: Spectral Network Analysis, Part II

David Bindel

21 June 2019

Department of Computer Science
Cornell University

Acknowledgements



Thanks **Kun Dong** (Cornell CAM), along with Anna Yesypenko, Moteleolu Onabajo, Jianqiu Wang.

Also: NSF DMS-1620038.

Mathematicians are like Frenchmen: whatever you say to them they translate into their own language and forthwith it is something entirely different.

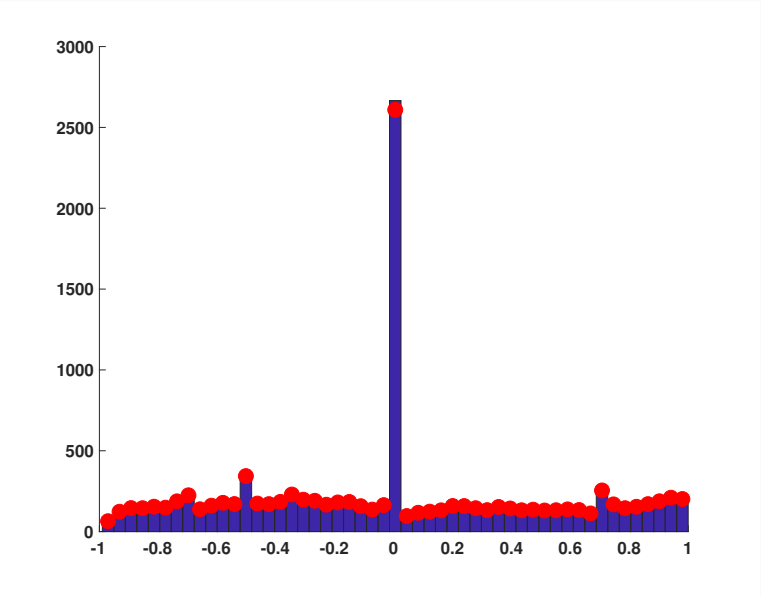
— Goethe

Stories of Spectral Geometry/Graph Theory

- Dynamics (operator on functions over manifold or graph)
  - Example: Continuous time diffusion + Simon-Ando theory
  - Diffuse according to heat kernel $\exp(-tLD^{-1})$
  - Rapid mixing + slow equilibration across bottlenecks
  - Regions of rapid mixing via extreme eigenvectors
- Counting and measure (quadratic form)
  - Example: Spectral partitioning
  - Measure cut sizes via $x^T Lx/4$, $x \in \{\pm 1\}^n$, relax to $\mathbb{R}^n$
  - $\lambda_2(L)$ bounds cuts (Cheeger), partition with Fiedler vector
- Geometric embedding (pos semidef form / kernel)
  - Example: Spectral coordinates via $R = L^\dagger$
  - Diffusion distances: $d_{ij}^2 = r_{ii}^2 - 2r_{ij} + r_{jj}^2$
  - First few eigenvectors of R give coordinates that approximate distance

Eigenvalues Two Ways

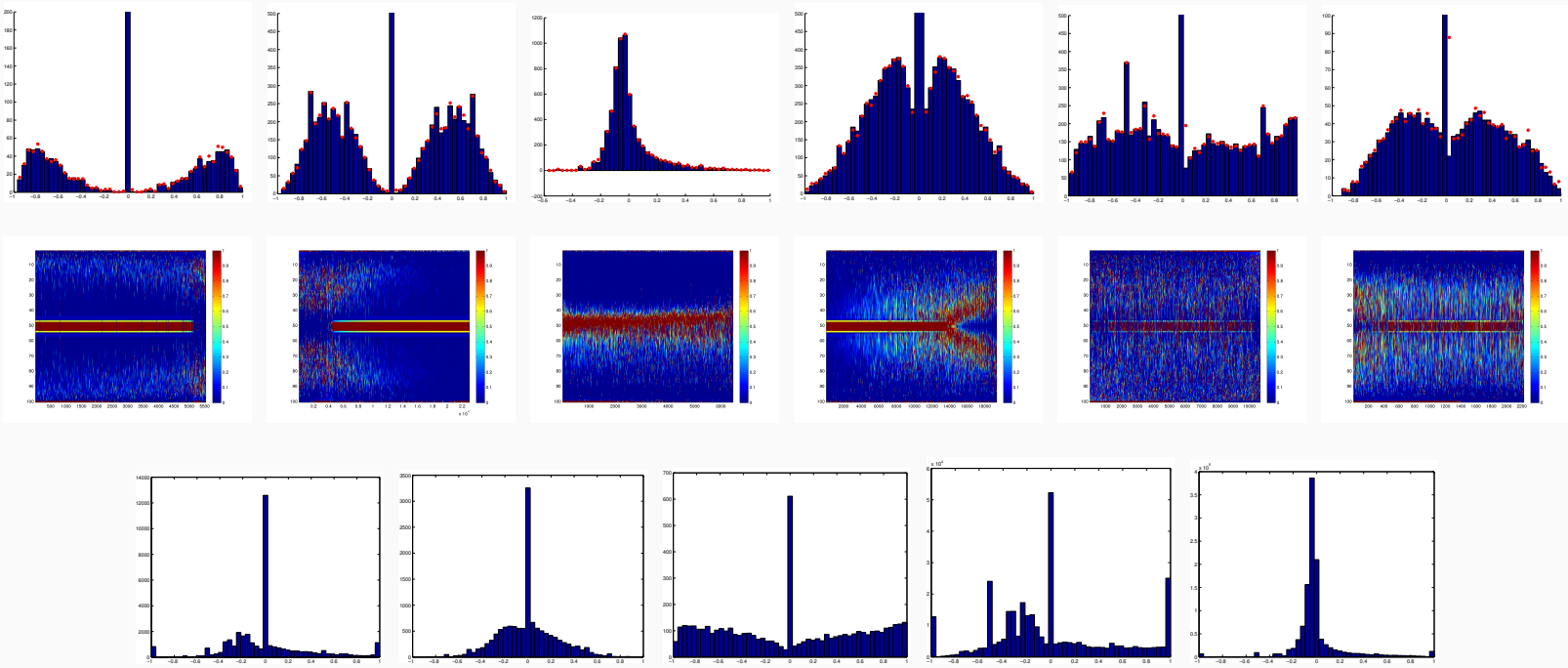
What can we tell from *partial* spectral information (eigenvalues and/or vectors)



Claim: Most spectral analyses involve one of two perspectives:

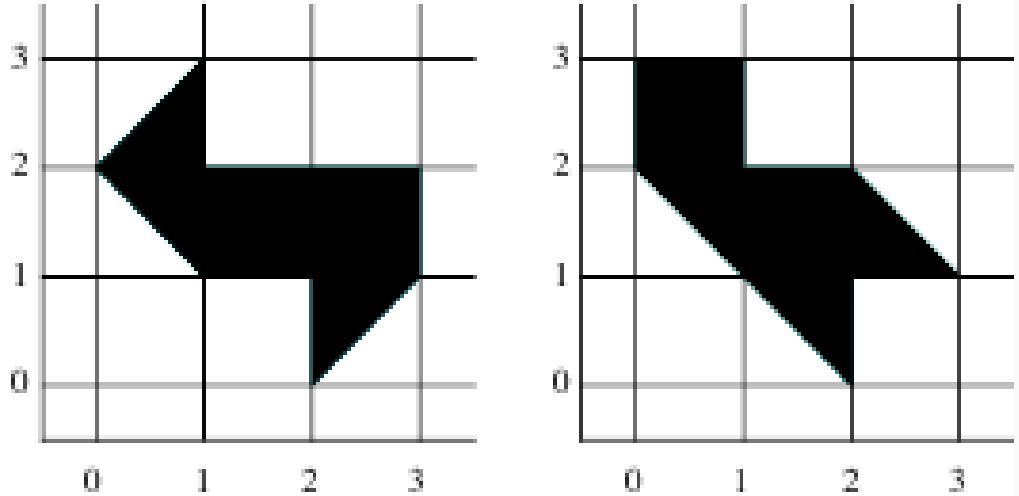
- Approximate something via a *few* (extreme) eigenvalues.
- Look at *all* the eigenvalues (or all in a range).

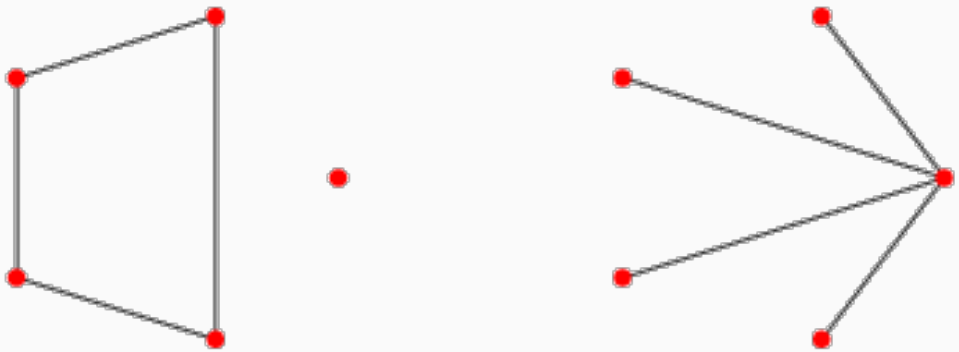
Today in Three Acts



- **Act 1: Spectral densities**
- Act 2: Algorithms and approximations
- Act 3: Spectral densities of “real world” graphs

Can One Hear the Shape of a Drum?

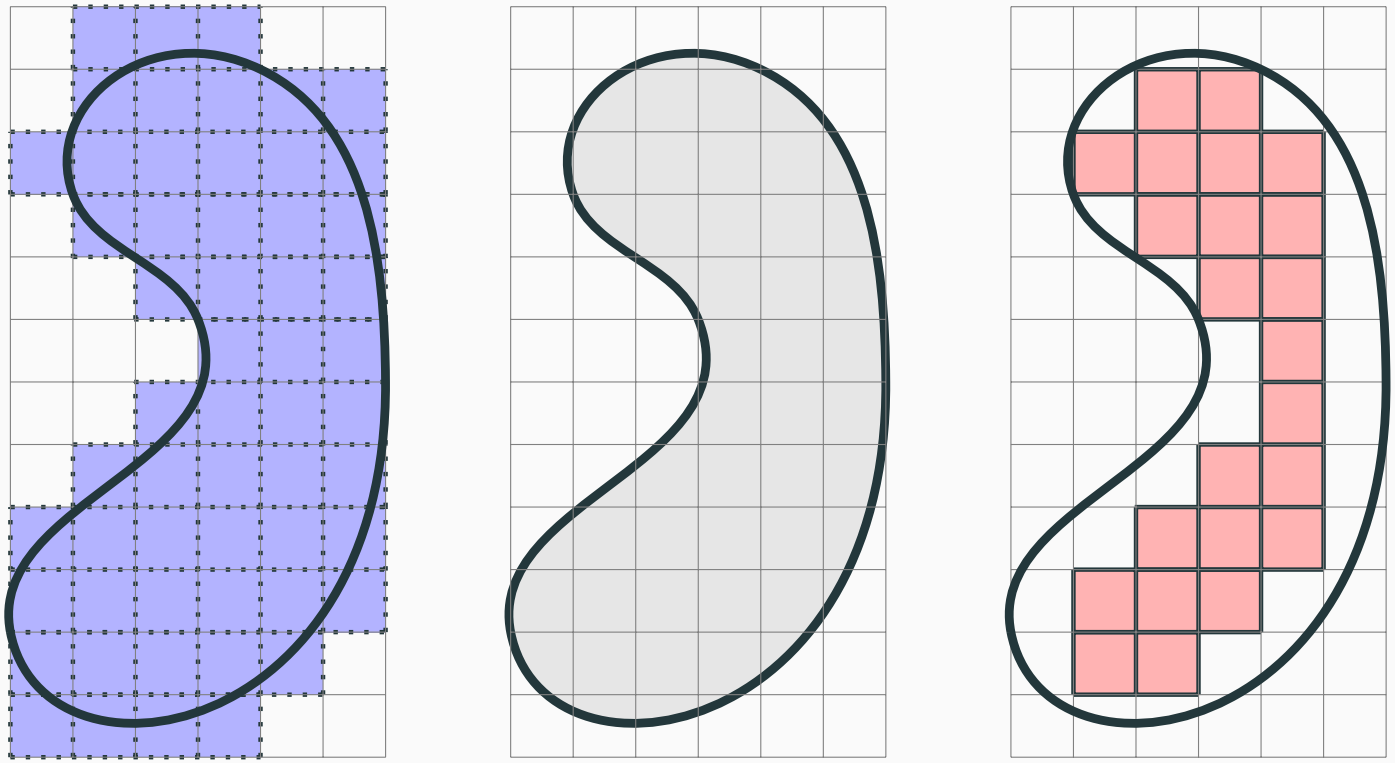




"You mean, if you had perfect pitch could you find the shape of a drum." — Mark Kac (quoting Lipmann Bers)

American Math Monthly, *1966*

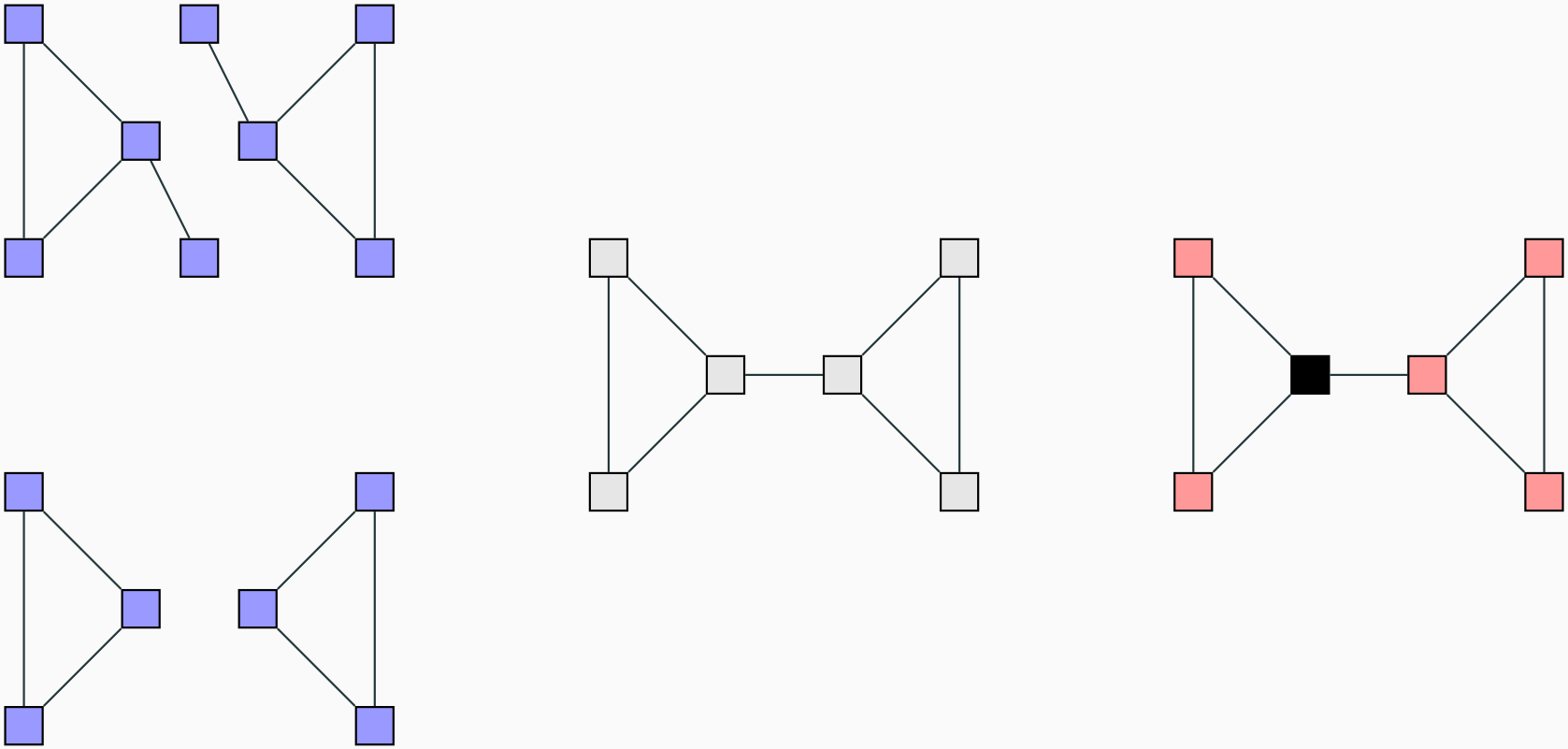
What Do You Hear?



Use $\mathcal{H}_{\mathrm{lo}} \supset \mathcal{H} \supset \mathcal{H}_{\mathrm{hi}}$ to get $\lambda_{k,\mathrm{lo}} \leq \lambda_k \leq \lambda_{k,\mathrm{hi}}$

$$\lambda_k = \min_{\dim(\mathcal{V})=k, \mathcal{V}\subset\mathcal{H}} \left(\max_{v\in\mathcal{V}} \rho_L(v) \right)$$

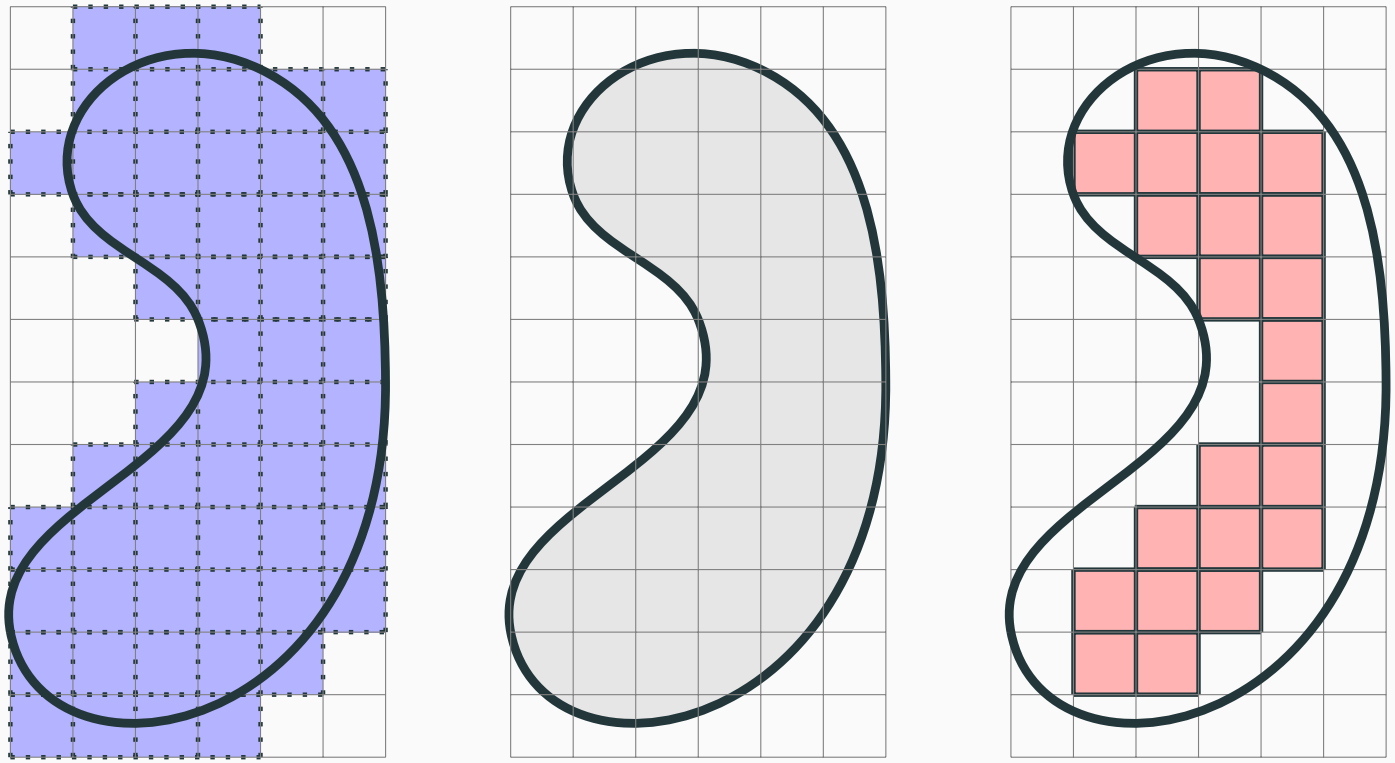
What Do You Hear?



Use $\mathcal{H}_{\mathrm{lo}} \supset \mathcal{H} \supset \mathcal{H}_{\mathrm{hi}}$ to get $\lambda_{k,\mathrm{lo}} \leq \lambda_k \leq \lambda_{k,\mathrm{hi}}$

$$\lambda_k = \min_{\dim(\mathcal{V})=k, \mathcal{V}\subset\mathcal{H}} \left(\max_{v\in\mathcal{V}} \rho_L(v) \right)$$

What Do You Hear?



Weyl law (asymptotics for $N(x) = \{\#\text{ eigenvalues } \leq x\}$):

$$\lim_{x\to\infty} \frac{N(x)}{x^{d/2}} = (2\pi)^{-d}\omega_d \,\mathrm{vol}(\Omega).$$

What Do You Hear?

What information hides in the eigenvalue distribution?

1. Discretizations of Laplacian: something like Weyl's law
2. Sparse E-R random graphs: Wigner semicircular law
3. Some other random graphs: Wigner semicircle + a bit (Farkas *et al*, Phys Rev E (64), 2001)
4. "Real" networks: less well understood

But computing all eigenvalues seems *expensive*!

A Bestiary of Matrices

- Adjacency matrix: A
- Laplacian matrix: $L = D - A$
- Unsigned Laplacian: $L = D + A$
- **Random walk matrix**: $P = AD^{-1}$ (or $D^{-1}A$)
- **Normalized adjacency**: $\bar{A} = D^{-1/2}AD^{-1/2}$
- **Normalized Laplacian**: $\bar{L} = I - \bar{A} = D^{-1/2}LD^{-1/2}$
- Modularity matrix: $B = A - \frac{dd^T}{2n}$
- Motif adjacency: $W = A^2 \odot A$

All have examples of co-spectral graphs

... through spectrum uniquely identifies *quantum graphs*

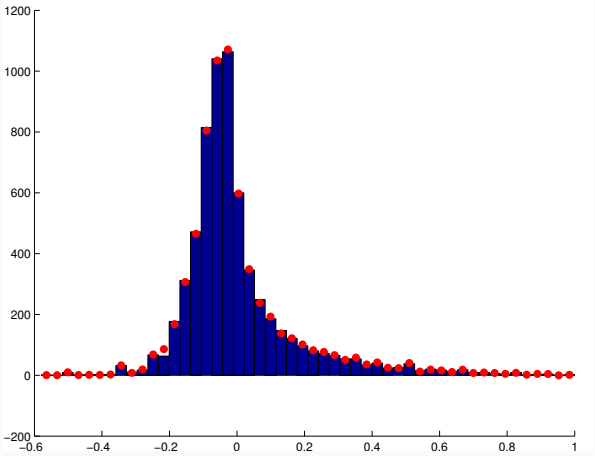
Reminder: Spectral Mapping

Consider a matrix H, and let f be analytic on the spectrum. Then if $H = V\Lambda V^{-1}$,

$$f(H) = Vf(\Lambda)V^{-1}.$$

(generalizes to non-diagonalizable case)

Another Perspective: Density of States



Spectra define a *generalized function* (a *density*):

$$\mathrm{tr}(f(H)) = \int f(\lambda)\mu(\lambda)\,dx = \sum_{j=k}^{N} f(\lambda_k)$$

where f is an analytic test function. Smooth to get a picture: a *spectral histogram* or *kernel density estimate*.

Example: Estrada Index

Consider

$$\mathrm{tr}(\mathsf{exp}(\alpha A)) = \sum_{k=1}^{\infty} \frac{\alpha^k}{k!} \cdot (\#\text{ closed random walks of length } k)\,.$$

- Global measure of connectivity in a graph.
- Can clearly be computed via DoS.
- Generalizes to other weights.

Heat Kernels

DoS information equivalent to looking at the *heat kernel trace*:

$$h(s) = \mathrm{tr}(\exp(-sH)) = \mathcal{L}[\mu](s)$$

Use $H = LD^{-1}$ (continuous time random walk generator) $\implies$
$h(s)/N$ = P(self-return after time s from uniform start).

Power Moments

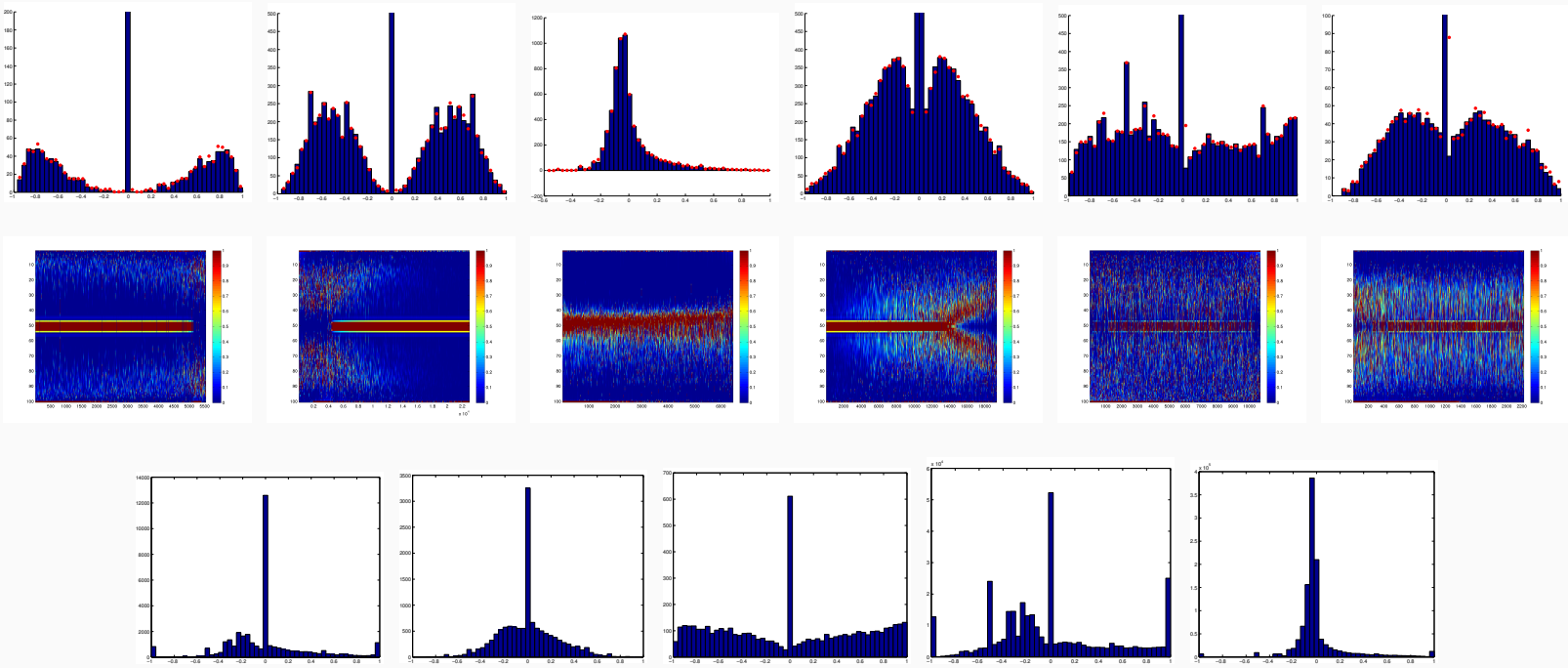
DoS information equivalent to looking at the *power moments*:

$$\mathrm{tr}(H^j).$$

Natural interpretation for matrices associated with graphs:

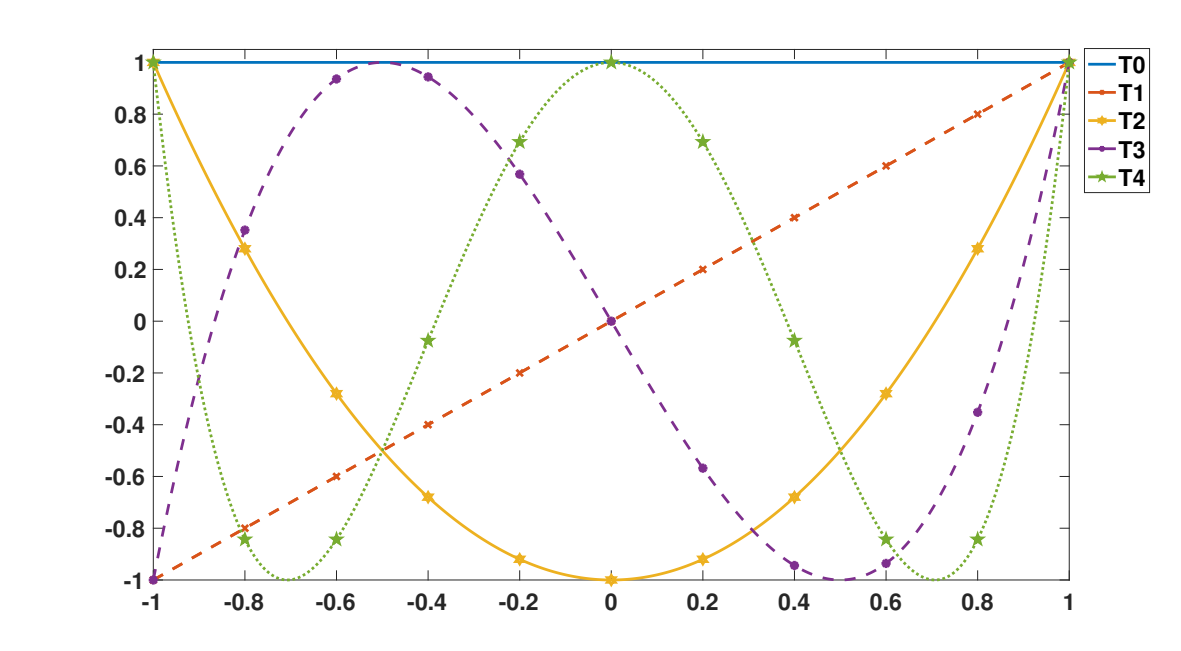
- A: number of length k cycles.
- $\bar{A}$ or P: return probability for k-step random walk (times N).
- L: ??

Today in Three Acts



- Act 1: Spectral densities
- **Act 2: Algorithms and approximations**
- Act 3: Spectral densities of “real world” graphs

Chebyshev Moments



For numerics, prefer Chebyshev moments to power moments:

$$d_j = T_j(A)$$

where $T_j(z) = \cos(j \cos^{-1}(z))$ is the jth Chebyshev polynomial:

$$T_0(z) = 1, \quad T_1(z) = z, \quad T_{k+1}z = 2zT_k(z) - T_{k-1}(z).$$

Exploring Spectral Densities

Kernel polynomial method (see Weisse, Rev. Modern Phys.)

- Spectral distribution on $[-1, 1]$ is a generalized function:

$$\int_{-1}^{1} \mu(x) f(x)\, dx = \frac{1}{N} \sum_{k=1}^{N} f(\lambda_k)$$

- Write $f(x) = \sum_{j=1}^{\infty} c_j T_j(x)$ and $\mu(x) = \sum_{j=1}^{\infty} d_j \phi_j(x)$, where $\int_{-1}^{1} \phi_j(x) T_k(x)\, dx = \delta_{jk}$
- Estimate $d_j = \mathrm{tr}(T_j(H))$ by stochastic methods
- Truncate series for $\mu(x)$ and filter (avoid Gibbs)

Much cheaper than computing all eigenvalues!

Alternatives: Lanczos (Golub-Meurant), maxent (Röder-Silver)

Stochastic Trace and Diagonal Estimation

$Z \in \mathbb{R}^n$ with independent entries, mean 0 and variance 1.

$$E[(Z \odot HZ)_i] = \sum_j h_{ij} E[Z_i Z_j] = h_{ii}$$

$$\mathrm{Var}[(Z \odot HZ)_i] = \sum_j h_{ij}^2.$$

Serves as the basis for stochastic estimation of

- Trace (Hutchinson, others; review by Toledo and Avron)
- Diagonal (Bekas, Kokiopoulou, and Saad)

Independent probes $\implies$ $1/\sqrt{N}$ convergence (usual MC).

Beyond Independent Probes

For probes $Z = [Z_1, \cdots, Z_s]$, have *exact* diagonal

$$d = \left[(A \odot ZZ^T)e\right] \oslash [(Z \odot Z)e]$$

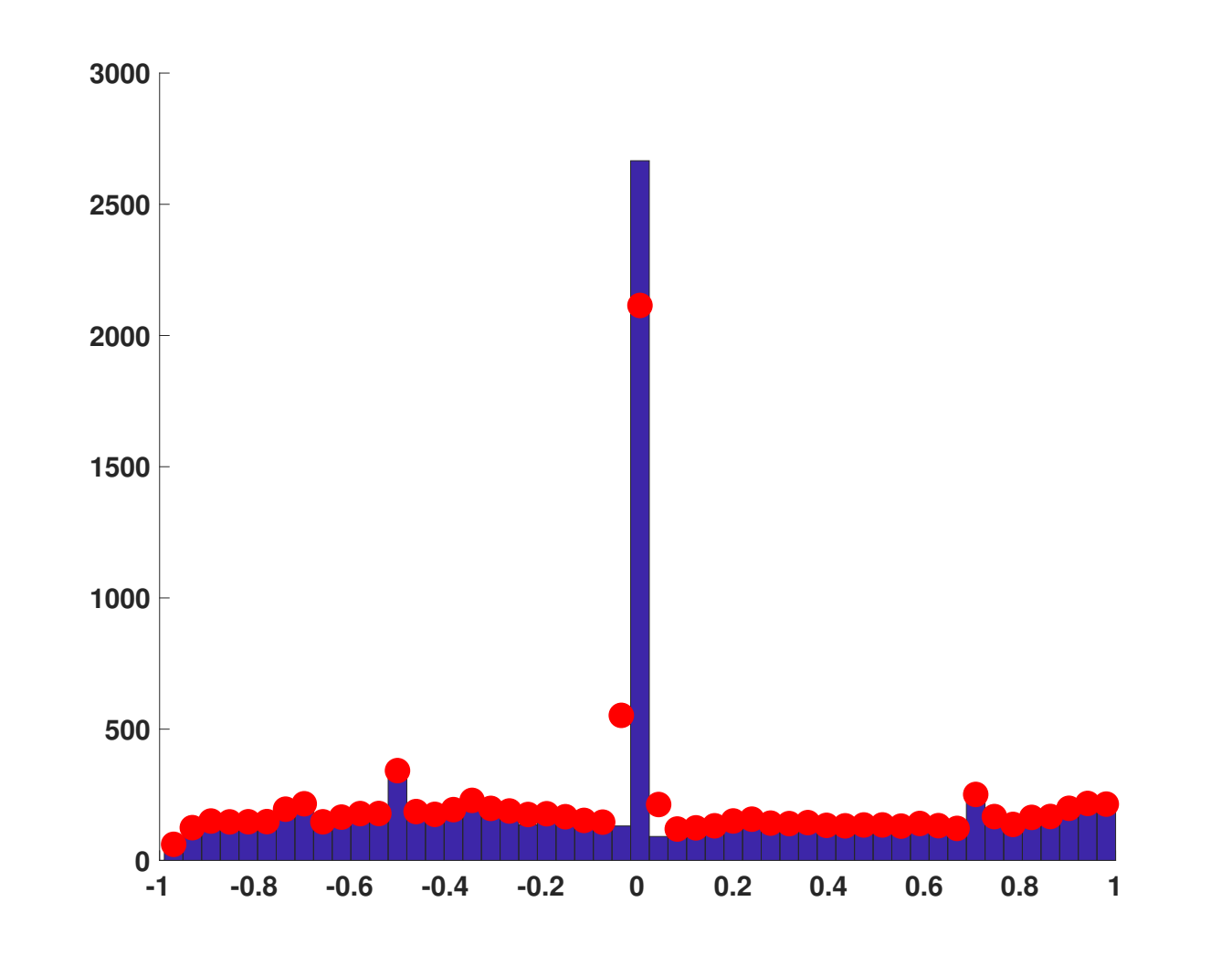
if $Z_{i,:}Zj,:^T = 0$ whenever $A_{ij} \neq 0$.

Idea:

- Pick rows $\{Z_{i,:}\}$ such that $Z_{i,:} \perp Z_{j,:}$ whenever $A_{ij} \neq 0$
- A an adjacency matrix $\implies$ graph coloring.

Combined with randomization, still gives unbiased estimates.

Example: PGP Network



Spike (non-smoothness) at eigenvalues of 0 leads to inaccurate approximation.

Motifs and Symmetry

Suppose $PH = HP$. Then

$$\mathcal{V} \text{ a max invariant subspace for } P \implies$$
$$\mathcal{V} \text{ a max invariant subspace for } H$$

So *local symmetry* $\implies$ *localized eigenvectors*.

Simplest example: P swaps (i, j)

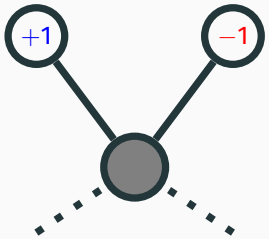
- $e_i - e_j$ an eigenvector of P with eigenvalue -1
- $e_i - e_j$ an eigenvector of $\bar{A}$ with eigenvalue

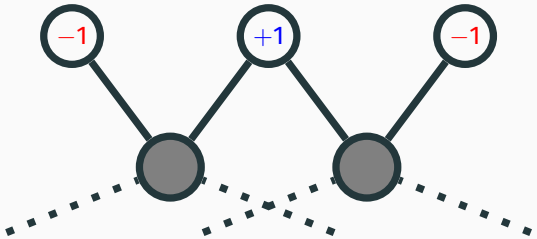
$$\lambda = \rho_{\bar{A}}(e_i - e_j) = \begin{cases} d^{-1}, & (i,j) \in \mathcal{E} \\ 0, & \text{otherwise.} \end{cases}$$

- All other eigenvectors (eigenvalue -1) satisfy $v_i = v_j$

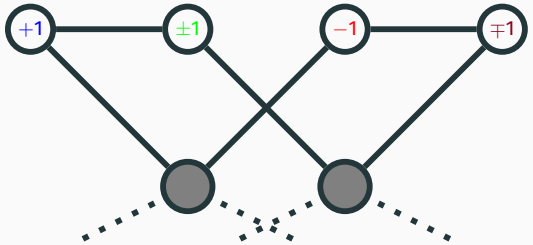
Motifs in Spectrum

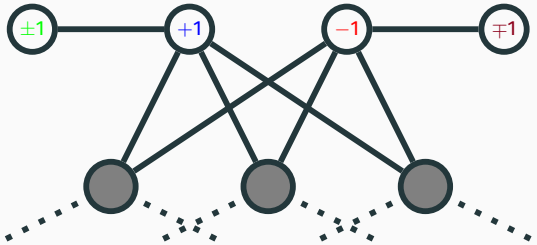
- $\lambda = 0$



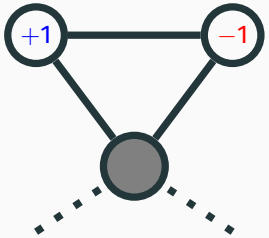


- $\lambda = \pm 1/2$

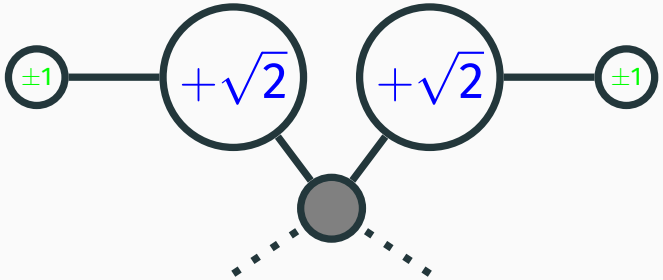




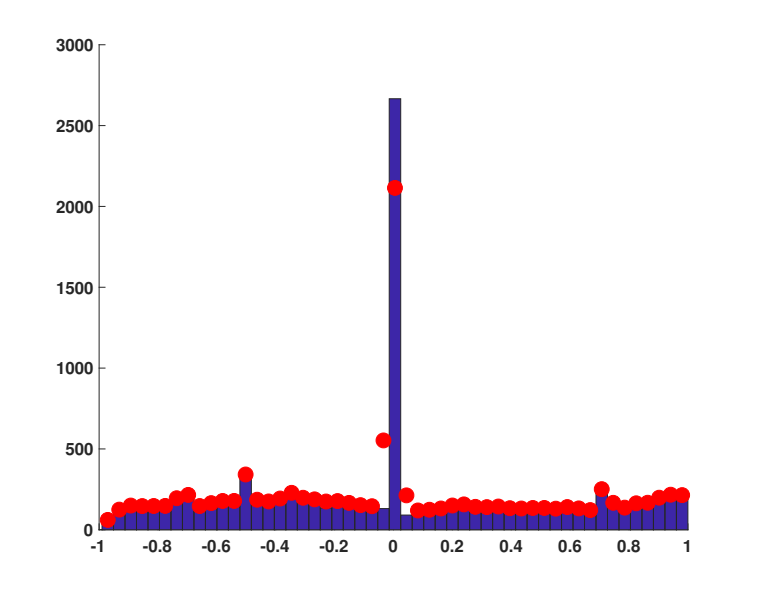
- $\lambda = -1/2$

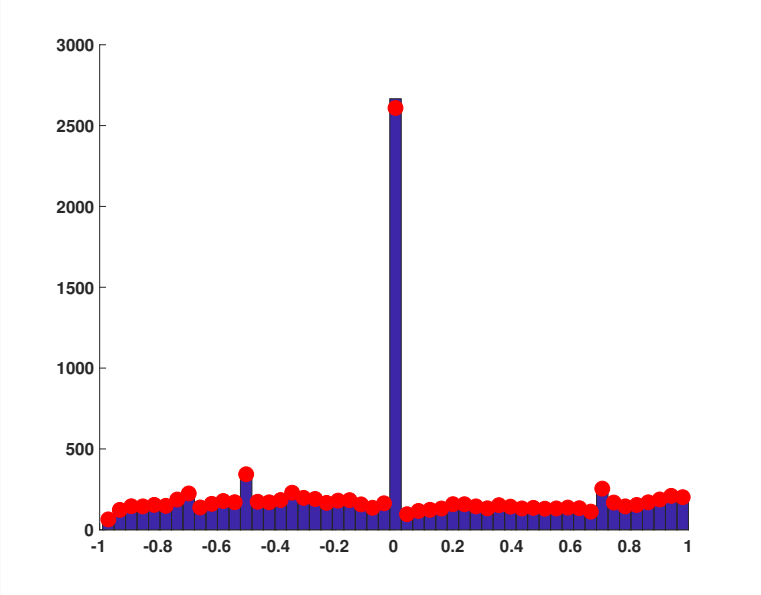


$\lambda = \pm 1/\sqrt{2}$



Motif Filtering





Motif "spikes" slow convergence – deflate motif eigenvectors!
If $P \in \mathbb{R}^{n \times m}$ an orthonormal basis for the quotient space,

- Apply estimator to $P^T \overline{A} P$ to reduce size for $m \ll n$.
- or use $Proj_P(Z)$ to probe the desired subspace.

Diagonal Estimation and LDoS

Diagonal estimation also useful for *local* DoS $\nu_k(x)$;
in the symmetric case with $H = Q\Lambda Q^T$, have

$$\int f(x)\nu_k(x)\,dx = f(H)_{kk} = e_k^T Q f(\Lambda) Q^T e_k$$

$$\nu_k(x) = \sum_{j=1}^{n} q_{kj}^2\,\delta(x - \lambda_j)$$

DoS is sum of local densities of states:

$$\mu(x) = \sum_{k=1}^{n} \nu_k(x)$$

KPM for LDoS

Same game, different moments:

- Estimate $d_j = [T_j(H)]_{kk}$ by diag estimation
- Truncate series for $\mu(x)$ and filter (avoid Gibbs)

Diagonal estimator gives moments *for all k simultaneously*!

Alternatives: Lanczos (Golub-Meurant), maxent (Röder-Silver)

LDoS Information

Can compute common *centrality measures* with LDoS

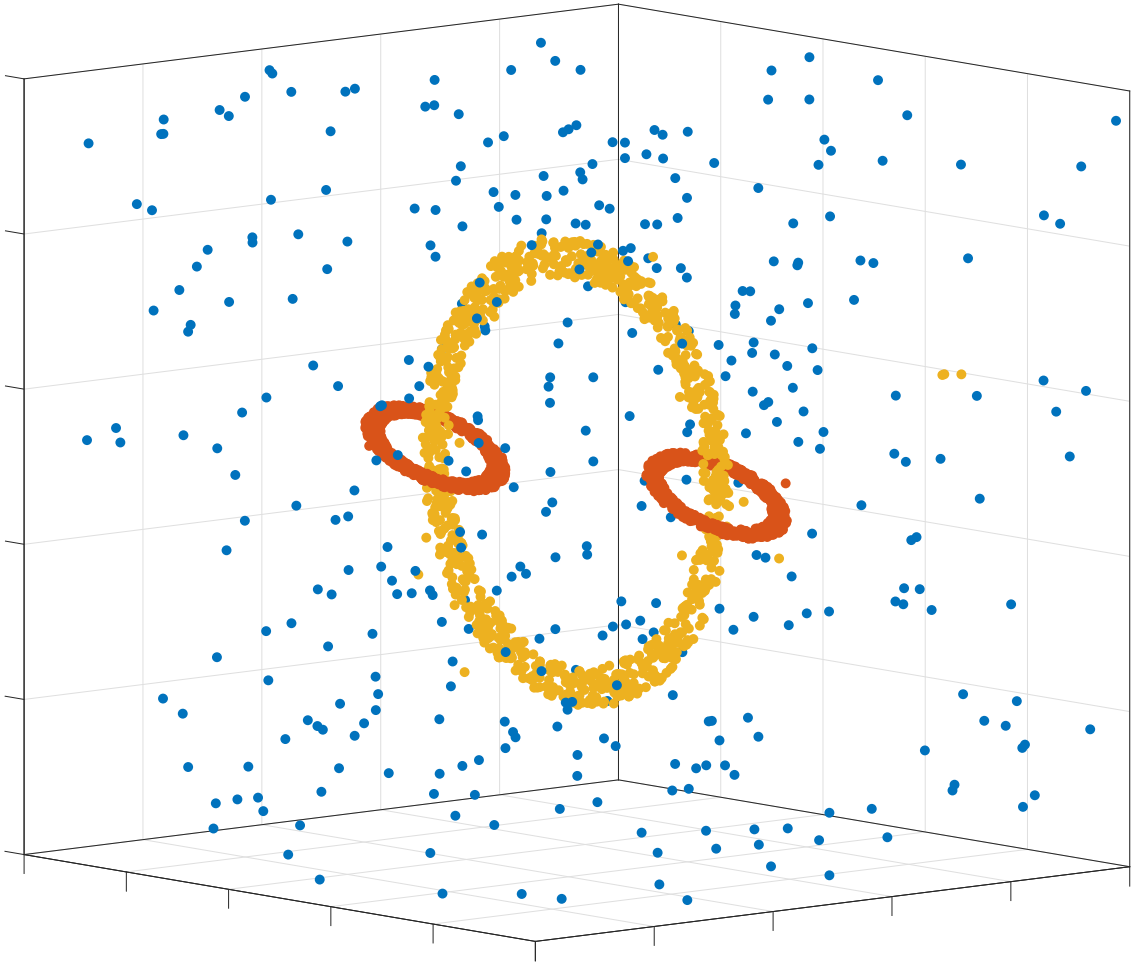
- Estrada centrality: $\exp(\gamma A)_{kk}$
- Resolvent centrality: $\left[(I - \gamma \bar{A})^{-1}\right]_{kk}$

Some motifs associated with localized eigenvectors:

- Chief example: Null vectors of $\bar{A}$ supported on leaves.
- Use LDoS + topology to find motifs?

What else?

LDoS and Clustering



Phase Retrieval in Graph Reconstruction

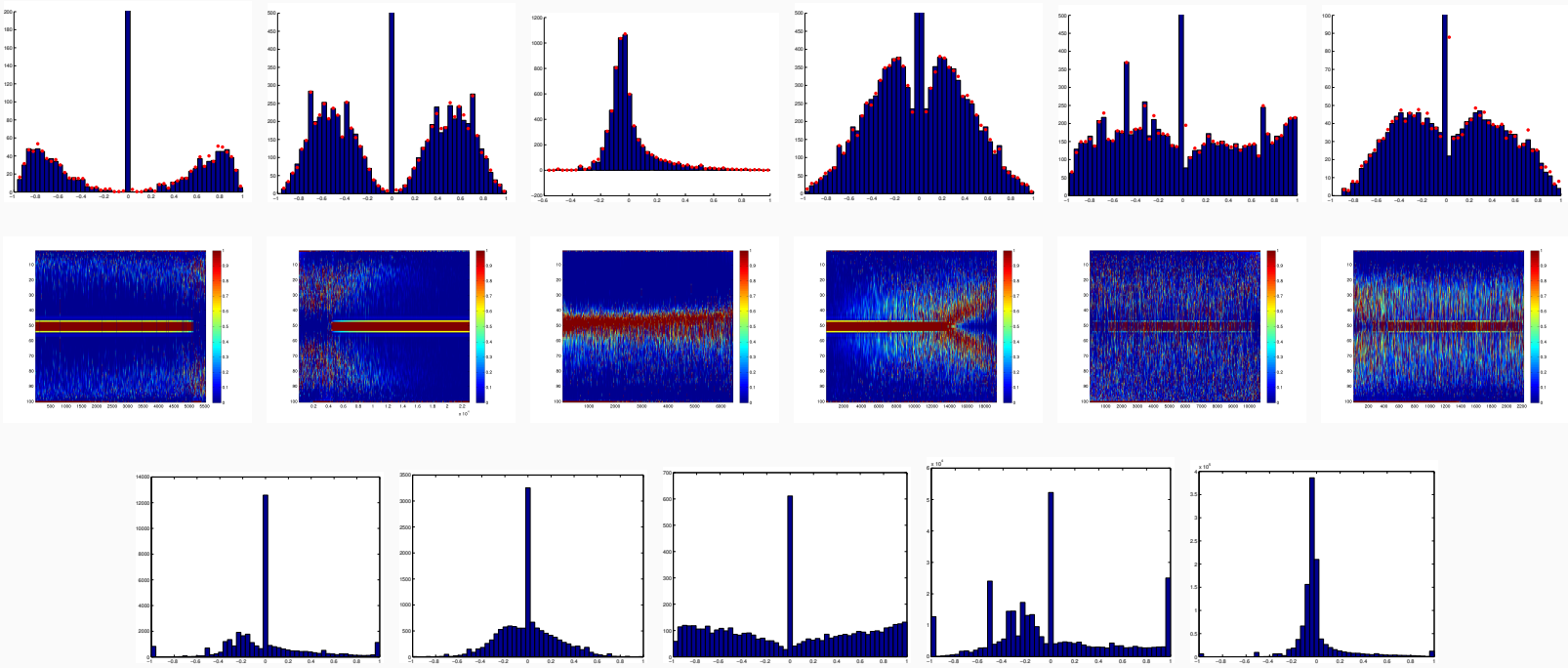
Reconstruct graph from *fully resolved* LDoS at all nodes?

- Assume $H = Q\Lambda Q^T$
- No multiple eigenvalues $\implies$ know $|Q|$ and Λ
- Can we recover signs in Q?

Feels a little like phase retrieval...

Of course, we usually have noisy LDoS estimates!

Today in Three Acts



- Act 1: Spectral densities
- Act 2: Algorithms and approximations
- **Act 3: Spectral densities of “real world” graphs**

Exploring Spectral Densities (with David Gleich)

- Compute spectrum of normalized Laplacian / RW matrix
- Compare KPM to full eigencomputation

Things we know

- Eigenvalues in $[-1, 1]$; nonsymmetric in general
- Stability: change d edges, have

$$\lambda_{j-d} \leq \hat{\lambda}_j \leq \lambda_{j+d}$$

- kth moment = P(return after k-step random walk)
- Eigenvalue cluster near 1 $\sim$ well-separated clusters
- Eigenvalue cluster near -1 $\sim$ bipartite structure
- Eigenvalue cluster near 0 $\sim$ leaf clusters

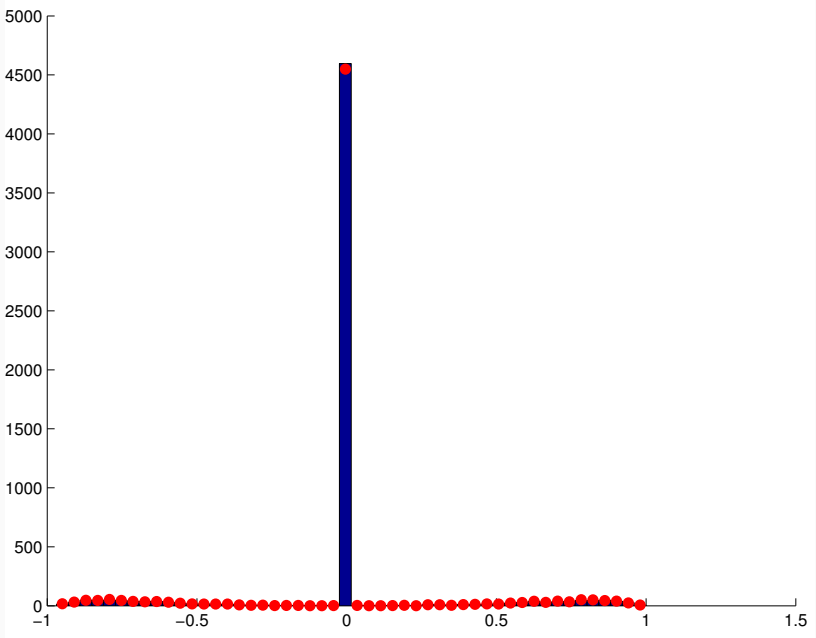
What else can we "hear"?

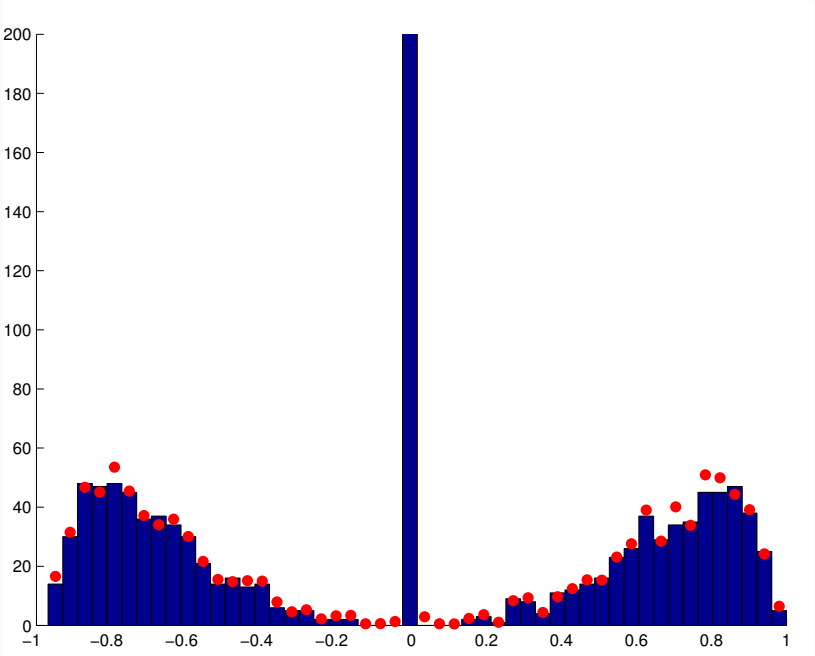
Experimental setup

- Global DoS
  - 1000 Chebyshev moments
  - 10 probe vectors (componentwise standard normal)
  - Histogram with 50 bins
- Local DoS
  - 100 Chebyshev moments
  - 10 probe vectors (componentwise standard normal)
  - Plot smoothed density on $[-1, 1]$
  - Spectrally order nodes by density plot

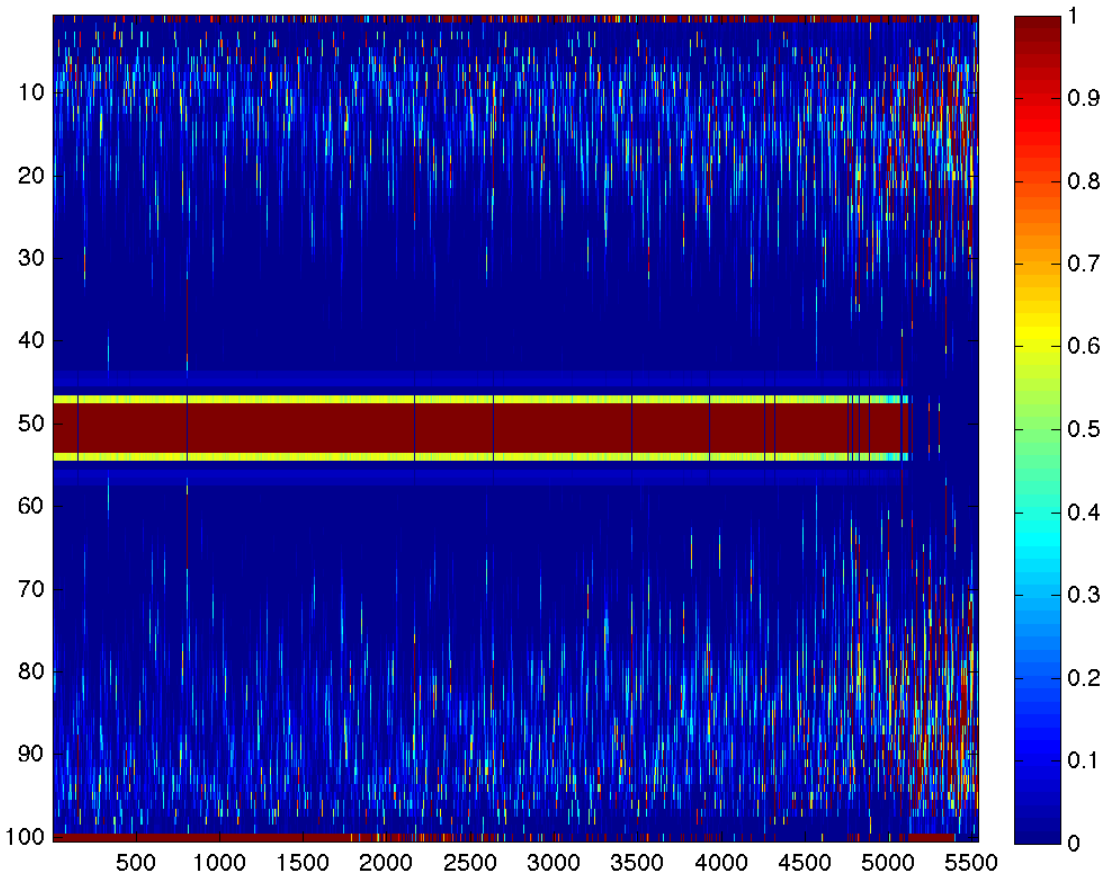
Suggestions for better pics are welcome!

Erdos

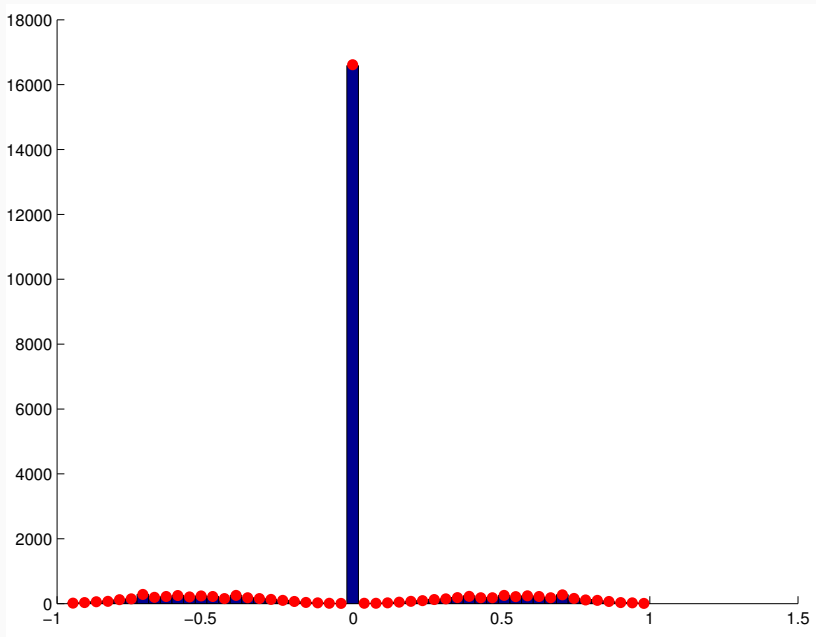

5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
−1
−0.5
0
0.5
1
1.5

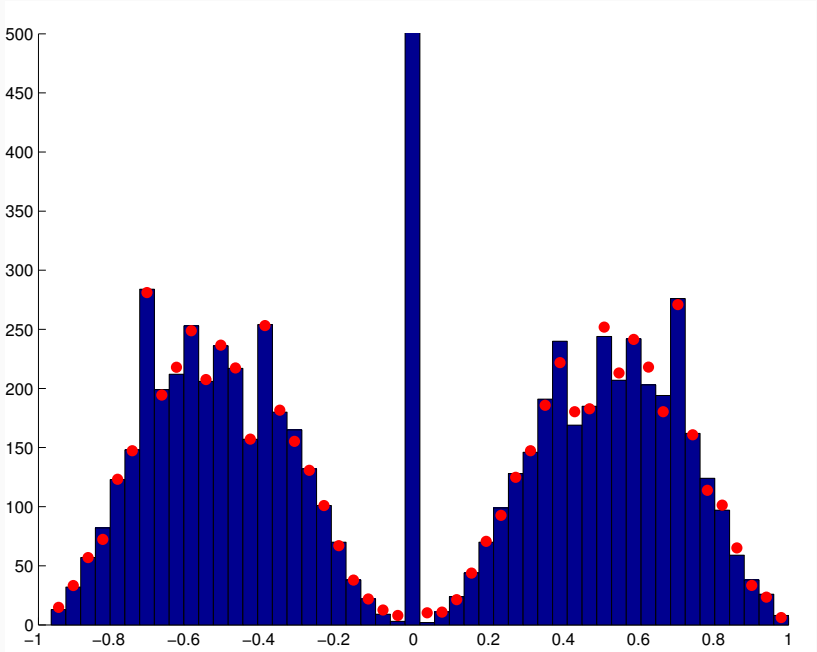

200
180
160
140
120
100
80
60
40
20
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

Erdos (local)

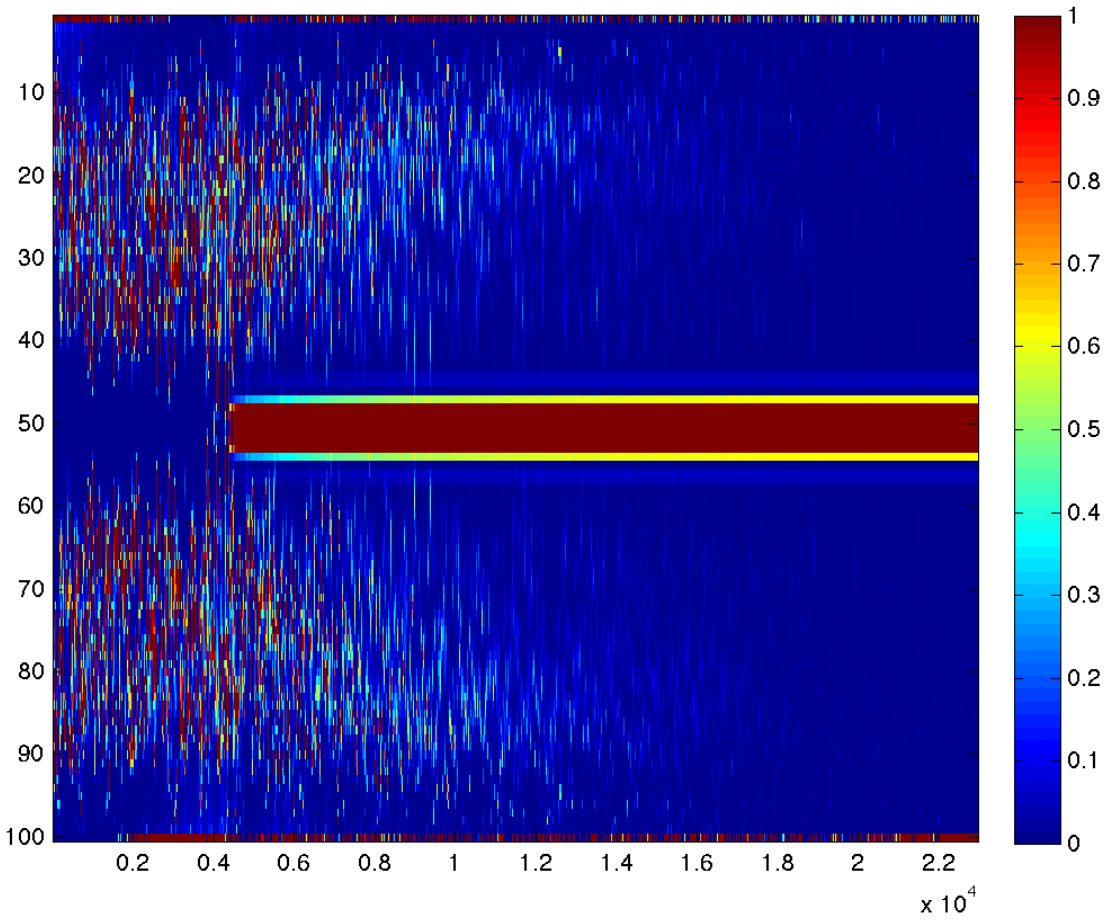

10
20
30
40
50
60
70
80
90
100
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
5500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Internet topology

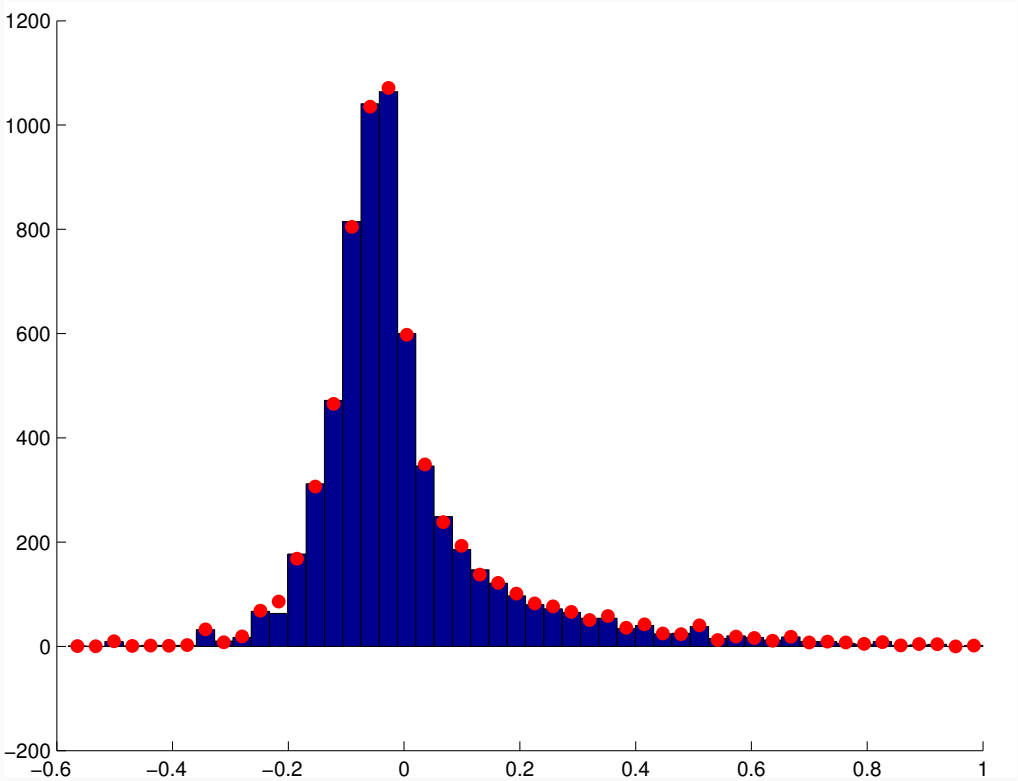




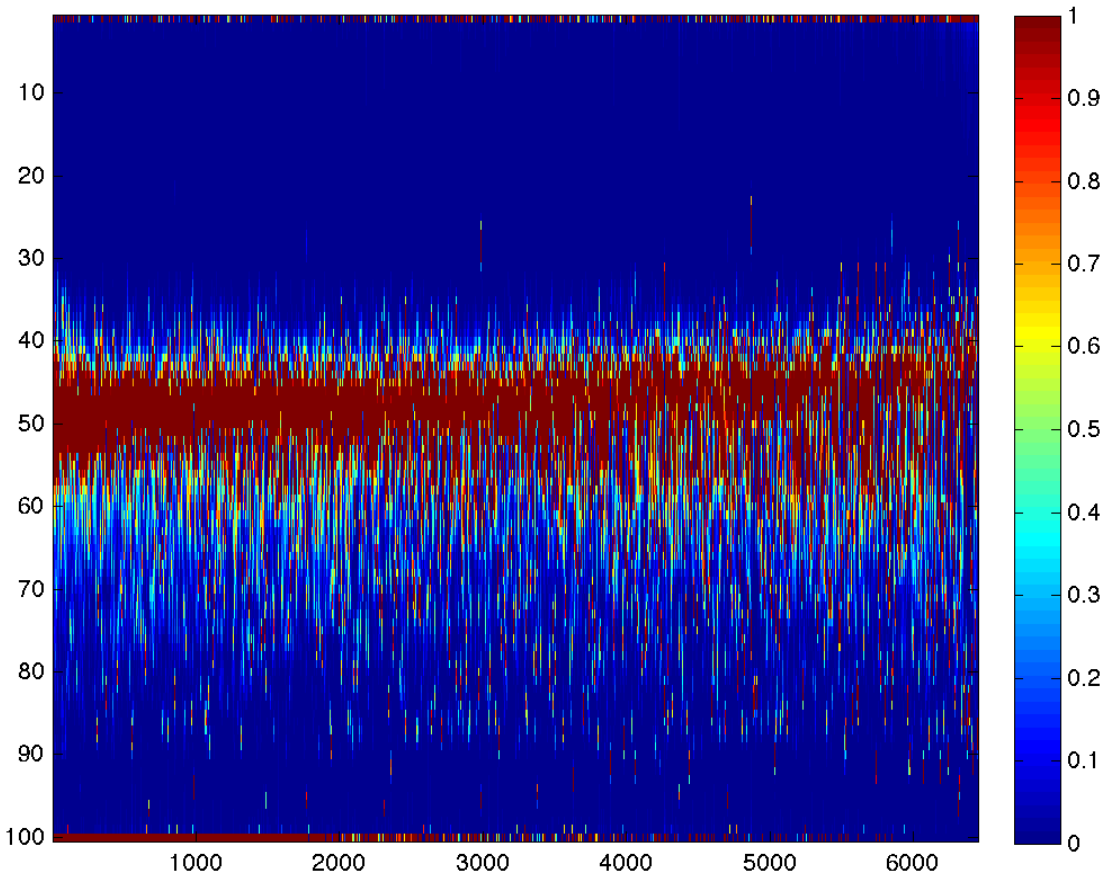
Internet topology (local)



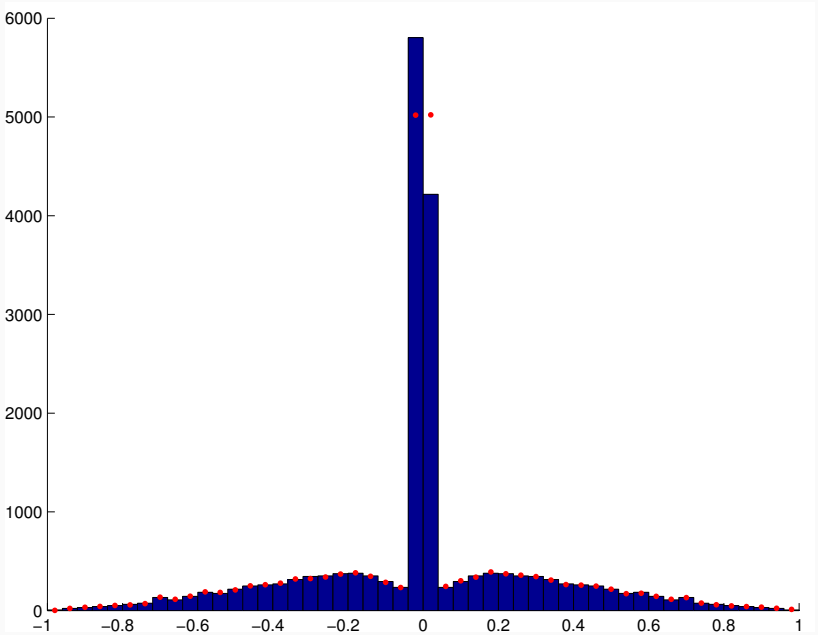
Marvel characters

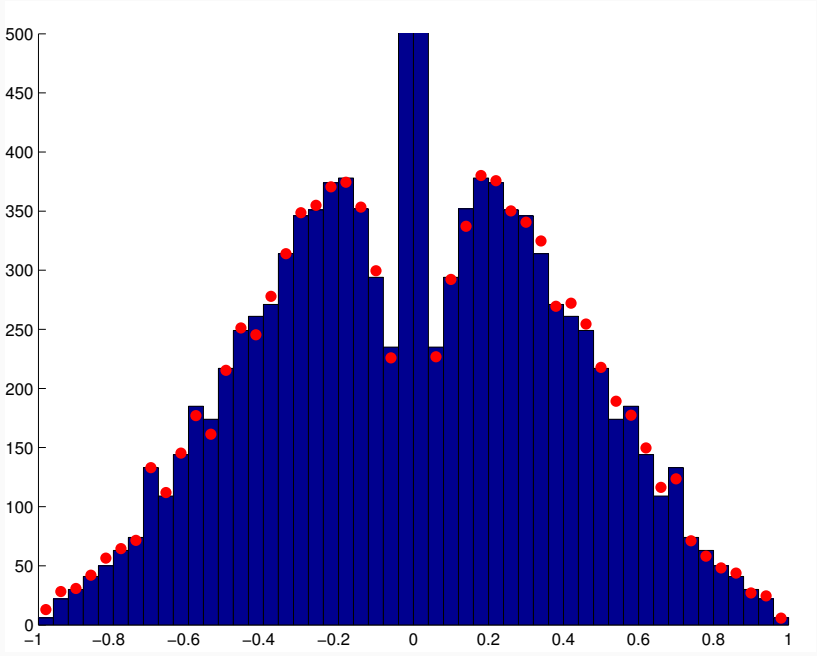


Marvel characters (local)

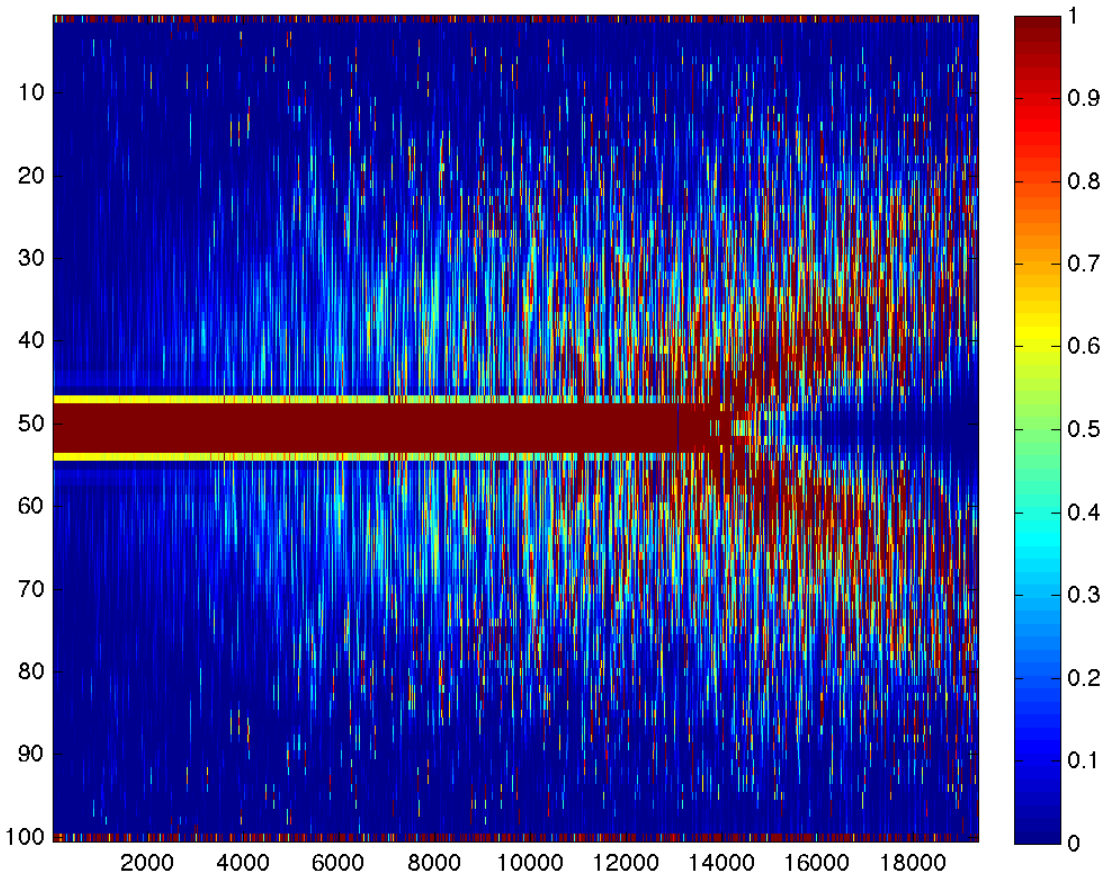


Marvel comics

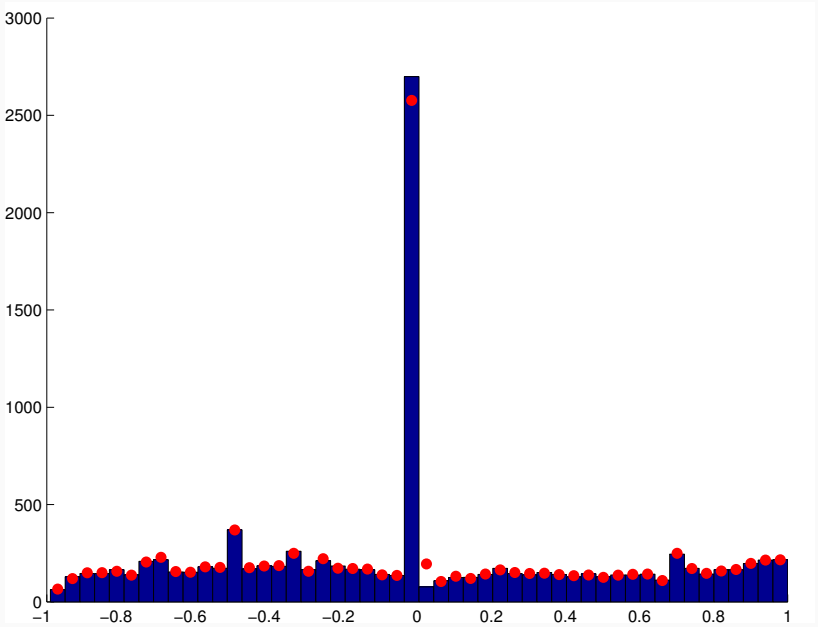


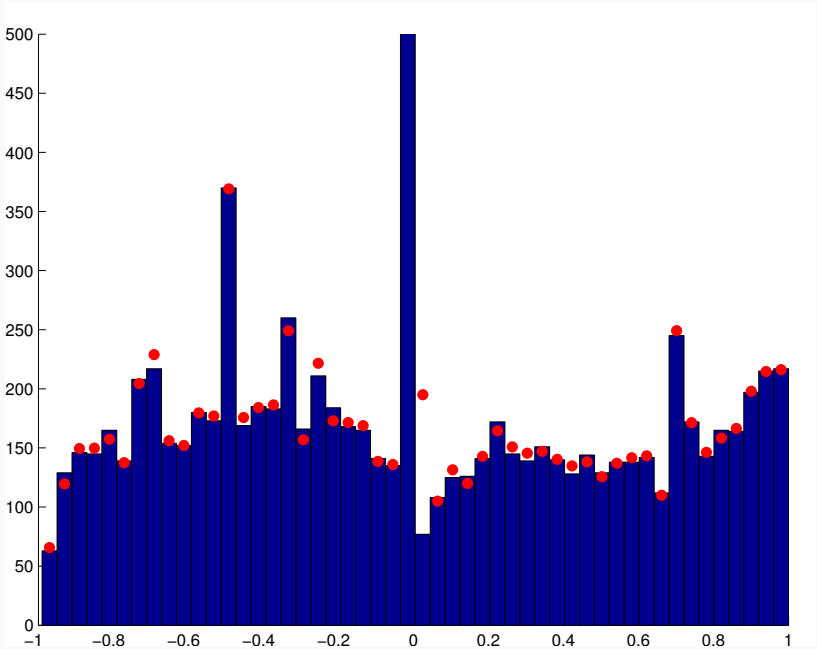


Marvel comics (local)

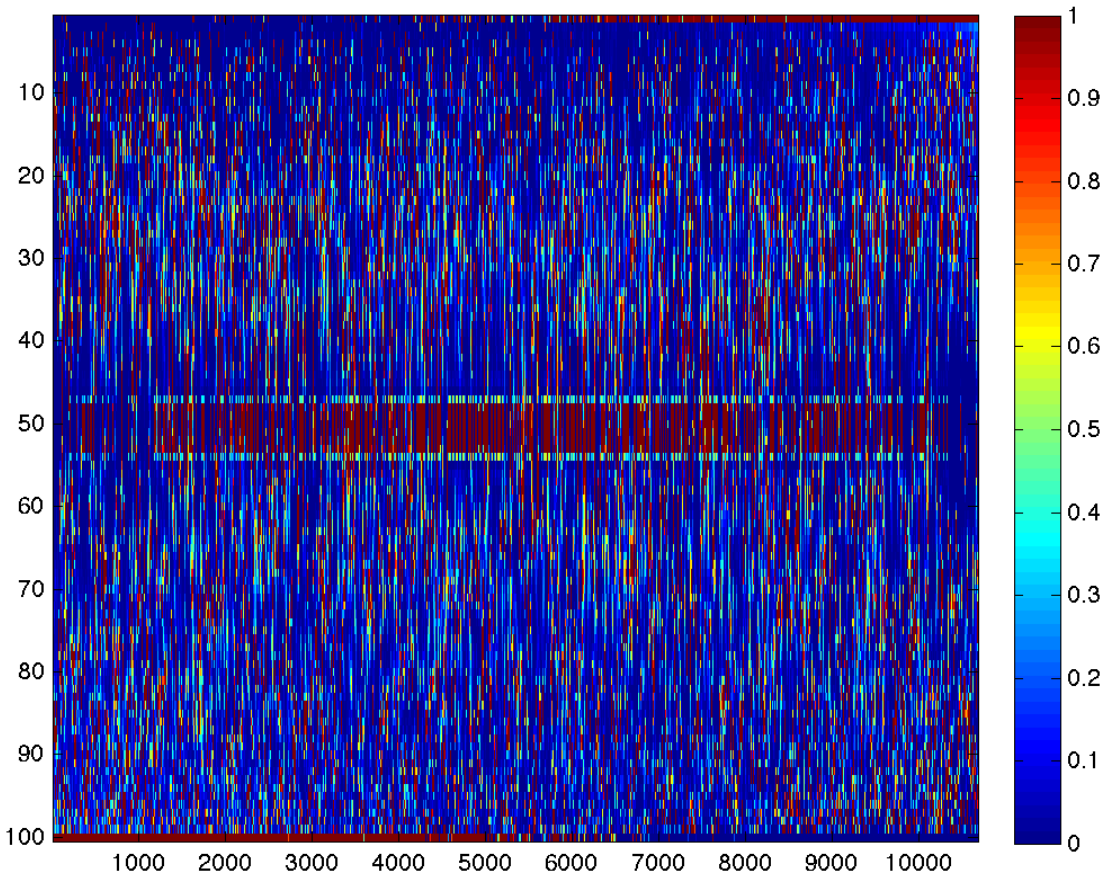


PGP

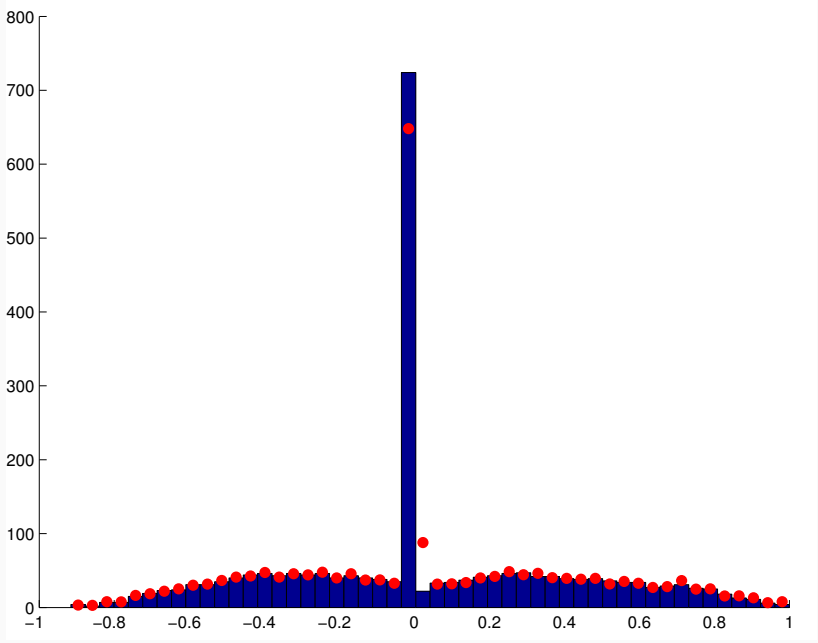

3000
2500
2000
1500
1000
500
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

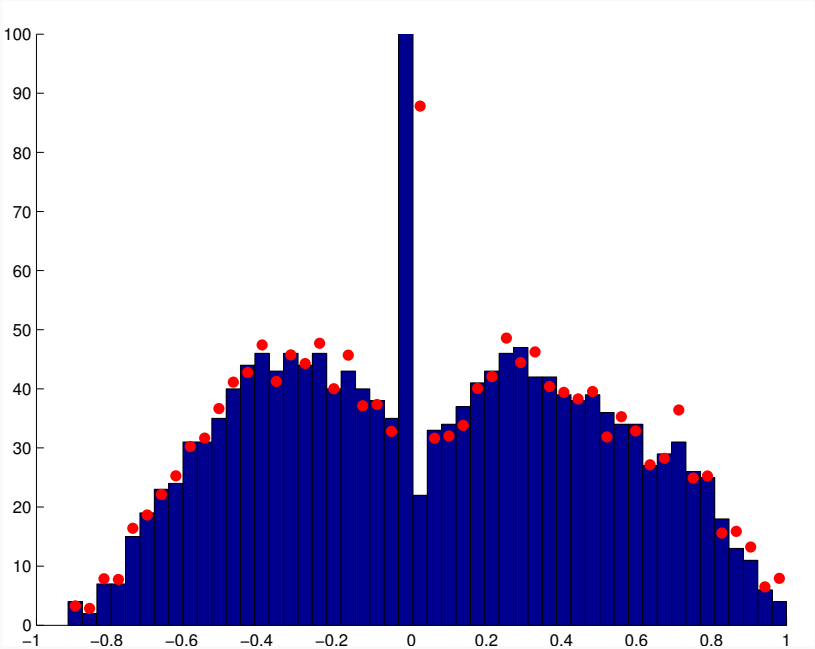

500
450
400
350
300
250
200
150
100
50
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

PGP (local)

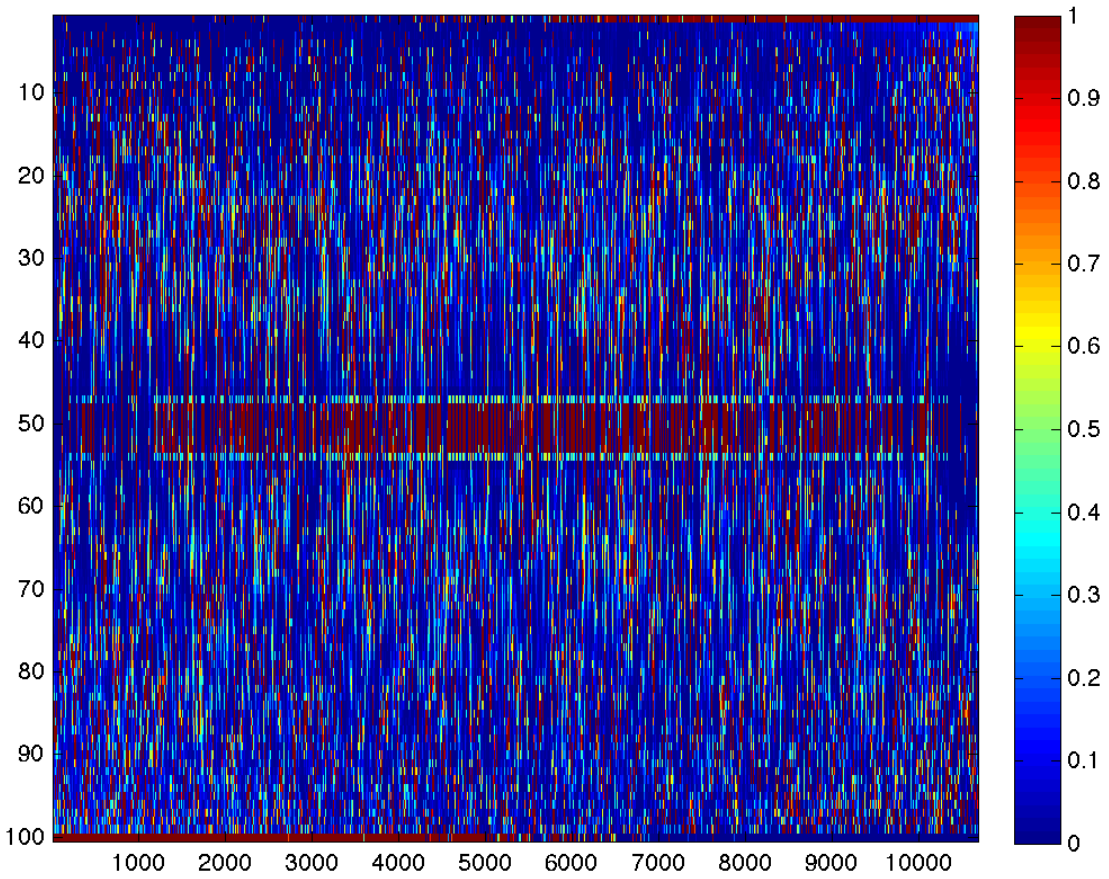

10
20
30
40
50
60
70
80
90
100
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Yeast


800
700
600
500
400
300
200
100
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

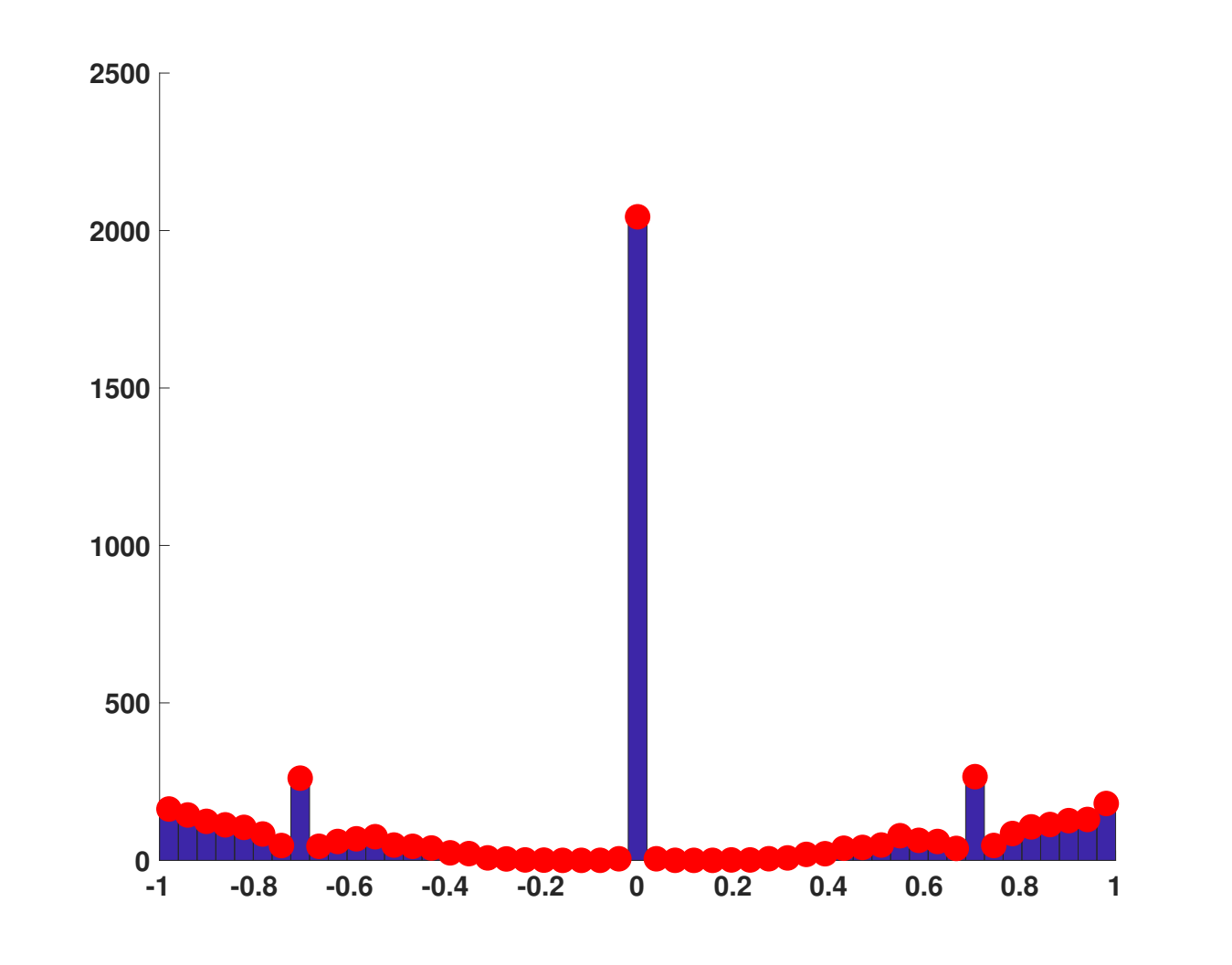

100
90
80
70
60
50
40
30
20
10
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

Yeast (local)


10
20
30
40
50
60
70
80
90
100
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

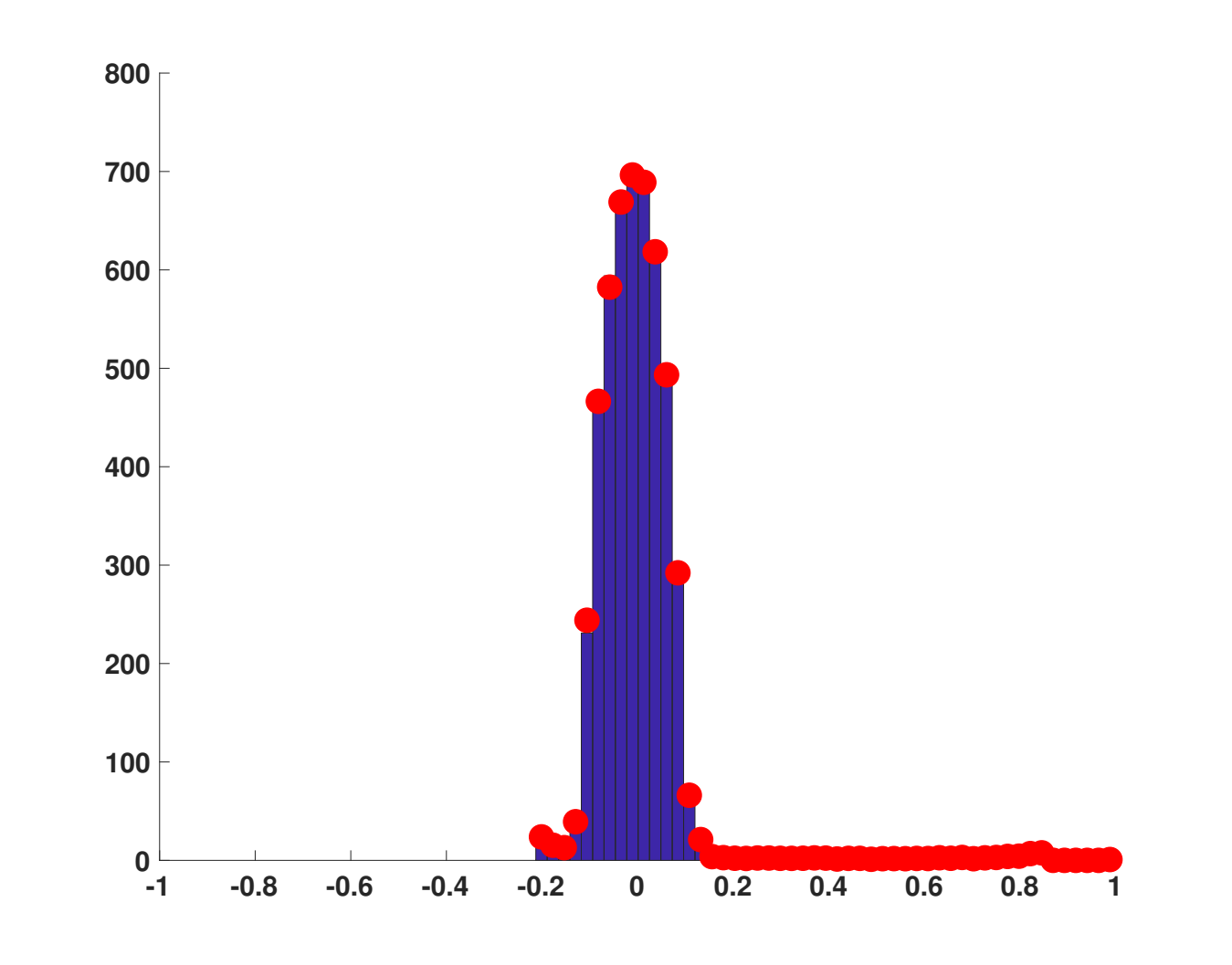
What about random graph models?

Barabási–Albert model



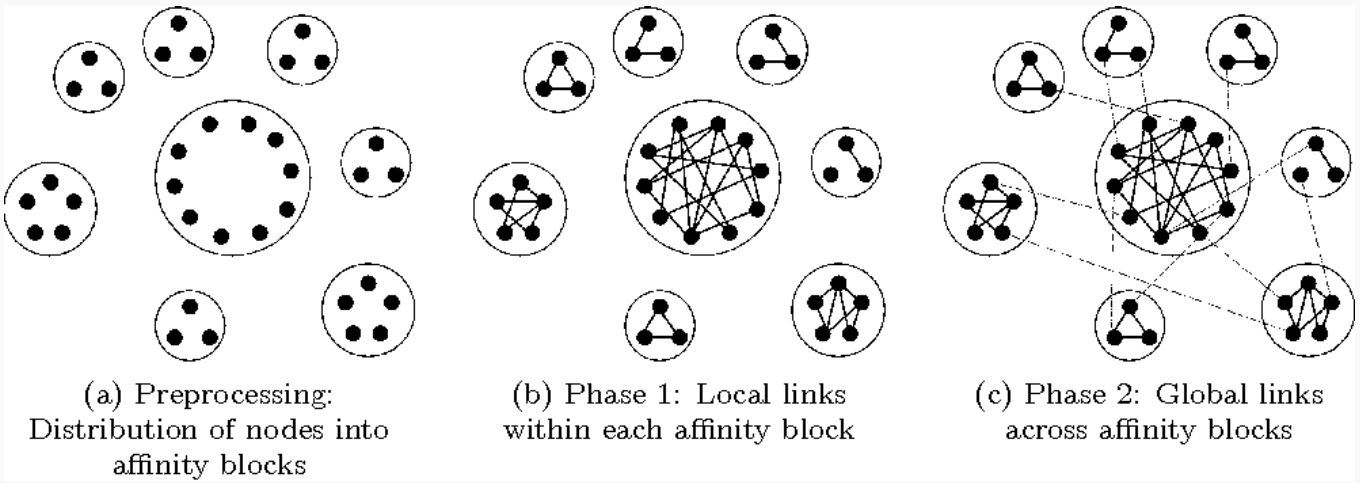
Scale-free network (5000 nodes, 4999 edges)

Watts–Strogatz



Small world network (5000 nodes, 260000 edges)

Model Verification: BTRE



Kolda et al, SISC (36), 2014

Block Two-Level Erdős-Rényi model (BTER)

- First Phase: Erdős-Rényi Blocks
- Second Phase: Using Chung-Lu Model to connect blocks with $p_{ij} = p(d_i, d_j)$

Model Verification: BTER

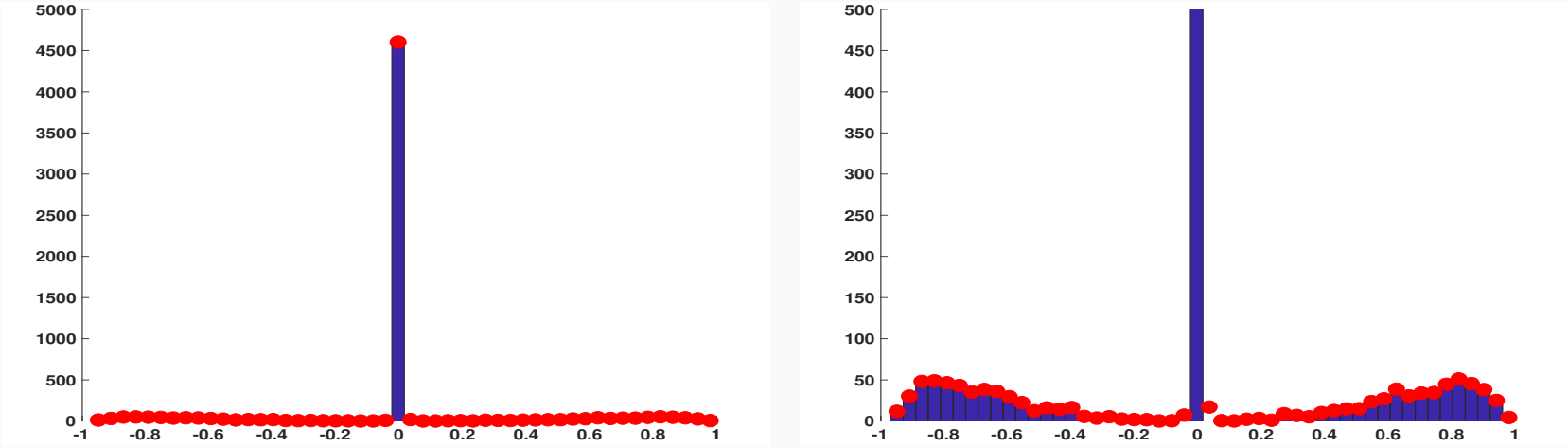


Figure 1: Erdos collaboration network.

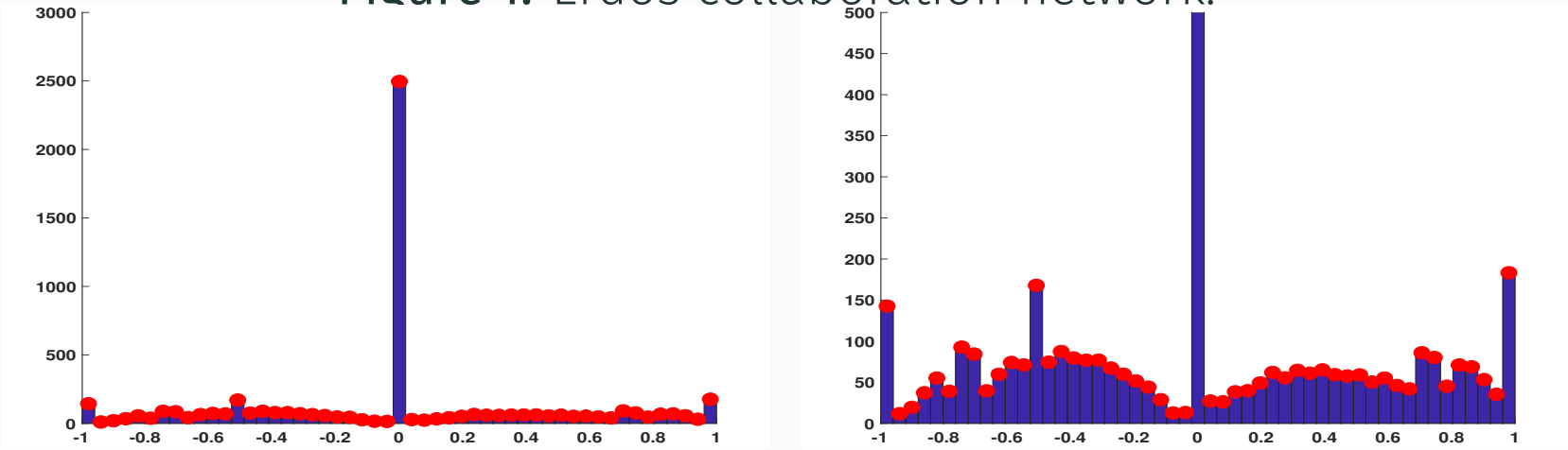
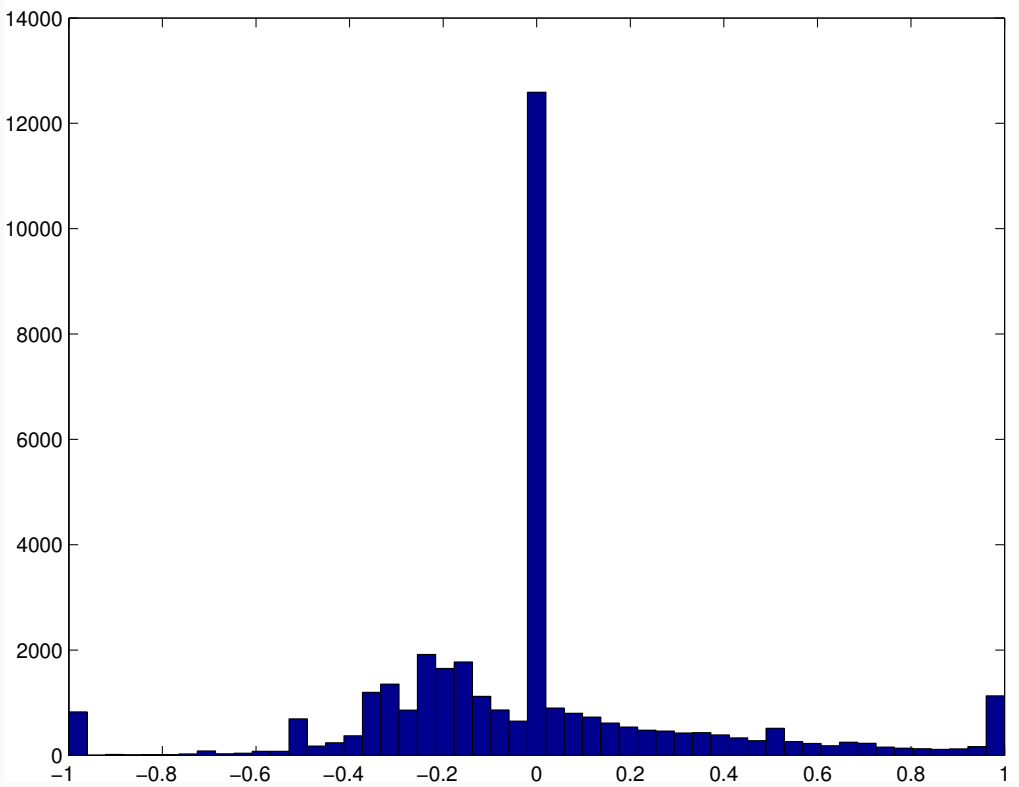


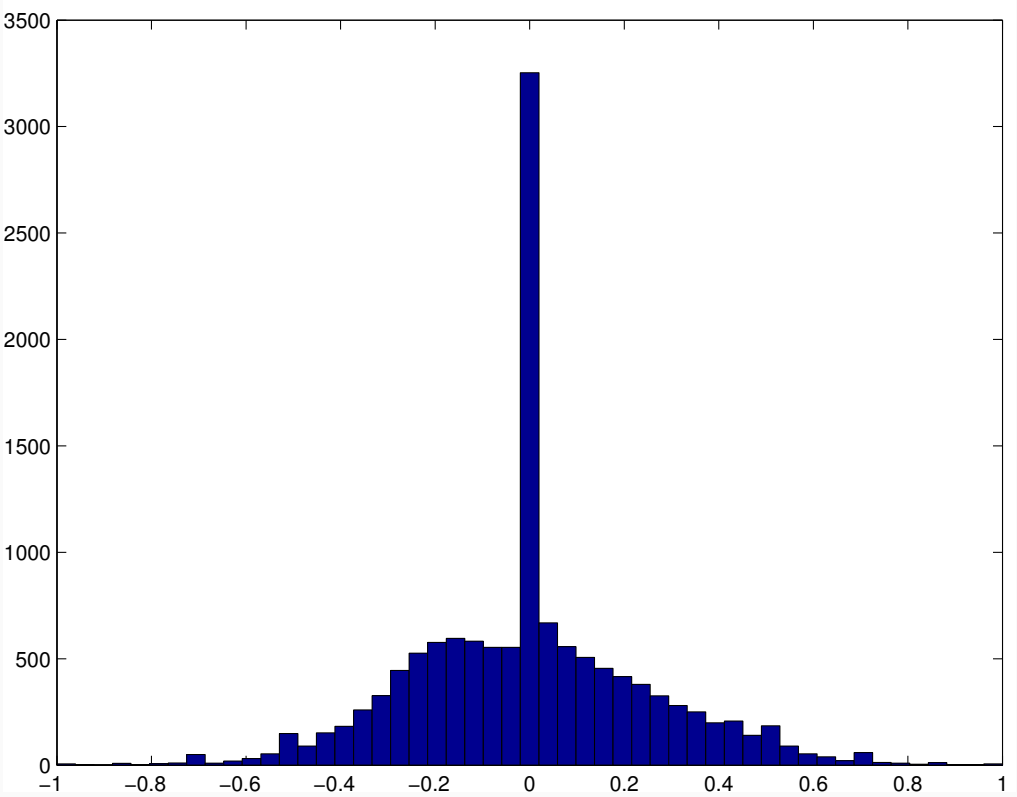
Figure 2: BTER model for Erdos collaboration network.

And a few more...

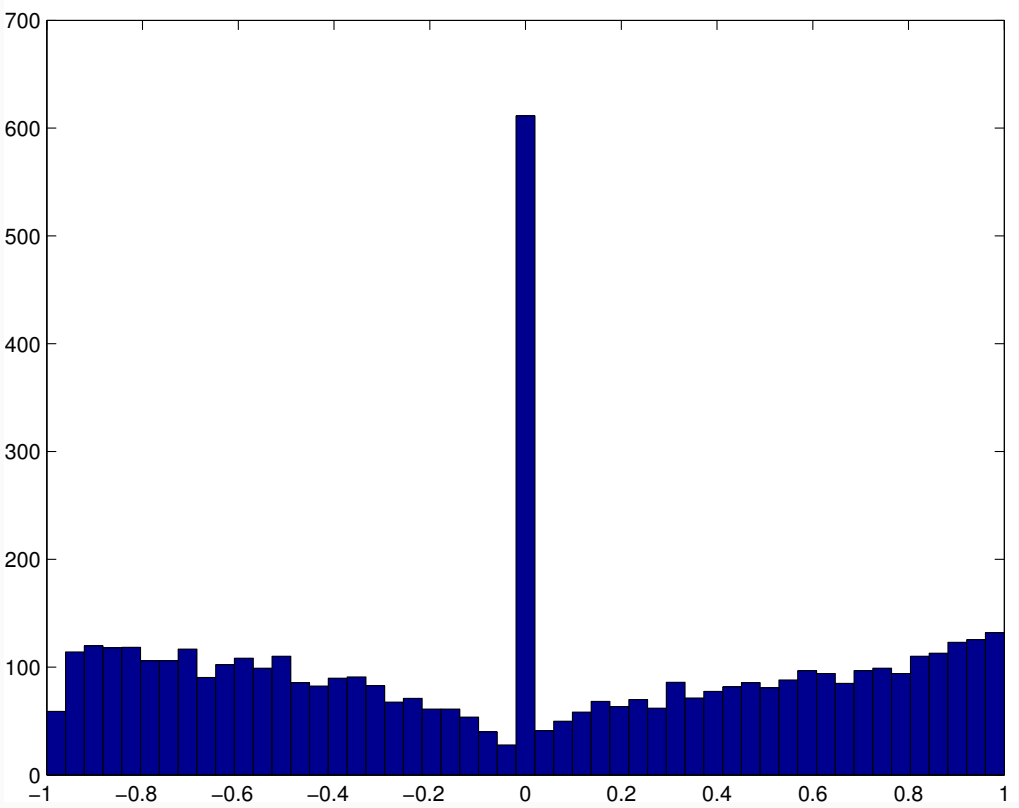
Enron emails (SNAP)



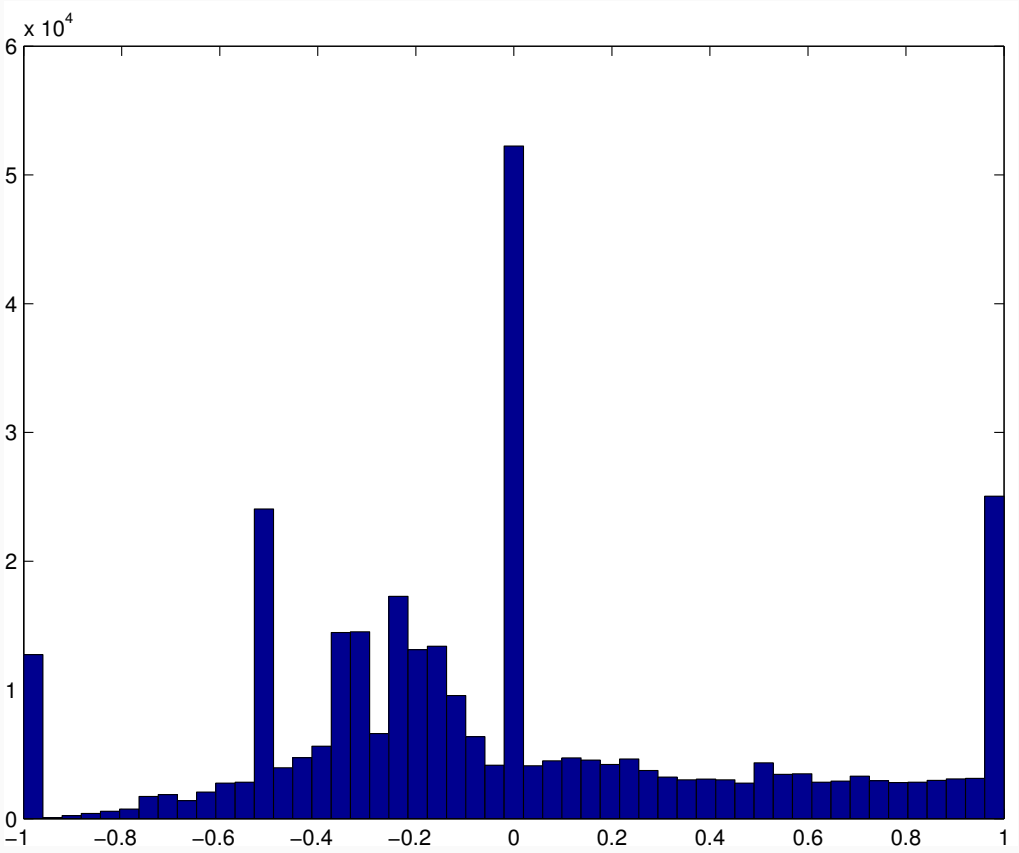
Reuters911 (Pajek)



US power grid (Pajek)

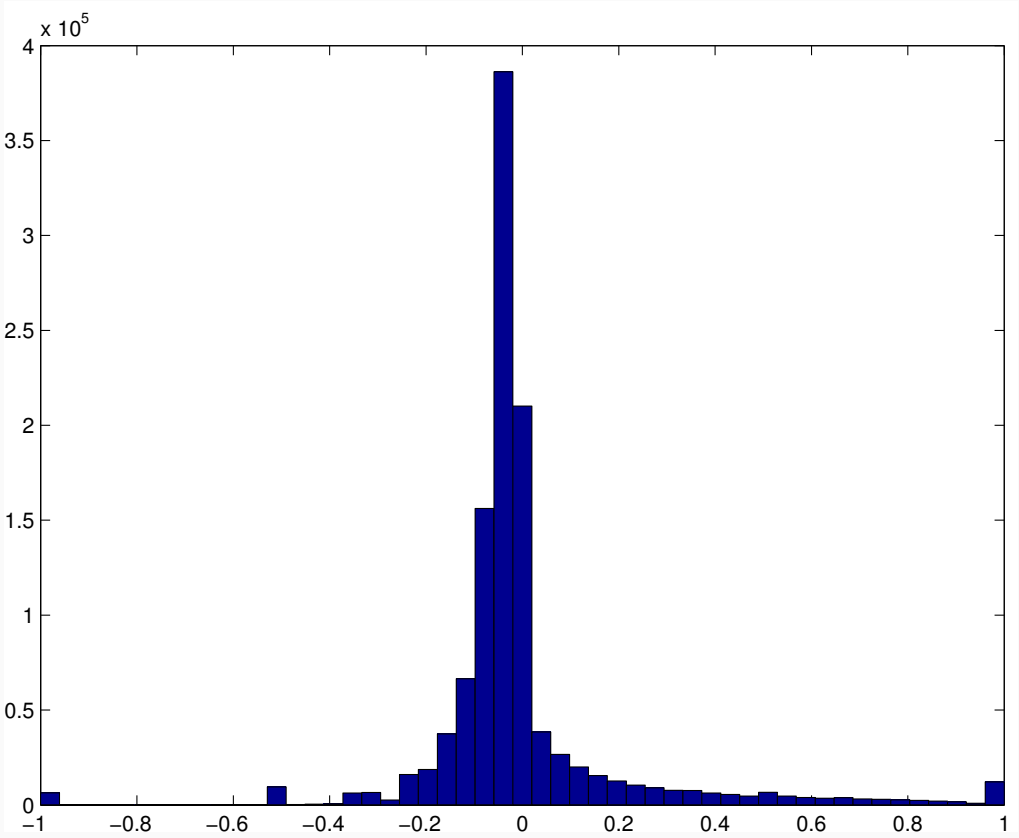


DBLP 2010 (LAW)



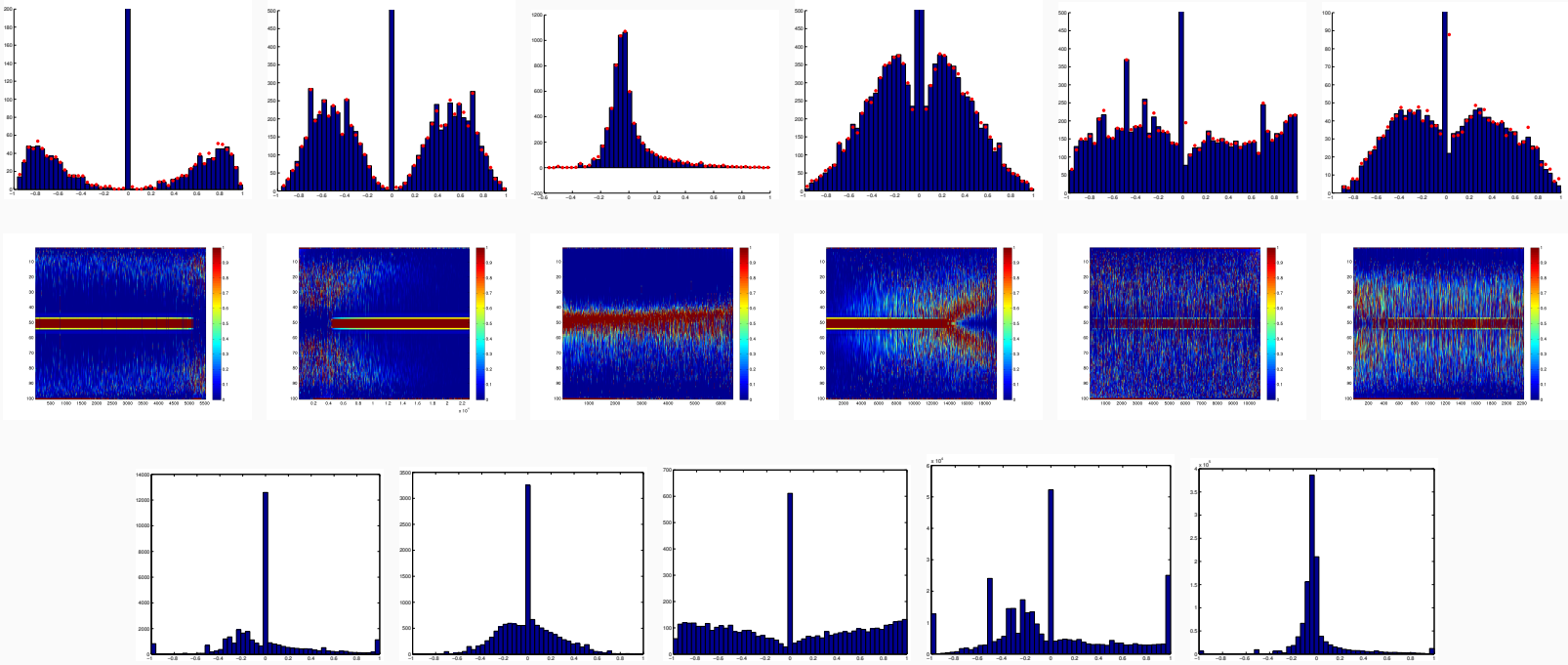
$N = 326186$, $nnz = 1615400$, 80 s (1000 moments, 10 probes)

Hollywood 2009 (LAW)



$N = 1139905$, $nnz = 113891327$, 2093 s (1000 moments, 10 probes)

Questions for You?

- Any isospectral graphs for multiple matrices?
- Can we recover topology from (exact) LDoS?
- Variance reduction in diagonal estimators?
- Random graphs with spectra that look “real”?
- Compression of moment information for diag estimators?
- More applications?

What Do You Hear?