UNIVERSITY OF MINNESOTA TWIN CITIES

Numerical Linear Algebra: from Scientific Computing to Data Science Applications

Yousef Saad

University of Minnesota

47th Annual Spring Lecture Series
University of Arkansas

May 4–6, 2022

This tutorial: Topics & Plan

- Current state of advanced Numerical Linear Algebra including:
  - First part: Sparse large matrix problems, linear systems, eigenvalue problems
  - Second: data-related problems: graphs, dimension reduction, ...
  - Prerequisite: senior level course in numerical linear algebra
  - 5 lectures + Matlab demos
  - All materials posted here:

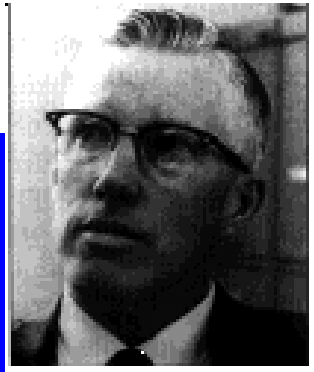
Schedule

Wed.	8:00– 9:00 am	Historical Perspective; Background & Examples; Sparsity; Data structures; Relaxation methods
Wed.	1:00 – 2:00 pm	Projection methods for lin. systems, Krylov methods Eigenvalue Pbs; Proj. Methods; Subs. it.; Lanczos
Thu.	8:00– 9:00 am	Backround on Graphs; Graph representations; Graphs for Data; Networks & Centrality; Graph Laplaceans.
Thu.	1:00 – 2:00 pm	Graph methods; Clustering; Segmentation; Graph embedding; Dimension Reduction; Informtion retrieval.
Fri.	8:00– 9:00 am	Supervised Learning; Neural Networks; Coarsening in scientific computing & in Data Sciences

Introduction: a historical perspective

In 1953, George Forsythe published a paper titled:

"Solving linear systems can be interesting".



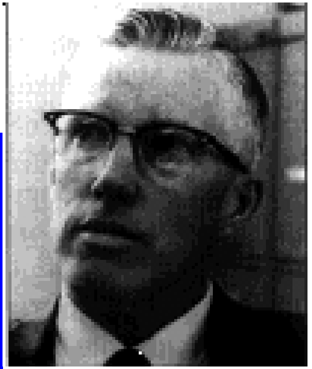
- Survey of the state of the art linear algebra in early 50s: direct methods, iterative methods, conditioning, preconditioning, The Conjugate Gradient, acceleration methods, ...

➤ An amazing paper in which the author was urging researchers to start looking at solving linear systems

Introduction: a historical perspective

In 1953, George Forsythe published a paper titled:

"Solving linear systems can be interesting".



- Survey of the state of the art linear algebra in early 50s: direct methods, iterative methods, conditioning, preconditioning, The Conjugate Gradient, acceleration methods, ...

➤ An amazing paper in which the author was urging researchers to start looking at solving linear systems

➤ Nearly 70 years later – we can certainly state that:

"Linear Algebra problems in Machine Learning can be interesting"

Focus of numerical linear algebra changes over time

- Linear algebra took many direction changes in the past

1940s–1950s: Major issue: flutter problem in aerospace engineering → eigenvalue problem [cf. Olga Taussky Todd] → LR, QR, .. → 'EISPACK'

1960s: Problems related to the power grid promoted what we would call today general sparse matrix techniques

1970s– Automotive, Aerospace, ..: Computational Fluid Dynamics (CFD)

Late 1980s: Thrust on parallel matrix computations.

Late 1990s: Spur of interest in "financial computing"

Current: Machine Learning

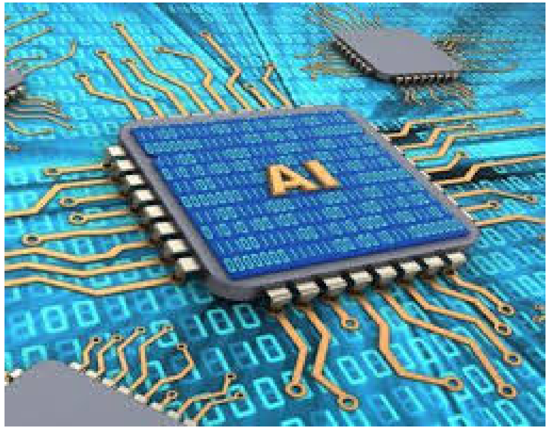
Solution of PDEs (e.g., Fluid Dynamics) and problems in mechanical eng. (e.g. structures) major force behind numerical linear algebra algorithms in the past few decades.

➤ Strong new forces are now reshaping the field today: Applications related to the use of “data”

➤ Machine learning is appearing in unexpected places:

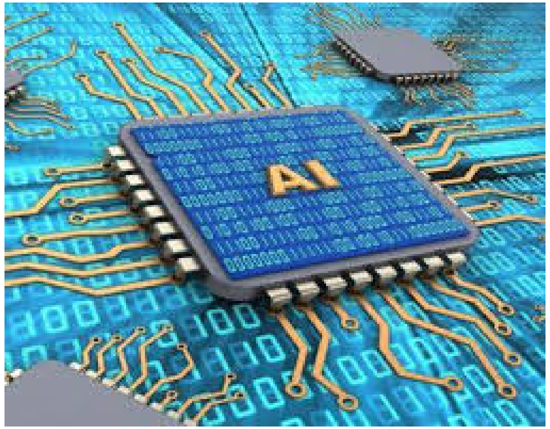
- design of materials
- machine learning in geophysics
- self-driving cars, ..
- ....

Big impact on the economy



- New economy driven by Google, Facebook, Netflix, Amazon, Twitter, Ali-Baba, Tencent, ..., and even the big department stores (Walmart, ...)
- Huge impact on **Jobs**

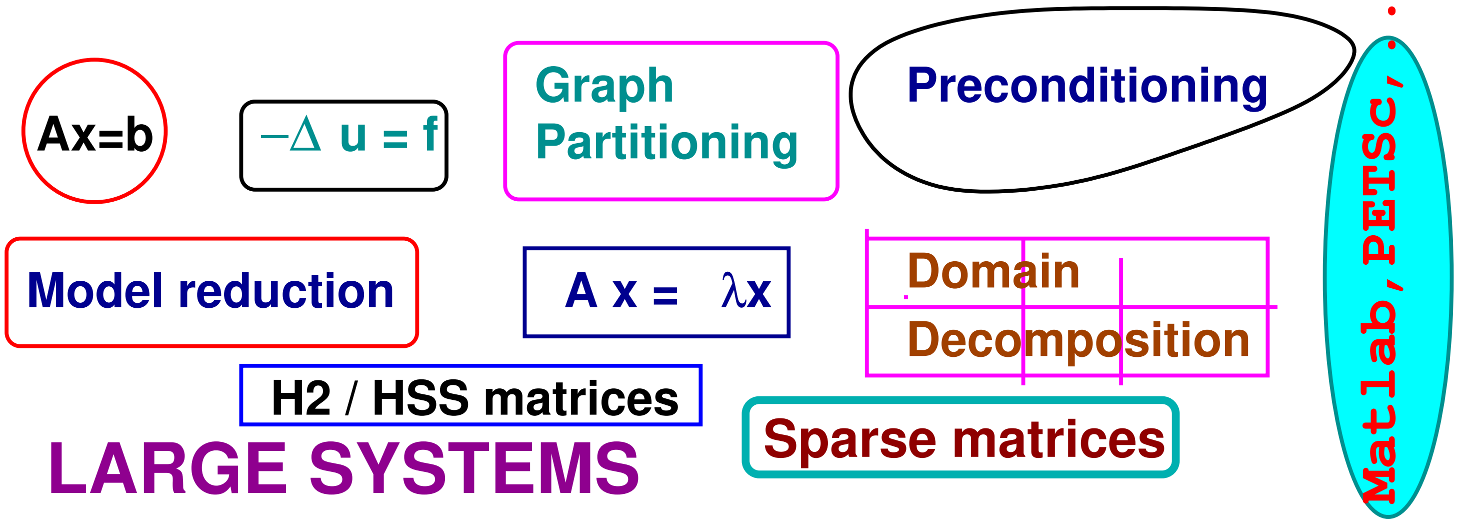
Big impact on the economy



- ➤ New economy driven by Google, Facebook, Netflix, Amazon, Twitter, Ali-Baba, Tencent, ..., and even the big department stores (Walmart, ...)
- ➤ Huge impact on **Jobs**

- ➤ Old leaders - e.g., Mining; Car companies; Aerospace; Manufacturing; offer little growth – Some instances of renewal driven by new technologies [e.g. Tesla]

- ➤ Look at what you are doing under new lenses: DATA



$Ax=b$

$-\Delta u = f$

Graph Partitioning

Preconditioning

Matlab, PETSc, ..

Model reduction

$A x = \lambda x$

Domain Decomposition

H2 / HSS matrices

Sparse matrices

LARGE SYSTEMS

$Ax=b$

$-\Delta u = f$

Graph Partitioning

Preconditioning

Matlab, PETSC, ...

Model reduction

$A x = \lambda x$

Domain Decomposition

H2 / HSS matrices

Sparse matrices

LARGE SYSTEMS

Translate

$A = U \Sigma V^T$

PCA

Clustering

Dimension Reduction

Python, PyTorch

Semi–Supervised Learning

Graph Laplaceans

Divide & Conquer

Data Sparsity

Regression

LASSO

BIG DATA!

Impact on what we teach...

- My course: *CSCI 8314: Sparse Matrix Computations*
  [url: my website - follow teaching]

... Has changed substantially in past 4-6 years

Before: —*PDEs, solving linear systems, Sparse direct solvers, Iterative methods, Krylov methods, Preconditioners, Multigrid,..*

$\longrightarrow$

Now: — a little of sparse direct methods + Applications of graphs, dimension reduction, Krylov methods.. Examples in: PCA, Information retrieval, Segmentation, Clustering, ...

General Introduction and Background

➤ This tutorial is about Numerical Linear Algebra – both the *classical* kind and the *new*:

- Standard matrix computations (e.g. solving linear systems, eigenvalue/SVD problems, ...)
- Graph algorithms and tools (Sparse graphs, graph coarsening, graphs and sparse methods). ..
- Dimension reduction methods; Graph embeddings;
- Specific machine learning algorithms; unsupervised/ supervised learning;
- Graph coarsening methods in scientific computing and machine learning

Example: Fluid flow

Physical Model

↓

Nonlinear PDEs

↓

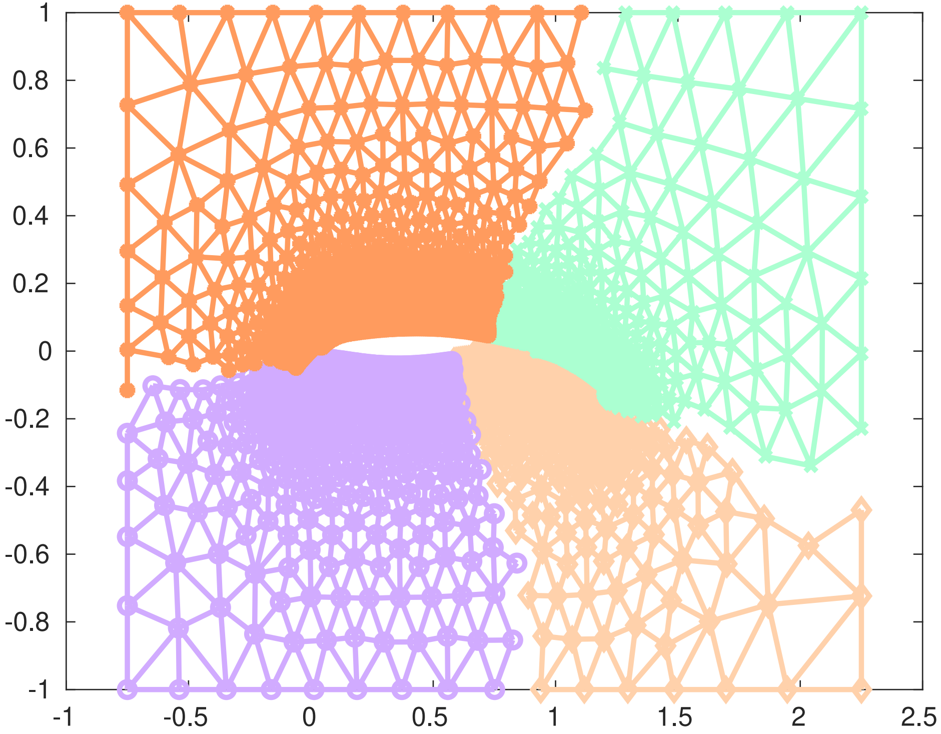
Discretization

↓

Linearization (Newton)

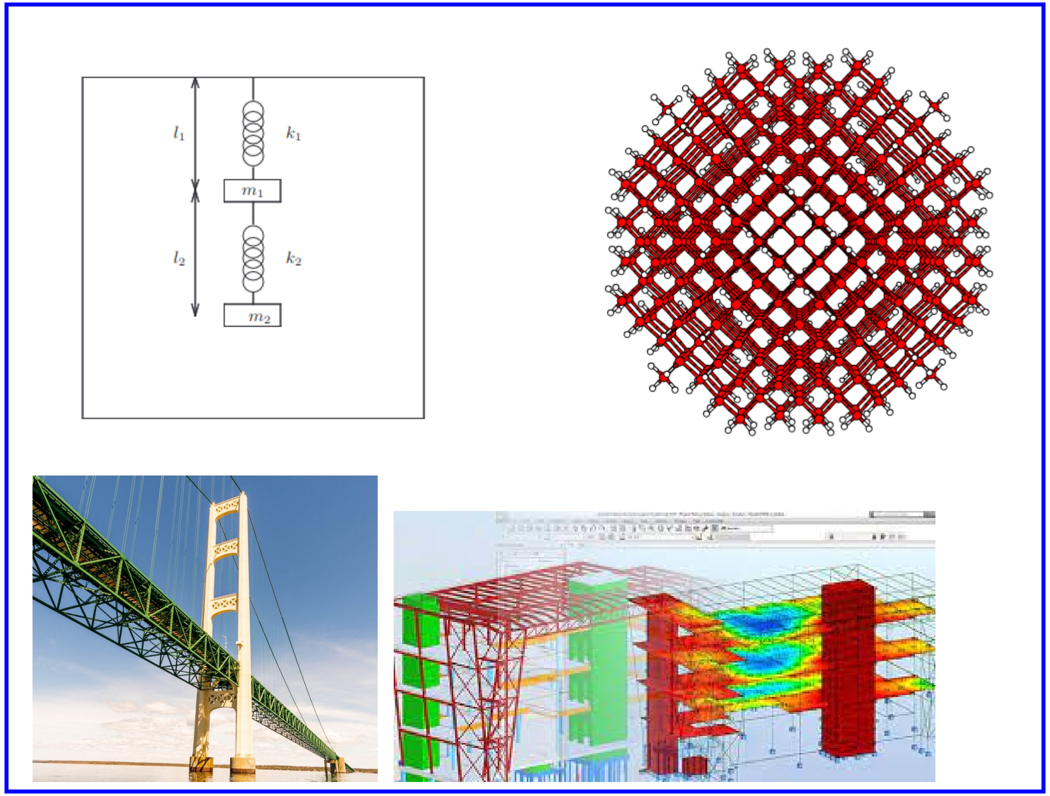
↓

Sparse Linear Systems $Ax = b$



Example: Eigenvalue Problems

➤ Many applications require the computation of a few eigenvalues + associated eigenvectors of a matrix A

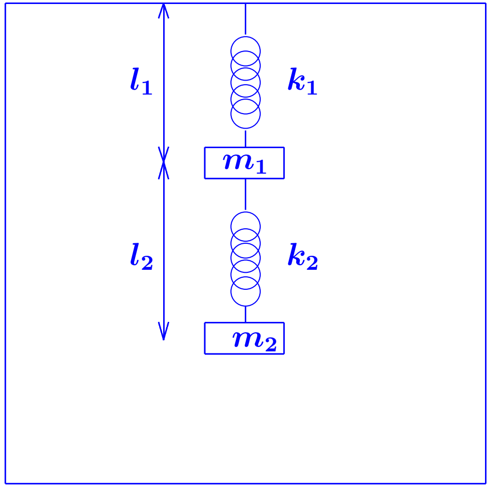


- Structural Engineering – (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..] – Quantum chemistry
- Stability analysis [e.g., electrical networks, mechanical system,..]
- ...

Example: Vibrations

- Vibrations in mechanical systems. See: www.cs.umn.edu/~saad/eig_book_2ndEd.pdf

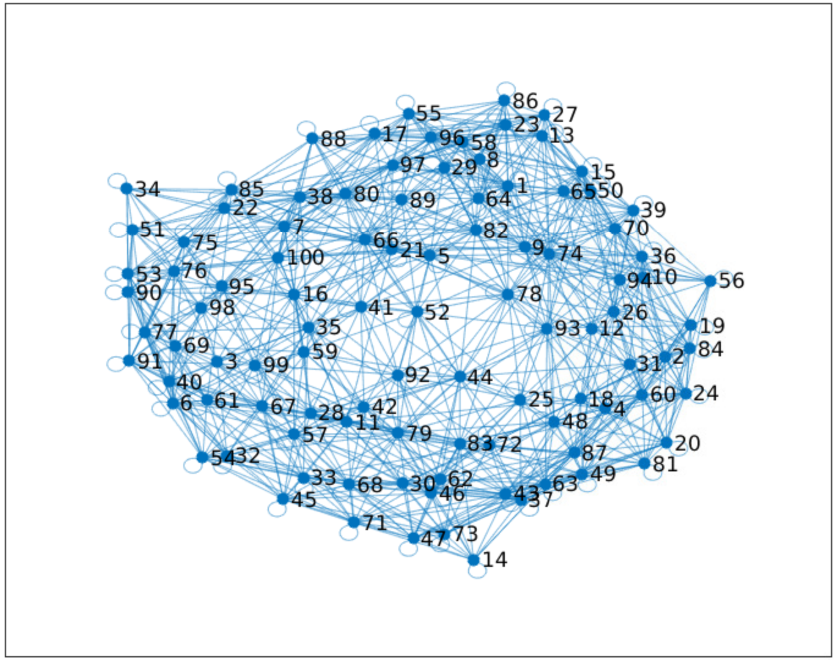
Problem: Determine the vibration modes of the mechanical system [to avoid resonance]. See details in Chapter 10 (sec. 10.2) of above reference.



- Problem type: Eigenvalue Problem

Example: Google Rank (pagerank)

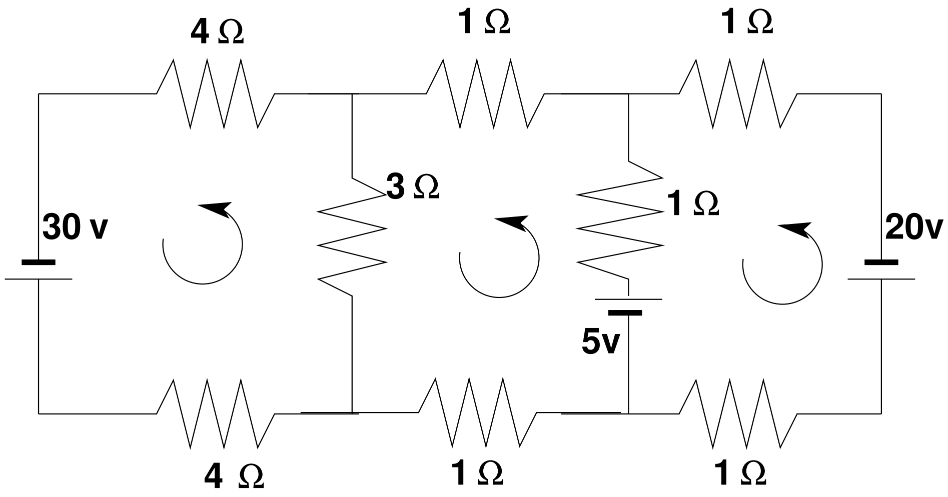
If one were to do a random walk from web page to web page, following each link on a given web page at random with equal likelihood, which are the pages to be encountered this way most often?



➤ Problem type: (homogeneous) Linear system. Eigenvector problem.

Example: Power networks

➤ Electrical circuits .. [Kirchhiff’s voltage Law]

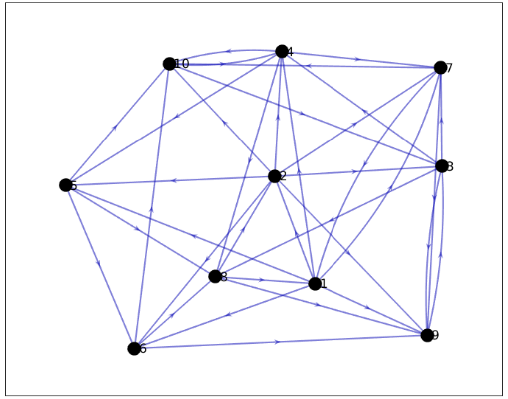


Problem: Determine the loop currents in a an electrical circuit - using Kirchhoff’s Law ($V = RI$)

➤ Problem: Sparse Linear Systems [at the origin Sparse Direct Methods]

Example: Economics/ Marketing/ Social Networks

- Given: an influence graph G: g_{ij} = strength of influence of j over i
- Goal: charge member i price p_i in order to maximize profit
- Utility for member i: [x_i = consumption of i]



$$u_i = ax_i - bx_i^2 + \sum_{j \neq i} g_{ij} x_j - p_i x_i$$

- 1: 'Monopolist' fixes prices; 2: agent i fixes consumption x_i

Result: Optimal pricing proportional to Bonacich centrality: $(I - \alpha G)^{-1} \mathbb{1}$ where $\alpha = \frac{1}{2b}$ [*Candogan et al., 2012* + many refs.]

- ’centrality’ defines a measure of importance of a node (or an edge) in a graph
- Many other ideas of centrality in graphs [degree centrality, betweenness centrality, closeness centrality,
- Important application: Social Network Analysis

Example: Method of least-squares

➤ First use of least squares by Gauss, in early 1800's:

A planet follows an elliptical orbit according to $ay^2+bxy+cx+dy+e=x^2$ in cartesian coordinates. Given a set of noisy observations of (x,y) positions, compute a,b,c,d,e, and use to predict future positions of the planet. This least squares problem is nearly rank-deficient and hence very sensitive to perturbations in the observations.

➤ Problem type: Least-Squares system

Read Wikipedia's article on planet ceres:
http://en.wikipedia.org/wiki/Ceres_(dwarf_planet)

Example: Dynamical systems and epidemiology

A set of variables that fill a vector y are governed by the equation

$$\frac{dy}{dt} = Ay$$

Determine $y(t)$ for $t > 0$, given $y(0)$ [called ‘orbit’ of y]

- ➤ Problem type: (Linear) system of ordinary differential equations.

Solution:

$$y(t) = e^{tA} y(0)$$

- ➤ Involves exponential of A [think Taylor series], i.e., a matrix function

- This is the simplest form of dynamical systems (linear).
- Consider the slightly more complex system:

$$\frac{dy}{dt} = A(y)y$$

- Nonlinear. Requires ‘integration scheme’.
- Next: a little digression into our interesting times...

Example: The SIR model in epidemiology

A population of N individuals, with $N = S + I + R$ where:

S Susceptible population. These are susecptible to being contaminated by others (not immune).

I Infectious population: will contaminate susceptible individuals.

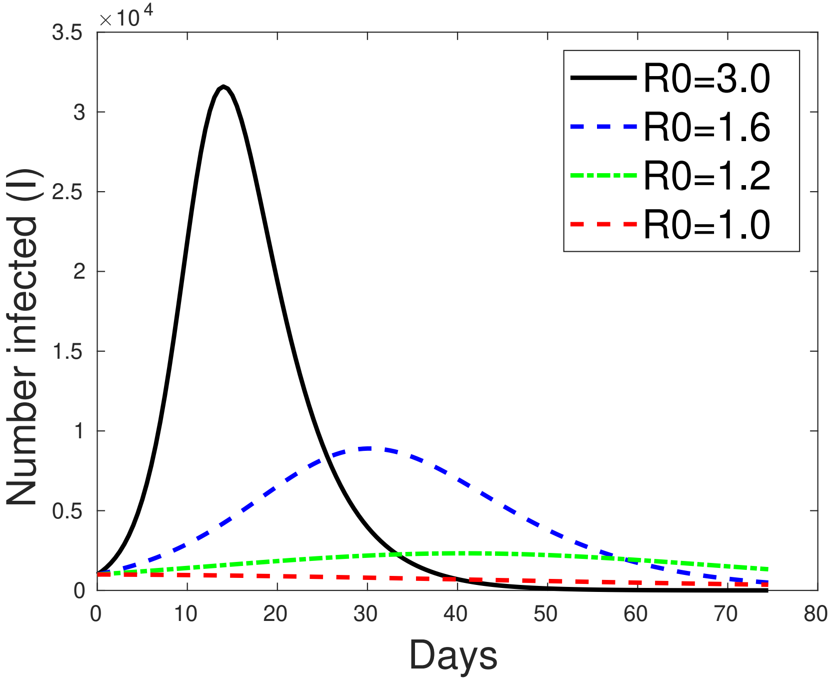
R ‘Removed’ population: either deceased or recovered. These will no longer contaminate others.

Three equations:

$$\frac{dS}{dt} = -\beta I S; \quad \frac{dI}{dt} = (\beta S - \mu) I; \quad \frac{dR}{dt} = \mu I$$

$1/\mu$ = infection period; $\beta = \mu R_0 / N$; R_0 = reproduction number.

- The importance of reducing R_0 (a.k.a. “social distancing”):



- See the latest on this ($R_0 \approx 8.2$ for variant BA.1 and ≈ 12 for BA.2 !!)

- ... and keep away from each other

Problems in Numerical Linear Algebra

- Linear systems: $Ax = b$. Often: A is large and sparse
- Least-squares problems $\min \|b - Ax\|_2$
- Eigenvalue problem $Ax = \lambda x$. Several variations -
- SVD .. and
- ... Low-rank approximation
- Tensors and low-rank tensor approximation
- Matrix equations: Sylvester, Lyapunov, Riccati, ..
- Nonlinear equations – acceleration methods
- Matrix functions and applications
- Many many more ...

SPARSE MATRICES ; DATA STRUCTURES

What are sparse matrices?

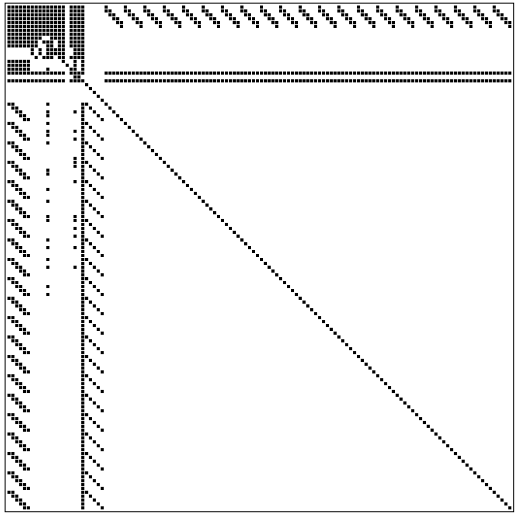
Vague definition: “..matrices that allow special techniques to take advantage of the large number of zero elements and the structure.”

A few applications of sparse matrices: Structural Engineering, Reservoir simulation, Electrical Networks, optimization problems, ...

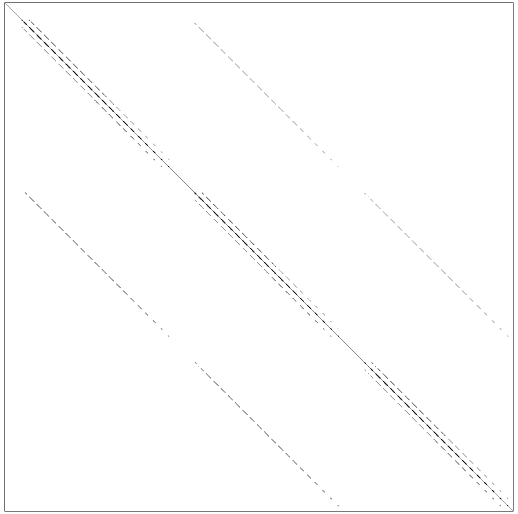
Goals: Much less storage and work than dense computations.

Observation: A^{-1} is usually dense, but L and U in the LU factorization may be reasonably sparse (if a good technique is used).

Sample sparsity patterns



ARC130: Unsymmetric matrix from laser problem. a.r.curtis, oct 1974



SHERMAN5: fully implicit black oil simulator 16 by 23 by 3 grid, 3 unk

Sparse matrices in Matlab

✍ Explore the scripts `Lap2D`, `mark` (provided in matlab suite) for generating sparse matrices

✍ Explore the command `spy`

✍ Explore the command `sparse`

✍ Run the demos titled `demo_sparse0` and `demo_sparse1`

✍ Load the matrix `can_256.mat` from the SuiteSparse collection. Show its pattern

Sparse matrices - continued

- *Main goal of Sparse Matrix Techniques:* To perform standard matrix computations economically, i.e., without storing the zeros

- *Example:* To add two square dense matrices of size n requires $O(n^2)$ operations. To add two sparse matrices A and B requires $O(nnz(A) + nnz(B))$ where $nnz(X) =$ number of nonzero elements of a matrix X.

- For typical Finite Element /Finite difference matrices, number of nonzero elements is $O(n)$.

Data structures: The coordinate format (COO)

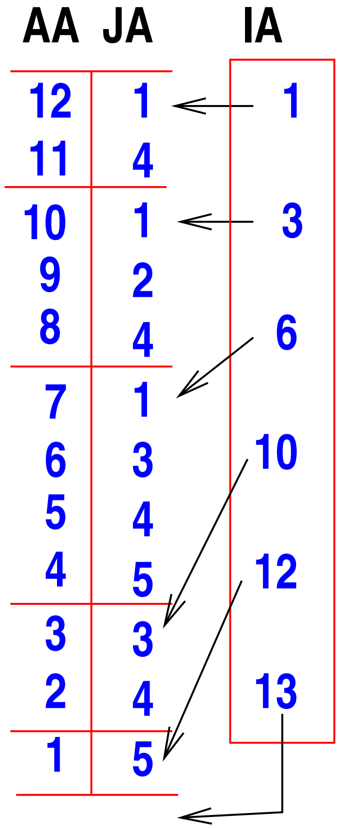
$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

AA	JR	JC
12.	5	5
9.	3	5
7.	3	3
5.	2	4
1.	1	1
2.	1	4
11.	4	4
3.	2	1
6.	3	1
4.	2	2
8.	3	4
10.	4	3

- Also known as ‘triplet format’
- Simple data structure - Often used as ’entry’ format in packages
- Variant used in matlab
- Note: order of entries is arbitrary [in matlab: sorted by columns]

Compressed Sparse Row (CSR) format

$$A = \begin{pmatrix} 12. & 0. & 0. & 11. & 0. \\ 10. & 9. & 0. & 8. & 0. \\ 7. & 0. & 6. & 5. & 4. \\ 0. & 0. & 3. & 2. & 0. \\ 0. & 0. & 0. & 0. & 1. \end{pmatrix}$$



AA	JA
12	1
11	4
10	1
9	2
8	4
7	1
6	3
5	4
4	5
3	3
2	4
1	5

IA
1
3
6
10
12
13

- ➤ IA(j) points to beginning or row j in arrays AA, JA
- ➤ Related: Compressed Sparse Column format, Modified Sparse Row format (MSR).
- ➤ Used predominantly in Fortran & portable codes [e.g. Metis] – what about C?

CSR (CSC) format - C-style

* CSR: Collection of pointers of rows & array of row lengths

```
typedef struct SpaFmt {
/*--------------------------------------------
| C-style CSR format - used internally
| for all matrices in CSR/CSC format
|---------------------------------------------*/
  int n;          /* size of matrix           */
  int *nzcount;   /* length of each row       */
  int **ja;       /* to store column indices  */
  double **ma;    /* to store nonzero entries */
} SparMat;
```

`aa[i][*]` == entries of i-th row (col.);

`ja[i][*]` == col. (row) indices,

`nzcount[i]` == number of nonzero elmts in row (col.) `i`

Data structure used in Csparse [T. Davis' SuiteSparse code]

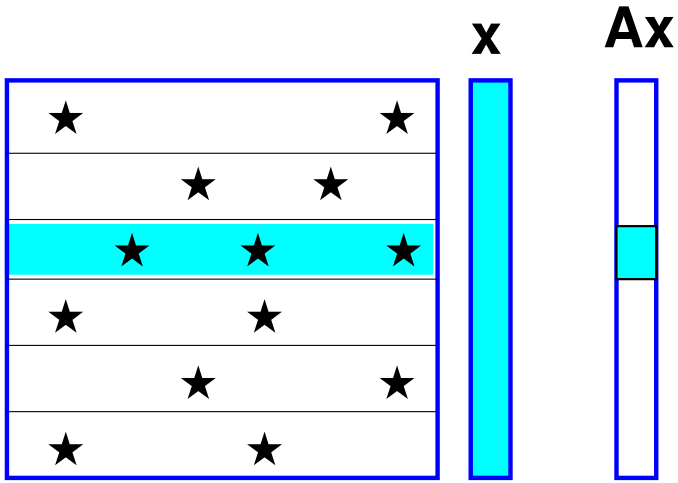
```
typedef struct cs_sparse
{/* matrix in compressed-column or triplet form */
 int nzmax ; /* maximum number of entries */
 int m ;     /* number of rows */
 int n ;     /* number of columns */
 int *p ;    /* column pointers (size n+1) or
                col indices (size nzmax) */
 int *i ;    /* row indices, size nzmax */
 double *x ; /* numerical values, size nzmax */
 int nz ;    /* # of entries in triplet matrix,
                -1 for compressed-col */
} cs ;
```

- Can be used for CSR, CSC, and COO (triplet) storage
- Easy to use from Fortran

Computing $y = Ax$; row and column storage

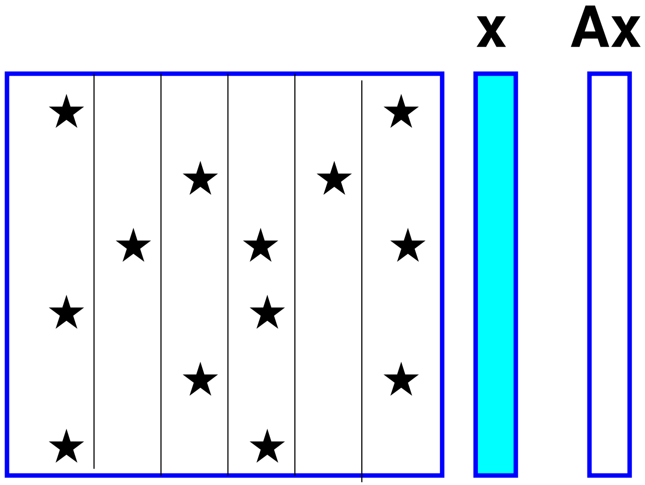
Row-form:

Dot product of $A(i,:)$ and x gives y_i



Column-form:

Linear combination of columns $A(:,j)$ with coefficients x_j yields y



Matvec – row version

```
void matvec( csptr mata, double *x, double *y )
{
    int i, k, *ki;
    double *kr;
    for (i=0; i<mata->n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];
        ki = mata->ja[i];
        for (k=0; k<mata->nzcount[i]; k++)
            y[i] += kr[k] * x[ki[k]];
    }
}
```

- Uses sparse dot products (sparse SDOTS)

✍ Operation count

Matvec – Column version

```
void matvecC( csptr mata, double *x, double *y )
{
  int n = mata->n, i, k, *ki;
  double *kr;
  for (i=0; i<n; i++)
    y[i] = 0.0;
  for (i=0; i<n; i++) {
    kr = mata->ma[i];
    ki = mata->ja[i];
    for (k=0; k<mata->nzcount[i]; k++)
      y[ki[k]] += kr[k] * x[i];
  }
}
```

- Uses sparse vector combinations (sparse SAXPY)

✍ Operation count

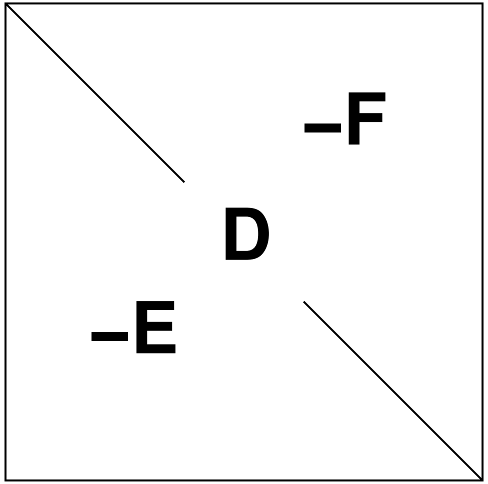
- Using the CS data structure from Suite-Sparse:

```
int cs_gaxpy (cs *A, double *x, double *y) {
 int p, j, n, *Ap, *Ai;
 n = A->n; Ap = A-> p; Ai = A->i; Ax = A->x;
 for (j=0; j<n; j++) {
   for (p=Ap[j]; p<Ap[j+1];p++)
     y[Ai[p]] += Ax[p]*x[j];
 }
return(1)
}
```

BASIC RELAXATION METHODS

Linear Systems: Basic Relaxation Schemes

Relaxation schemes: based on the decomposition $A = D - E - F$



D = diag(A), $-E$ = strict lower part of A and $-F$ its strict upper part.

- For example, Gauss-Seidel iteration :

$$(D - E)x^{(k+1)} = Fx^{(k)} + b$$

- Most common techniques 60 years ago.

- Now: used as smoothers in Multigrid or as preconditioners

Note: If $\rho_i^{(k)} = i$th component of current residual $b - Ax$ then relaxation version of GS is:

$$\xi_i^{(k+1)} = \xi_i^{(k)} + \frac{\rho_i^{(k)}}{a_{ii}}$$

for $i = 1, \cdots, n$

Iteration matrices

➤ Jacobi, Gauss-Seidel, SOR, & SSOR iterations are of the form

$$x^{(k+1)} = Mx^{(k)} + f$$

- $M_{Jac} = D^{-1}(E+F) = I - D^{-1}A$
- $M_{GS}(A) = (D-E)^{-1}F = I - (D-E)^{-1}A$

SOR relaxation: $\xi_i^{(k+1)} = \omega\xi_i^{(GS,k+1)} + (1-\omega)\xi_i^{(k)}$

- $$M_{SOR}(A) = (D-\omega E)^{-1}(\omega F + (1-\omega)D) \\ = I - (\omega^{-1}D - E)^{-1}A$$

✍ Matlab: take a look at: *gs.m*, *sor.m*, and *sorRelax.m* in iters/

An observation & Introduction to Preconditioning

- The iteration $x^{(k+1)} = Mx^{(k)} + f$ is attempting to solve $(I - M)x = f$. Since M is of the form $M = I - P^{-1}A$ this system can be rewritten as

$$P^{-1}Ax = P^{-1}b$$

where for SSOR, we have

$$\boxed{P_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)}$$

referred to as the SSOR 'preconditioning' matrix.

In other words:

Relaxation Scheme $\Longleftrightarrow$ *Preconditioned Fixed Point Iteration*

PROJECTION METHODS

Projection Methods

➤ The main idea of projection methods is to extract an approximate solution from a subspace.

➤ We define a subspace of approximants of dimension m and a set of m conditions to extract the solution

➤ These conditions are typically expressed by orthogonality constraints.

➤ This defines one basic step which is repeated until convergence (alternatively the dimension of the subspace is increased until convergence).

Example: Each relaxation step in Gauss-Seidel can be viewed as a projection step

Background on projectors

- A projector is a linear operator that is idempotent:

$$P^2 = P$$

A few properties:

- P is a projector iff $I - P$ is a projector
- $x \in \mathbf{Ran}(P)$ iff $x = Px$ iff $x \in \mathbf{Null}(I - P)$
- This means that : $\mathbf{Ran}(P) = \mathbf{Null}(I - P)$.
- Any $x \in \mathbb{R}^n$ can be written (uniquely) as $x = x_1 + x_2$, $x_1 = Px \in \mathbf{Ran}(P)$ $x_2 = (I - P)x \in \mathbf{Null}(P)$ - So:

$$\mathbb{R}^n = \mathbf{Ran}(P) \oplus \mathbf{Null}(P)$$

Background on projectors (Continued)

Decomposition $\mathbb{R}^n = K \oplus S$ defines a (unique) projector P:

- From $x = x_1 + x_2$, set $Px = x_1$.
- For this P: $\mathbf{Ran}(P) = K$ and $\mathbf{Null}(P) = S$.
- Note: $dim(K) = m$, $dim(S) = n - m$.

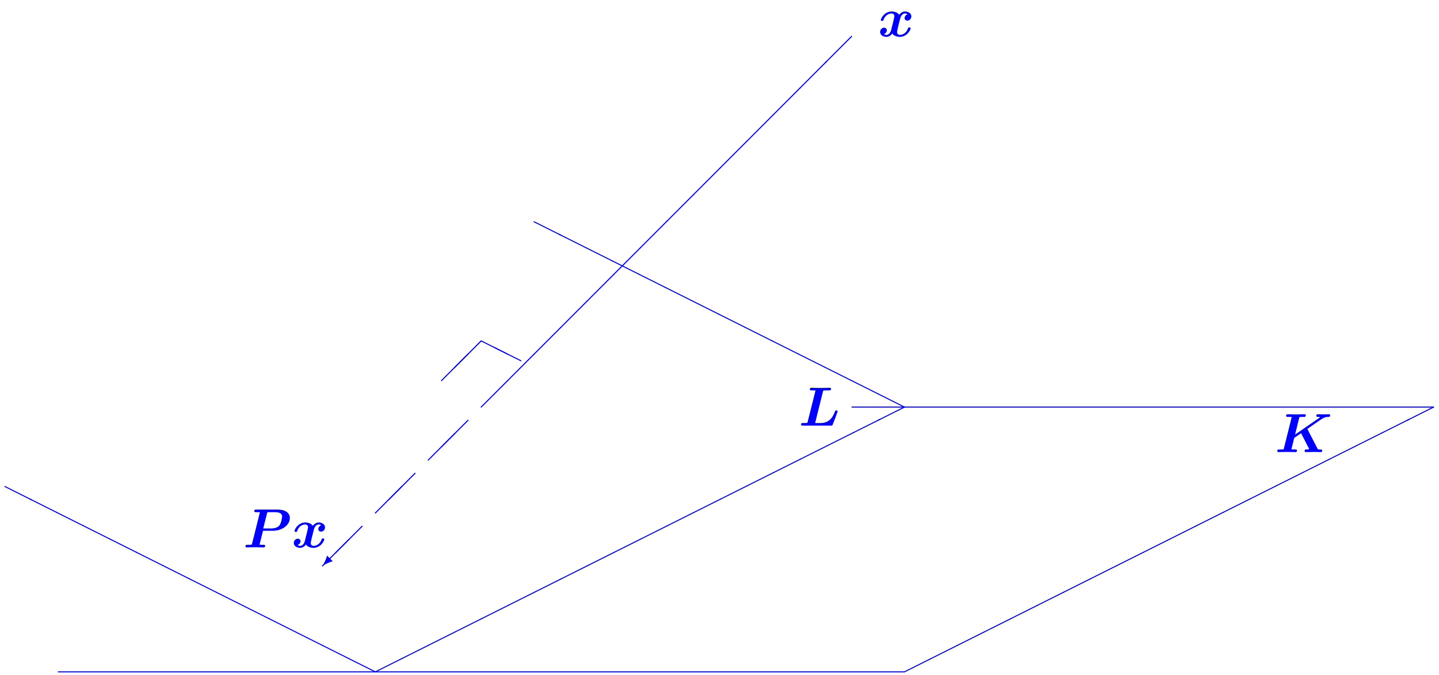
➤ Pb: express mapping $x \rightarrow u = Px$ in terms of K, S

➤ Note $u \in K$, $x - u \in S$

➤ Express 2nd part with m constraints: let $L = S^{\perp}$, then

$$u = Px \text{ iff } \begin{cases} u \in K \\ x - u \perp L \end{cases}$$

➤ Projection onto K and orthogonally to L



- Illustration: P projects onto K and orthogonally to L
- When $L = K$ projector is orthogonal.
- Note: $Px = 0$ iff $x \perp L$.

Projection methods for linear systems

- Initial Problem: $\boxed{b - Ax = 0}$
- Given two subspaces K and L of $\mathbb{R}^N$ of dimension m, define ...

Approximate problem: Find $\tilde{x} \in K$ such that $\underbrace{b - A\tilde{x} \perp L}_{\text{Petrov-Galerkin cond.}}$

- m degrees of freedom (K) + m constraints (L) $\rightarrow$
- To solve: A small linear system (‘projected problem’)
- Basic projection step. Typically a sequence of such steps are applied

- With a nonzero initial guess x_0, approximate problem is

$$\text{Find} \quad \tilde{x} \in x_0 + K \quad \text{such that} \quad b - A\tilde{x} \perp L$$

Write $\tilde{x} = x_0 + \delta$ and $r_0 = b - Ax_0$. → system for δ:

$$\boxed{\text{Find } \delta \ \in K \text{ such that } r_0 - A\delta \perp L}$$

✍ Formulate Gauss-Seidel as a projection method -

✍ Generalize Gauss-Seidel by defining subspaces consisting of 'blocks' of coordinates $\mathbf{span}\{e_i, e_{i+1}, ..., e_{i+p}\}$

Matrix representation:

Let

- $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m]$ a basis of $\boldsymbol{K}$ &
- $\boldsymbol{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m]$ a basis of $\boldsymbol{L}$

➤ Write approximate solution as $\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{\delta} \equiv \boldsymbol{x}_0 + \boldsymbol{V}\boldsymbol{y}$ where $\boldsymbol{y} \in \mathbb{R}^m$. Then Petrov-Galerkin condition yields:

$$\boldsymbol{W}^T(\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}\boldsymbol{y}) = \boldsymbol{0}$$

➤ Therefore,

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{V}[\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{V}]^{-1}\boldsymbol{W}^T\boldsymbol{r}_0$$

Remark: In practice $\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{V}$ is known from algorithm and has a simple structure [tridiagonal, Hessenberg,..]

Prototype Projection Method

Until Convergence Do:

1. Select a pair of subspaces K, and L;

2. Choose bases: $V = [v_1, \ldots, v_m]$ for K and $W = [w_1, \ldots, w_m]$ for L.

3. Compute :

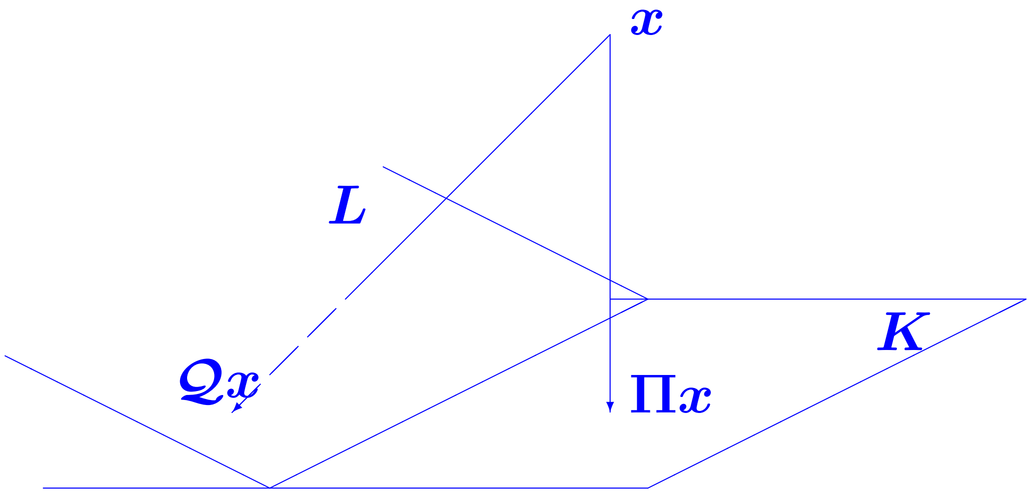
$$r \leftarrow b - Ax,$$
$$y \leftarrow (W^T AV)^{-1} W^T r,$$
$$x \leftarrow x + Vy.$$

Projection methods: Operator form representation

- Let Π = the orthogonal projector onto K and

Q the (oblique) projector onto K and orthogonally to L.

$$\boxed{\begin{array}{ll} \Pi x \in K, & x - \Pi x \perp K \\ Q x \in K, & x - Q x \perp L \end{array}}$$



Assumption: no vector of K is $\perp$ to L

In the case $x_0 = 0$, approximate problem amounts to solving

$$\boxed{\mathcal{Q}(b - Ax) = 0, \quad x \quad \in K}$$

or in operator form (solution is Πx)

$$\boxed{\mathcal{Q}(b - A\Pi x) = 0}$$

Question: what accuracy can one expect?

- Let x^* be the exact solution. Then

1) We cannot get better accuracy than $\|(I - \Pi)x^*\|_2$, i.e.,

$$\boxed{\|\tilde{x} - x^*\|_2 \geq \|(I - \Pi)x^*\|_2}$$

2) The residual of the exact solution for the approximate problem satisfies:

$$\boxed{\|b - QA\Pi x^*\|_2 \leq \|QA(I - \Pi)\|_2 \|(I - \Pi)x^*\|_2}$$

Two Important Particular Cases.

1. $L = K$

- When A is SPD then $\|x^* - \tilde{x}\|_A = \min_{z\in K} \|x^* - z\|_A$.
- Class of Galerkin or Orthogonal projection methods
- Important member of this class: Conjugate Gradient (CG) method

2. $L = AK$.

In this case $\|b - A\tilde{x}\|_2 = \min_{z\in K} \|b - Az\|_2$

- Class of Minimal Residual Methods: CR, GCR, ORTHOMIN, GMRES, CGNR, ...

One-dimensional projection processes

$$K = span\{d\}$$
and
$$L = span\{e\}$$

Then $\tilde{x} = x + \alpha d$. Condition $r - A\delta \perp e$ yields

$$\boxed{\alpha = \frac{(r,e)}{(Ad,e)}}$$

- Three popular choices:

(1) Steepest descent

(2) Minimal residual iteration

(3) Residual norm steepest descent

1. Steepest descent.

A is SPD. Take at each step $d = r$ and $e = r$.

Iteration: $\boxed{\begin{array}{l} r \leftarrow b - Ax, \\ \alpha \leftarrow (r,r)/(Ar,r) \\ x \leftarrow x + \alpha r \end{array}}$

➤ Each step minimizes $f(x) = \|x - x^*\|_A^2 = (A(x - x^*), (x - x^*))$ in direction $-\nabla f$.

➤ Convergence guaranteed if A is SPD.

✍ As is formulated, the above algorithm requires 2 ‘matvecs’ per step. Reformulate it so only one is needed.

Convergence based on the Kantorovitch inequality: Let B be an SPD matrix, λ_{max}, λ_{min} its largest and smallest eigenvalues. Then,

$$\frac{(Bx,x)(B^{-1}x,x)}{(x,x)^2} \le \frac{(\lambda_{max}+\lambda_{min})^2}{4\,\lambda_{max}\lambda_{min}}, \quad \forall x \neq 0.$$

- This helps establish the convergence result

Let A an SPD matrix. Then, the A-norms of the error vectors $d_k = x_* - x_k$ generated by steepest descent satisfy:

$$\|d_{k+1}\|_A \le \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \|d_k\|_A$$

- Algorithm converges for any initial guess x_0.

Proof: Observe $\|d_{k+1}\|_A^2 = (Ad_{k+1}, d_{k+1}) = (r_{k+1}, d_{k+1})$

- by substitution,

$$\|d_{k+1}\|_A^2 = (r_{k+1}, d_k - \alpha_k r_k)$$

- By construction $r_{k+1} \perp r_k$ so we get $\|d_{k+1}\|_A^2 = (r_{k+1}, d_k)$. Now:

$$\begin{aligned} \|d_{k+1}\|_A^2 &= (r_k - \alpha_k A r_k, d_k) \\ &= (r_k, A^{-1} r_k) - \alpha_k (r_k, r_k) \\ &= \|d_k\|_A^2 \left(1 - \frac{(r_k, r_k)}{(r_k, A r_k)} \times \frac{(r_k, r_k)}{(r_k, A^{-1} r_k)} \right) . \end{aligned}$$

Result follows by applying the Kantorovich inequality. ■

2. Minimal residual iteration.

A positive definite ($A + A^T$ is SPD). Take at each step $d = r$ and $e = Ar$.

Iteration: $$\boxed{\begin{array}{l} r \leftarrow b - Ax, \\ \alpha \leftarrow (Ar, r)/(Ar, Ar) \\ x \leftarrow x + \alpha r \end{array}}$$

➤ Each step minimizes $f(x) = \|b - Ax\|_2^2$ in direction r.

➤ Converges under the condition that $A + A^T$ is SPD.

✍ As is formulated, the above algorithm would require 2 'matvecs' at each step. Reformulate it so that only one matvec is required

Convergence

Let A be a real positive definite matrix, and let

$$\mu = \lambda_{min}(A + A^T)/2, \quad \sigma = \|A\|_2.$$

Then the residual vectors generated by the Min. Res. Algorithm satisfy:

$$\|r_{k+1}\|_2 \leq \left(1 - \frac{\mu^2}{\sigma^2}\right)^{1/2} \|r_k\|_2$$

- In this case Min. Res. converges for any initial guess x_0.

Proof: Similar to steepest descent. Start with

$$\begin{aligned}\|r_{k+1}\|_2^2 &= (r_{k+1}, r_k - \alpha_k A r_k) \\ &= (r_{k+1}, r_k) - \alpha_k (r_{k+1}, A r_k).\end{aligned}$$

By construction, $r_{k+1} = r_k - \alpha_k A r_k$ is $\perp$ $A r_k$, so: $\|r_{k+1}\|_2^2 = (r_{k+1}, r_k) = (r_k - \alpha_k A r_k, r_k)$. Then:

$$\begin{aligned}\|r_{k+1}\|_2^2 &= (r_k, r_k) - \alpha_k (A r_k, r_k) \\ &= \|r_k\|_2^2 \left(1 - \frac{(A r_k, r_k)}{(r_k, r_k)} \frac{(A r_k, r_k)}{(A r_k, A r_k)}\right) \\ &= \|r_k\|_2^2 \left(1 - \frac{(A r_k, r_k)^2}{(r_k, r_k)^2} \frac{\|r_k\|_2^2}{\|A r_k\|_2^2}\right).\end{aligned}$$

Result follows from the inequalities $(Ax, x)/(x, x) \geq \mu > 0$ and $\|A r_k\|_2 \leq \|A\|_2 \, \|r_k\|_2$. ■

3. Residual norm steepest descent.

A is arbitrary (nonsingular). Take at each step $d = A^T r$ and $e = Ad$.

Iteration:
$$\boxed{\begin{array}{l} r \leftarrow b - Ax, d = A^T r \\ \alpha \leftarrow \|d\|_2^2 / \|Ad\|_2^2 \\ x \leftarrow x + \alpha d \end{array}}$$

- Each step minimizes $f(x) = \|b - Ax\|_2^2$ in direction $-\nabla f$.
- Important Note: equivalent to usual steepest descent applied to normal equations $A^T A x = A^T b$.
- Converges under the condition that A is nonsingular.

KRYLOV SUBSPACE METHODS

Motivation

- One-dimensional projection techniques: $$x_{new} = x + \alpha d$$ where d = a certain direction.
- α is defined to optimize a certain function.
- Equivalently: determine α by an orthogonality constraint

Example

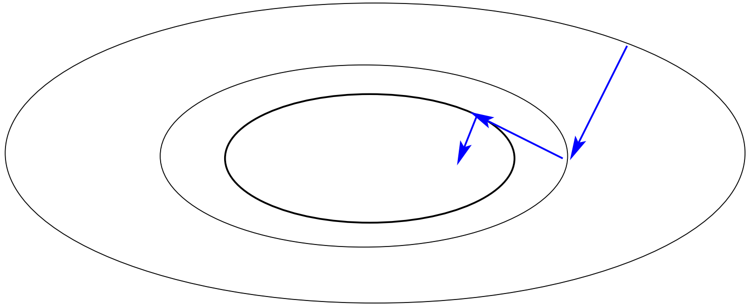
In MR:
$x(\alpha) = x + \alpha d$, with $d = b - Ax$.
$\min_\alpha \ \|b - Ax(\alpha)\|_2$ reached iff $b - Ax(\alpha) \perp r$

- One-dimensional projection methods are greedy methods. They are 'short-sighted'.

Example:

Recall in Steepest Descent: New direction of search $\tilde{r}$ is ⊥ to old direction of search r.

$$\boxed{\begin{array}{l} r \leftarrow b - Ax, \\ \alpha \leftarrow (r, r)/(Ar, r) \\ x \leftarrow x + \alpha r \end{array}}$$



Question: can we do better by combining successive iterates?

- Yes: Krylov subspace methods..

Krylov subspace methods: Introduction

➤ Consider MR (or steepest descent). At each iteration:

$$\begin{aligned} r_{k+1} &= b - A(x^{(k)} + \alpha_k r_k) \\ &= r_k - \alpha_k A r_k \\ &= (I - \alpha_k A) r_k \end{aligned}$$

➤ In the end: $r_{k+1} = (I - \alpha_k A)(I - \alpha_{k-1} A) \cdots (I - \alpha_0 A) r_0 = p_{k+1}(A) r_0$ where $p_{k+1}(t)$ is a polynomial of degree $k+1$ of the form

$$p_{k+1}(t) = 1 - t q_k(t)$$

✍ Show that: $x^{(k+1)} = x^{(0)} + q_k(A) r_0$, with deg $(q_k) = k$

➤ Krylov subspace methods: iterations of this form that are 'optimal' [from m-dimensional projection methods]

Krylov subspace methods

Principle: Projection methods on Krylov subspaces:

$$K_m(A, v_1) = \text{span}\{v_1, Av_1, \cdots, A^{m-1}v_1\}$$

- The most important class of iterative methods.
- Many variants exist depending on the subspace L.

Simple properties of K_m [$\mu \equiv$ deg. of minimal polynomial of v_1.]

- $K_m = \{p(A)v_1 | p = \text{polynomial of degree} \leq m - 1\}$
- $K_m = K_\mu$ for all $m \geq \mu$. Moreover, K_μ is invariant under A.
- $dim(K_m) = m$ iff $\mu \geq m$.

Arnoldi's algorithm

- Goal: to compute an orthogonal basis of K_m.
- Input: Initial vector v_1, with $\|v_1\|_2 = 1$ and m.

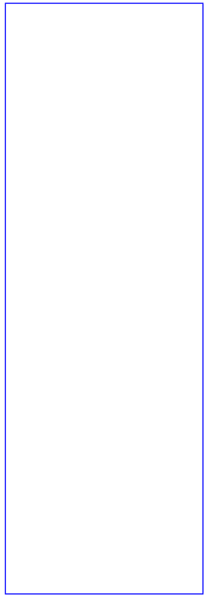
ALGORITHM : 1. *Arnoldi*

```
for $j = 1, ..., m$ do
    Compute $w := Av_j$
    for $i = 1, \ldots, j$ do
        $h_{i,j} := (w, v_i)$
        $w := w - h_{i,j} v_i$
    end for
    Compute: $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$
end for
```

Result of orthogonalization process (Arnoldi):

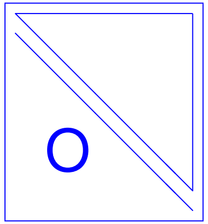
1. $V_m = [v_1, v_2, ..., v_m]$ orthonormal basis of K_m.
2. $AV_m = V_{m+1}\overline{H}_m$
3. $V_m^T A V_m = H_m \equiv \overline{H}_m -$ last row.

V_m =



$$AV_m = V_{m+1}\overline{H}_m$$

$\overline{H}_m =$



$$V_{m+1} = [V_m, v_{m+1}]$$

Arnoldi's Method for linear systems $(L_m = K_m)$

From Petrov-Galerkin condition when $L_m = K_m$, we get

$$\boxed{x_m = x_0 + V_m H_m^{-1} V_m^T r_0}$$

➤ Select $v_1 = r_0/\|r_0\|_2 \equiv r_0/\beta$ in Arnoldi's. Then

$$\boxed{x_m = x_0 + \beta V_m H_m^{-1} e_1}$$

✍ What is the residual vector $r_m = b - Ax_m$?

Several algorithms mathematically equivalent to this approach:

* FOM [Y. Saad, 1981] (above formulation), Young and Jea's ORTHORES [1982], Axelsson's projection method [1981],..

* Also Conjugate Gradient method [see later]

Minimal residual methods $(L_m = AK_m)$

When $L_m = AK_m$, we let $W_m \equiv AV_m$ and obtain relation

$$\begin{aligned} x_m &= x_0 + V_m[W_m^T A V_m]^{-1} W_m^T r_0 \\ &= x_0 + V_m[(AV_m)^T A V_m]^{-1} (AV_m)^T r_0. \end{aligned}$$

➤ Use again $v_1 := r_0/(\beta := \|r_0\|_2)$ and the relation

$$\boxed{AV_m = V_{m+1}\overline{H}_m}$$

➤ $x_m = x_0 + V_m[\bar{H}_m^T \bar{H}_m]^{-1} \bar{H}_m^T \beta e_1 = x_0 + V_m y_m$
where y_m minimizes $\|\beta e_1 - \bar{H}_m y\|_2$ over $y \in \mathbb{R}^m$.

- Gives the Generalized Minimal Residual method (GMRES) ([YS-Schultz,'86]):

$$\begin{aligned} \boldsymbol{x}_m &= \boldsymbol{x}_0 + \boldsymbol{V}_m \boldsymbol{y}_m \quad \text{where} \\ \boldsymbol{y}_m &= \min_{\boldsymbol{y}} \|\beta \boldsymbol{e}_1 - \bar{\boldsymbol{H}}_m \boldsymbol{y}\|_2 \end{aligned}$$

- Several Mathematically equivalent methods:

• Axelsson's CGLS	• Orthomin (1980)
• Orthodir	• GCR

The symmetric case: Observation

Observe: When A is real symmetric then in Arnoldi's method:

$$\boxed{H_m = V_m^T A V_m}$$

must be symmetric. Therefore

Theorem. When Arnoldi's algorithm is applied to a (real) symmetric matrix then the matrix H_m is symmetric tridiagonal:

$$\begin{aligned} h_{ij} &= 0 \quad 1 \le i < j-1; \quad \text{and} \\ h_{j,j+1} &= h_{j+1,j},\ j = 1, \ldots, m \end{aligned}$$

- We can write

$$H_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{bmatrix} \tag{1}$$

The v_i's satisfy a 3-term recurrence [Lanczos Algorithm]:

$$\beta_{j+1} v_{j+1} = A v_j - \alpha_j v_j - \beta_j v_{j-1}$$

- Simplified version of Arnoldi's algorithm for sym. systems.

Symmetric matrix + Arnoldi → Symmetric Lanczos

The Lanczos algorithm

ALGORITHM : 2. Lanczos

```
1. Choose an initial vector $v_1$, s.t. $\|v_1\|_2 = 1$
        Set $\beta_1 \equiv 0, v_0 \equiv 0$
2. For $j = 1, 2, \ldots, m$ Do:
3.      $w_j := Av_j - \beta_j v_{j-1}$
4.      $\alpha_j := (w_j, v_j)$
5.      $w_j := w_j - \alpha_j v_j$
6.      $\beta_{j+1} := \|w_j\|_2$. If $\beta_{j+1} = 0$ then Stop
7.      $v_{j+1} := w_j / \beta_{j+1}$
8. EndDo
```

Lanczos algorithm for linear systems

➤ Usual orthogonal projection method setting:

- $L_m = K_m = span\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$
- Basis $V_m = [v_1, \ldots, v_m]$ of K_m generated by the Lanczos algorithm

➤ Three different possible implementations.

(1) Arnoldi-like;

(2) Exploit tridiagonal nature of H_m (DIOM);

(3) Conjugate gradient (CG) - derived from (2)

We will skip details and show the CG algorithm

The Conjugate Gradient Algorithm (A S.P.D.)

ALGORITHM : 3. *Conjugate Gradient Method*

```
Start: $r_0 := b - Ax_0$, $p_0 := r_0$.
while ($x_j$ Not-converged) do
    $\alpha_j := (r_j, r_j)/(Ap_j, p_j)$
    $x_{j+1} := x_j + \alpha_j p_j$
    $r_{j+1} := r_j - \alpha_j A p_j$
    $\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
    $p_{j+1} := r_{j+1} + \beta_j p_j$
end while
```

- $r_j = scaling \times v_{j+1} \rightarrow$ the r_j's are orthogonal.
- The p_j's are A-conjugate, i.e., $(Ap_i, p_j) = 0$ for $i \neq j$.

A bit of history. From the 1952 CG article:

> *“The method of conjugate gradients was developed independently by E. Stiefel of the Institute of Applied Mathematics at Zurich and by M. R. Hestenes with the cooperation of J. B. Rosser, G. Forsythe, and L. Paige of the Institute for Numerical Analysis, National Bureau of Standards. (...) The first papers on this method were given by E. Stiefel [1952] and by M. R. Hestenes [1951]. Reports on this method were given by E. Stiefel and J. B. Rosser at a Symposium on August 23-25, 1951. Recently, C. Lanczos [1952] developed a closely related routine based on his earlier paper on eigenvalue problem [1950]. Examples and numerical tests of the method have been by R. Hayes, U. Hoschstrasser, and M. Stein.”*

SOLUTION OF EIGENVALUE PROBLEMS

Background. Origins of Eigenvalue Problems

- Structural Engineering [$Ku = \lambda Mu$] (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..]
- Stability analysis [e.g., electrical networks, mechanical system,..]
- Bifurcation analysis [e.g., in fluid flow]

➤ Large eigenvalue problems in quantum chemistry use up biggest portion of the time in supercomputer centers

Background. New applications in data analytics

➤ Machine learning problems often require a (partial) *Singular Value Decomposition* -

➤ Somewhat different issues in this case:

- Very large matrices, update the SVD
- Compute dominant singular values/vectors
- Many problems of approximating a matrix (or a tensor) by one of lower rank (Dimension reduction, ...)

➤ But: Methods for computing SVD often based on those for standard eigenvalue problems

Background. The Problem (s)

- Standard eigenvalue problem:

$$Ax = \lambda x$$

Often: A is symmetric real (or Hermitian complex)

- Generalized problem $Ax = \lambda Bx$ Often: B is symmetric positive definite, A is symmetric or nonsymmetric

- Quadratic problems: $(A + \lambda B + \lambda^2 C)u = 0$

- Nonlinear eigenvalue problems (NEVP)

$$\boxed{\left[A_0 + \lambda B_0 + \sum_{i=1}^{n} f_i(\lambda) A_i\right] u = 0}$$

- General form of NEVP $A(\lambda)x = 0$

- Nonlinear eigenvector problems:

$$[A + \lambda B + F(u_1, u_2, \cdots, u_k)]u = 0$$

What to compute:

- A few λ_i 's with smallest or largest real parts;
- All λ_i's in a certain region of $\mathbb{C}$;
- A few of the dominant eigenvalues;
- All λ_i's (rare).

Large eigenvalue problems in applications

➤ Some applications require the computation of a large number of eigenvalues and vectors of very large matrices.

➤ Density Functional Theory in electronic structure calculations: *‘ground states’*

➤ *Excited states* involve transitions and invariably lead to much more complex computations. → Large matrices, *many* eigen-pairs to compute

Background: The main tools

Projection process: Rayleigh-Ritz

(a) Build a ‘good’ subspace $\boldsymbol{K} = \mathbf{span}(\boldsymbol{V})$;

(b) get approximate eigenpairs by a Rayleigh-Ritz process:

$$\text{Find } \tilde{\boldsymbol{\lambda}} \in \mathbb{C},\ \tilde{\boldsymbol{u}} \in \boldsymbol{K} \ \text{ such that: } \ (\boldsymbol{A} - \tilde{\boldsymbol{\lambda}} \boldsymbol{I})\tilde{\boldsymbol{u}} \ \perp \ \boldsymbol{K}$$

- Will revisit this shortly

The main tools: Shift-and-invert:

- If we want eigenvalues near σ, replace A by $(A - \sigma I)^{-1}$.

Example: power method: $v_j = Av_{j-1}/\text{scaling}$ replaced by

$$v_j = \frac{(A - \sigma I)^{-1} v_{j-1}}{\text{scaling}}$$

- Works well for computing *a few* eigenvalues near σ/
- Used in commercial package NASTRAN (for decades!)
- Requires factoring $(A - \sigma I)$ (or $(A - \sigma B)$ in generalized case.) But convergence will be much faster.
- A solve each time - Factorization done once (ideally).

The main tools: Deflation / Restarting

Deflation: ➤ Once eigenvectors converge remove them from the picture (e.g., with power method, second largest becomes largest eigenvalue after deflation).

Restarting Strategies:

➤ Restart projection process by using information gathered in previous steps

➤ ALL available methods use some combination of these ingredients.

[e.g. ARPACK: Arnoldi/Lanczos + 'implicit restarts' + shift-and-invert (option).]

Current state-of-the art in eigensolvers

➤ Eigenvalues at one end of the spectrum:

- Subspace iteration + filtering [e.g. FEAST, Cheb,...]
- Lanczos+variants (no restart, thick restart, implicit restart, Davidson,..), e.g., ARPACK code, PRIMME.
- Block Algorithms [Block Lanczos, TraceMin, LOBPCG, SlepSc,...]
- + Many others - more or less related to above

➤ 'Interior' eigenvalue problems (middle of spectrum):

- Combine shift-and-invert + Lanczos/block Lanczos. Used in, e.g., NASTRAN
- Rational filtering [FEAST, Sakurai et al.,..]

Projection Methods for Eigenvalue Problems

General formulation:

- Projection method onto $\boldsymbol{K}$ orthogonal to $\boldsymbol{L}$
- Given: Two subspaces $\boldsymbol{K}$ and $\boldsymbol{L}$ of same dimension.
- Find: $\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{u}}$ such that:

$$\tilde{\boldsymbol{\lambda}} \in \mathbb{C}, \tilde{\boldsymbol{u}} \in \boldsymbol{K}; \quad (\tilde{\boldsymbol{\lambda}} \boldsymbol{I} - \boldsymbol{A})\tilde{\boldsymbol{u}} \perp \boldsymbol{L}$$

Two types of methods:

- Orthogonal projection methods: situation when $\boldsymbol{L} = \boldsymbol{K}$.
- Oblique projection methods: When $\boldsymbol{L} \neq \boldsymbol{K}$.

Rayleigh-Ritz projection

Given: a subspace X known to contain good approximations to eigenvectors of A.

Question: How to extract good approximations to eigenvalues/ eigenvectors from this subspace?

Answer: Rayleigh Ritz process.

Let $Q = [q_1, \ldots, q_m]$ an orthonormal basis of X. Then write an approximation in the form $\tilde{u} = Qy$ and obtain y by writing

$$Q^H(A - \tilde{\lambda}I)\tilde{u} = 0$$

➤ $Q^H AQy = \tilde{\lambda} y$

Procedure:

1. Obtain an orthonormal basis of X
2. Compute $C = Q^H AQ$ (an $m \times m$ matrix)
3. Obtain Schur factorization of C, $C = YRY^H$
4. Compute $\tilde{U} = QY$

Property: if X is (exactly) invariant, then procedure will yield exact eigenvalues and eigenvectors.

Proof: Since X is invariant, $(A - \tilde{\lambda}I)u = Qz$ for a certain z. $Q^H Qz = 0$ implies $z = 0$ and therefore $(A - \tilde{\lambda}I)u = 0$.

➤ Can use this procedure in conjunction with the subspace obtained from subspace iteration algorithm

Subspace Iteration

- *Original idea:* projection technique onto a subspace if the form $\boldsymbol{Y} = \boldsymbol{A}^k\boldsymbol{X}$

- In practice: Replace $\boldsymbol{A}^k$ by suitable polynomial [Chebyshev]

Advantages:
- Easy to implement (in symmetric case);
- Easy to analyze;

Disadvantage: Slow.

- Often used with polynomial acceleration: $\boldsymbol{A}^k\boldsymbol{X}$ replaced by $\boldsymbol{C}_k(\boldsymbol{A})\boldsymbol{X}$. Typically $\boldsymbol{C}_k$ = Chebyshev polynomial.

Algorithm: *Subspace Iteration with Projection*

1. **Start:** Choose an initial system of vectors $X = [x_0, \ldots, x_m]$ and an initial polynomial C_k.

2. **Iterate:** Until convergence do:

   (a) Compute $\hat{Z} = C_k(A)X_{old}$.

   (b) Orthonormalize $\hat{Z}$ into Z.

   (c) Compute $B = Z^H AZ$ and use the QR algorithm to compute the Schur vectors $Y = [y_1, \ldots, y_m]$ of B.

   (d) Compute $X_{new} = ZY$.

   (e) Test for convergence. If satisfied stop. Else select a new polynomial $C'_{k'}$ and continue.

THEOREM: Let $S_0 = span\{x_1, x_2, \ldots, x_m\}$ and assume that S_0 is such that the vectors $\{Px_i\}_{i=1,\ldots,m}$ are linearly independent where P is the spectral projector associated with $\lambda_1, \ldots, \lambda_m$. Let $\mathcal{P}_k$ the orthogonal projector onto the subspace $S_k = span\{X_k\}$. Then for each eigenvector u_i of A, $i = 1, \ldots, m$, there exists a unique vector s_i in the subspace S_0 such that $Ps_i = u_i$. Moreover, the following inequality is satisfied

$$\|(I - \mathcal{P}_k)u_i\|_2 \leq \|u_i - s_i\|_2 \left(\left| \frac{\lambda_{m+1}}{\lambda_i} \right| + \epsilon_k \right)^k , \tag{2}$$

where ϵ_k tends to zero as k tends to infinity.

Krylov subspace methods

Principle: Projection methods on Krylov subspaces:

$$K_m(A, v_1) = \text{span}\{v_1, Av_1, \cdots, A^{m-1}v_1\}$$

- The most important class of iterative methods.
- Many variants exist depending on the subspace L.

Simple properties of K_m [$\mu \equiv$ deg. of minimal polynomial of v_1.]

- $K_m = \{p(A)v_1 | p = \text{polynomial of degree} \le m - 1\}$
- $K_m = K_\mu$ for all $m \ge \mu$. Moreover, K_μ is invariant under A.
- $dim(K_m) = m$ iff $\mu \ge m$.

Arnoldi's Algorithm

- Goal: to compute an orthogonal basis of K_m.
- Input: Initial vector v_1, with $\|v_1\|_2 = 1$ and m.

ALGORITHM : 4. *Arnoldi's procedure*

```
For $j = 1, ..., m$ do
    Compute $w := Av_j$
    For $i = 1, \ldots, j$, do $\begin{cases} h_{i,j} := (w, v_i) \\ w := w - h_{i,j}v_i \end{cases}$
    $h_{j+1,j} = \|w\|_2$; $v_{j+1} = w/h_{j+1,j}$
End
```

Result of Arnoldi's algorithm

Let

$$\overline{H}_m = \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \end{bmatrix} \; ; \quad H_m = \overline{H}_m(1:m, 1:m)$$

1. $V_m = [v_1, v_2, ..., v_m]$ orthonormal basis of K_m.

2. $AV_m = V_{m+1}\overline{H}_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$

3. $V_m^T A V_m = H_m \equiv \overline{H}_m -$ last row.

Appliaction to eigenvalue problems

- Write approximate eigenvector as $\tilde{u} = V_m y$ + Galerkin condition

$$(A - \tilde{\lambda} I) V_m y \perp \mathcal{K}_m \rightarrow V_m^H (A - \tilde{\lambda} I) V_m y = 0$$

- Approximate eigenvalues are eigenvalues of H_m

$$\boxed{H_m y_j = \tilde{\lambda}_j y_j}$$

Associated approximate eigenvectors are

$$\boxed{\tilde{u}_j = V_m y_j}$$

Typically a few of the outermost eigenvalues will converge first.

Restarted Arnoldi

In practice: Memory requirement of algorithm implies restarting is necessary

➤ Restarted Arnoldi for computing rightmost eigenpair:

ALGORITHM : 5 . Restarted Arnoldi

```
1. Start: Choose an initial vector $v_1$ and a dimension $m$.
2. Iterate: Perform $m$ steps of Arnoldi's algorithm.
3. Restart: Compute the approximate eigenvector $u_1^{(m)}$
4.        associated with the rightmost eigenvalue $\lambda_1^{(m)}$.
5.        If satisfied stop, else set $v_1 \equiv u_1^{(m)}$ and goto 2.
```

Deflation

- Very useful in practice.

- Different forms: locking (subspace iteration), selective orthogonalization (Lanczos), Schur deflation, ...

A little background Consider Schur canonical form $A = URU^H$ where U is a (complex) upper triangular matrix.

- Vector columns $u_1, \ldots, u_n$ called Schur vectors.

- Note: Schur vectors are not unique. In particular, they depend on the order of the eigenvalues

Wiedlandt Deflation: Assume we have computed a right eigenpair λ_1, u_1. Wielandt deflation considers eigenvalues of

$$A_1 = A - \sigma u_1 v^H$$

Note:

$$\Lambda(A_1) = \{\lambda_1 - \sigma, \lambda_2, \ldots, \lambda_n\}$$

Wielandt deflation preserves u_1 as an eigenvector as well all the left eigenvectors not associated with λ_1.

- An interesting choice for v is to take simply $v = u_1$. In this case Wielandt deflation preserves Schur vectors as well.
- Can apply above procedure successively.

ALGORITHM : 6. *Explicit Deflation*

```
1. $A_0 = A$
2. For $j = 0 \ldots \mu - 1$ Do:
3.     Compute a dominant eigenvector of $A_j$
4.     Define $A_{j+1} = A_j - \sigma_j u_j u_j^H$
5. End
```

- Computed $u_1, u_2., ..$ form a set of Schur vectors for A.

- In Arnoldi: Accumulate each new converged eigenvector in columns 1, 2, 3, ... ['locked' set of eigenvectors.] + maintain orthogonality

- Alternative: implicit deflation (within a procedure such as Arnoldi).

Deflated Arnoldi

```
For $k = 1, \ldots .NEV$ do: /* Eigenvalue loop */
1. For $j = k, k+1, ..., m$ do: /* Arnoldi loop*/
      • Compute $w := Av_j$.
      • Orthonormalize $w$ against $v_1, v_2, \ldots, v_j \;\rightarrow\; v_{j+1}$
2. Compute next approximate eigenpair $\tilde{\lambda}, \tilde{u}$.
3. Orthonormalize $\tilde{u}$ against $v_1, \ldots, v_j$ ➤ Result = $\tilde{s}$ = approximate
   Schur vector.
4. Define $v_k := \tilde{s}$.
5. If approximation not satisfactory go to 1.
6. Else define $h_{i,k} = (Av_k, v_i)$ , $i = 1, .., k$,
```

Thus, for $k = 2$:

$$V_m = \left[\underbrace{v_1, v_2}_{Locked}, \overbrace{v_3, \ldots, v_m}^{active} \right]$$

$$H_m = \left(\begin{array}{cc|cccc} * & * & * & * & * & * \\ & * & * & * & * & * \\ \hline & & * & * & * & * \\ & & * & * & * & * \\ & & & * & * & * \\ & & & & * & * \end{array} \right)$$

- Similar techniques in Subspace iteration [G. Stewart's SRRIT]

Hermitian case: The Lanczos Algorithm

- The Hessenberg matrix becomes tridiagonal :

$$A = A^H \quad \text{and} \quad V_m^H A V_m = H_m \quad \to H_m = H_m^H \longrightarrow$$

$$H_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & . & . & . & \\ & & & . & . & . \\ & & & & \beta_m & \alpha_m \end{bmatrix}$$

Consequence:

3-term recurrence:

$$\boxed{\beta_{j+1} v_{j+1} = A v_j - \alpha_j v_j - \beta_j v_{j-1}}$$

Hermitian matrix + Arnoldi → Hermitian Lanczos

ALGORITHM : 7. *Lanczos*

```
1. Choose $v_1$ of norm unity. Set $\beta_1 \equiv 0, v_0 \equiv 0$
2. For $j = 1, 2, \ldots, m$ Do:
3.     $w_j := Av_j - \beta_j v_{j-1}$
4.     $\alpha_j := (w_j, v_j)$
5.     $w_j := w_j - \alpha_j v_j$
6.     $\beta_{j+1} := \|w_j\|_2$. If $\beta_{j+1} = 0$ then Stop
7.     $v_{j+1} := w_j / \beta_{j+1}$
8. EndDo
```

- In theory v_i's defined by 3-term recurrence are orthogonal.
- However: in practice severe loss of orthogonality;

Lanczos with reorthogonalization

Observation [Paige, 1981]: Loss of orthogonality starts suddenly, when the first eigenpair converges. It indicates loss of linear indedependence of the v_is. When orthogonality is lost, then several copies of the same eigenvalue start appearing.

Forms of Re-orthogonalization

Full – reorthogonalize v_{j+1} against all previous v_i's every time.

Partial – reorthogonalize v_{j+1} against all previous v_i's only when needed

Selective – reorthogonalize v_{j+1} against computed eigenvectors

None – Do not reorthogonalize - but take measures to deal with 'spurious' eigenvalues.

The Lanczos Algorithm in the Hermitian Case

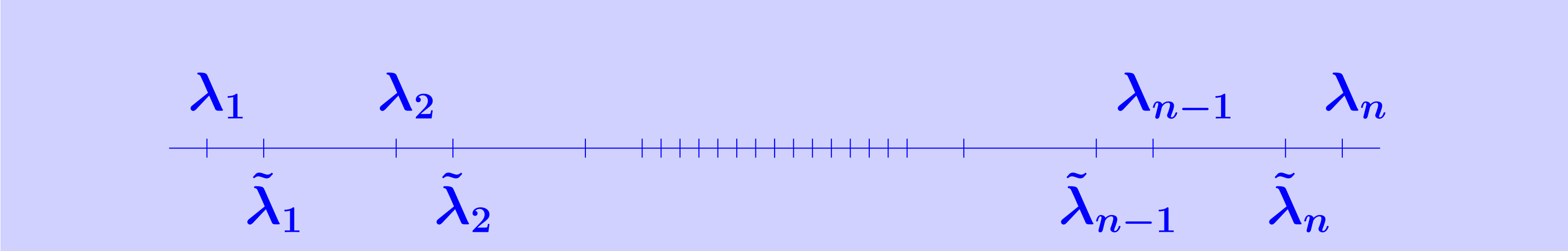
Assume eigenvalues sorted increasingly $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$

- ➤ Orthogonal projection method onto K_m;
- ➤ To derive error bounds, use the Courant characterization

$$\tilde{\lambda}_1 = \min_{u \,\in\, K,\ u \neq 0} \frac{(Au, u)}{(u, u)} = \frac{(A\tilde{u}_1, \tilde{u}_1)}{(\tilde{u}_1, \tilde{u}_1)}$$

$$\tilde{\lambda}_j = \min_{\left\{ \begin{smallmatrix} u & \in\ K,\ u \neq 0 \\ u & \perp \tilde{u}_1, \ldots, \tilde{u}_{j-1} \end{smallmatrix} \right.} \frac{(Au, u)}{(u, u)} = \frac{(A\tilde{u}_j, \tilde{u}_j)}{(\tilde{u}_j, \tilde{u}_j)}$$

- Bounds for λ_1 easy to find – similar to linear systems.
- Ritz values approximate eigenvalues of A inside out:



A-priori error bounds

Theorem [Kaniel, 1966]: Let $\gamma_1 = \frac{\lambda_2 - \lambda_1}{\lambda_N - \lambda_2}$; Then:

$$0 \le \lambda_1^{(m)} - \lambda_1 \le (\lambda_N - \lambda_1) \left[\frac{\tan \angle(v_1, u_1)}{T_{m-1}(1 + 2\gamma_1)} \right]^2$$

Theorem [Kaniel, Paige, YS]. Let $\gamma_i = \frac{\lambda_{i+1} - \lambda_i}{\lambda_N - \lambda_{i+1}}$, $\kappa_i^{(m)} = \prod_{j<i} \frac{\lambda_j^{(m)} - \lambda_N}{\lambda_j^{(m)} - \lambda_i}$ Then:

$$0 \le \lambda_i^{(m)} - \lambda_i \le (\lambda_N - \lambda_1) \left[\kappa_i^{(m)} \frac{\tan \angle(v_i, u_i)}{T_{m-i}(1 + 2\gamma_i)} \right]^2$$

The Lanczos biorthogonalization ($A^H \neq A$)

ALGORITHM : 8. *Lanczos bi-orthogonalization*

```
1.  Choose two vectors $v_1, w_1$ such that $(v_1, w_1) = 1.$
2.  Set $\beta_1 = \delta_1 \equiv 0$, $w_0 = v_0 \equiv 0$
3.  For $j = 1, 2, \ldots, m$ Do:
4.      $\alpha_j = (Av_j, w_j)$
5.      $\hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6.      $\hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7.      $\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. If $\delta_{j+1} = 0$ Stop
8.      $\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1}) / \delta_{j+1}$
9.      $w_{j+1} = \hat{w}_{j+1} / \beta_{j+1}$
10.     $v_{j+1} = \hat{v}_{j+1} / \delta_{j+1}$
11. EndDo
```

- Builds a pair of biorthogonal bases for the two subspaces

$$\mathcal{K}_m(A, v_1) \quad \text{and} \quad \mathcal{K}_m(A^H, w_1)$$

- Many choices for $\delta_{j+1}, \beta_{j+1}$ in lines 7 and 8. Only constraint:

$$\boxed{\delta_{j+1}\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})}$$

Let

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & . & . & . & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{bmatrix}.$$

- $v_i \in \mathcal{K}_m(A, v_1)$ and $w_j \in \mathcal{K}_m(A^T, w_1)$.

If the algorithm does not break down before step m, then the vectors $v_i, i = 1, \ldots, m$, and $w_j, j = 1, \ldots, m$, are biorthogonal, i.e.,

$$(v_j, w_i) = \delta_{ij} \quad 1 \leq i, \; j \leq m \; .$$

Moreover, $\{v_i\}_{i=1,2,\ldots,m}$ is a basis of $\mathcal{K}_m(A, v_1)$ and $\{w_i\}_{i=1,2,\ldots,m}$ is a basis of $\mathcal{K}_m(A^H, w_1)$ and

$$\begin{aligned} AV_m &= V_m T_m + \delta_{m+1} v_{m+1} e_m^H, \\ A^H W_m &= W_m T_m^H + \bar{\beta}_{m+1} w_{m+1} e_m^H, \\ W_m^H A V_m &= T_m \quad . \end{aligned}$$

- If θ_j, y_j, z_j are, respectively an eigenvalue of T_m, with associated right and left eigenvectors y_j and z_j respectively, then corresponding approximations for A are

Ritz value	Right Ritz vector	Left Ritz vector
θ_j	$V_m y_j$	$W_m z_j$

[Note: terminology is abused slightly - Ritz values and vectors normally refer to Hermitian cases.]

Advantages and disadvantages

Advantages:

- Nice three-term recurrence – requires little storage in theory.
- Computes left and a right eigenvectors at the same time

Disadvantages:

- Algorithm can break down or nearly break down.
- Convergence not too well understood. Erratic behavior
- Not easy to take advantage of the tridiagonal form of T_m.

Look-ahead Lanczos

Algorithm breaks down when:

$$(\hat{v}_{j+1}, \hat{w}_{j+1}) = 0$$

Three distinct situations.

- 'lucky breakdown' when either $\hat{v}_{j+1}$ or $\hat{w}_{j+1}$ is zero. In this case, eigenvalues of T_m are eigenvalues of A.
- $(\hat{v}_{j+1}, \hat{w}_{j+1}) = 0$ but of $\hat{v}_{j+1} \neq 0$, $\hat{w}_{j+1} \neq 0 \rightarrow$ serious breakdown. Often possible to bypass the step (+ a few more) and continue the algorithm. If this is not possible then we get an ...
- ... Incurable break-down. [very rare]

BACKGROUND ON GRAPHS

Graphs – definitions & representations

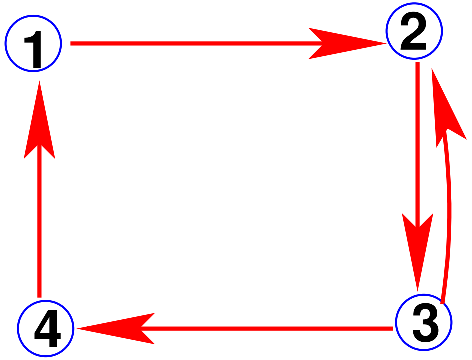
- Graph theory is a fundamental tool in many areas

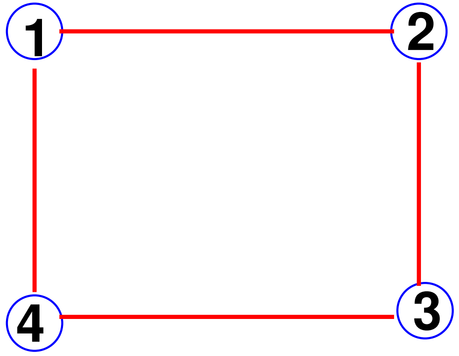
Definition. A graph G is defined as a pair of sets $G = (V, E)$ with $E \subset V \times V$. So G represents a binary relation. The graph is undirected if the binary relation is symmetric. It is directed otherwise.

- V is the vertex set and E is the edge set
- A binary relation R in V can be represente by graph $G = (V, E)$ where:

$$(u, v) \in E \leftrightarrow u \; R \; v$$

Undirected graph ↔ symmetric relation





✍ Given the numbers 5, 3, 9, 15, 16, show the two graphs representing the relations

R1: Either $x < y$ or y divides x.

R2: x and y are congruent modulo 3. [mod(x,3) = mod(y,3)]

- $|E| \le |V|^2$. For undirected graphs: $|E| \le |V|(|V| + 1)/2$.
- A sparse graph is one for which $|E| \ll |V|^2$.

Basic Terminology & notation:

- If $(u, v) \in E$, then v is adjacent to u. The edge (u, v) is incident to u and v.

- If the graph is directed, then (u, v) is an outgoing edge from u and incoming edge to v

- $Adj(i) = \{j | j \text{ adjacent to } i\}$

- The degree of a vertex v is the number of edges incident to v. Can also define the indegree and outdegree. (Sometimes self-edge $i \to i$ omitted)

- $|S|$ is the cardinality of set S [so $|Adj(i)|$ == deg(i)]

- A subgraph $G' = (V', E')$ of G is a graph with $V' \subset V$ and $E' \subset E$.

Representations of Graphs

- A graph is nothing but a collection of vertices (indices from 1 to n), each with a set of its adjacent vertices [in effect a 'sparse matrix without values']

- For sparse graphs: use any of the sparse matrix storage formats - omit the real values arrays.

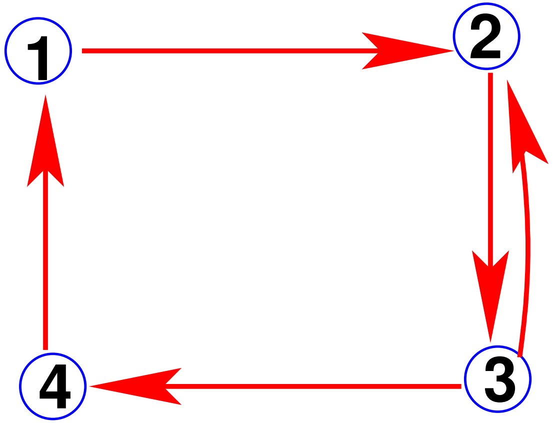
Adjacency matrix Assume $V = \{1, 2, \cdots, n\}$. Then the adjacency matrix of $G = (V, E)$ is the $n \times n$ matrix, with entries:

$$a_{i,j} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

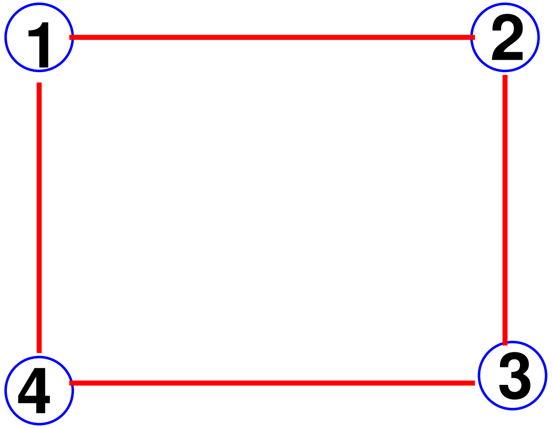
Representations of Graphs (cont.)

Example:

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



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



More terminology & notation

- Given $Y \subset X$, the section graph of Y is the subgraph $G_Y = (Y, E(Y))$ where $E(Y) = \{(x,y) \in E | x \in Y, \quad y \; in \; Y\}$

- A section graph is a clique if all the nodes in the subgraph are pairwise adjacent (→ dense block in matrix)

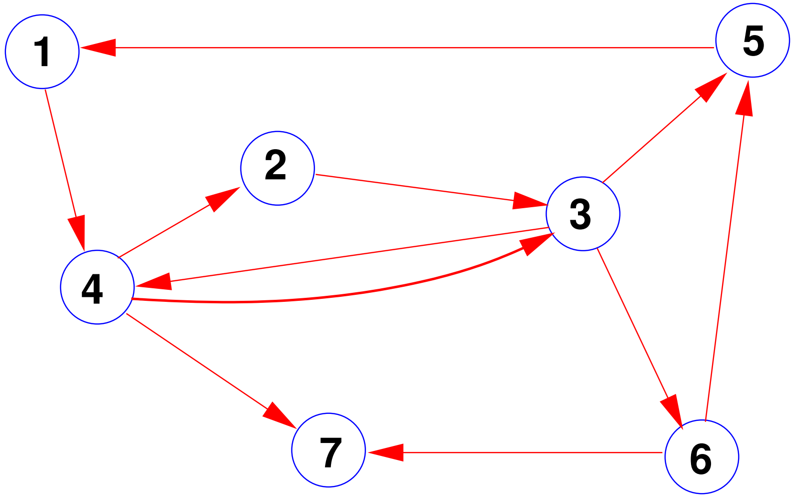
- A path is a sequence of vertices $w_0, w_1, \ldots, w_k$ such that $(w_i, w_{i+1}) \in E$ for $i = 0, \ldots, k-1$.

- The length of the path $w_0, w_1, \ldots, w_k$ is k (# of edges in the path)

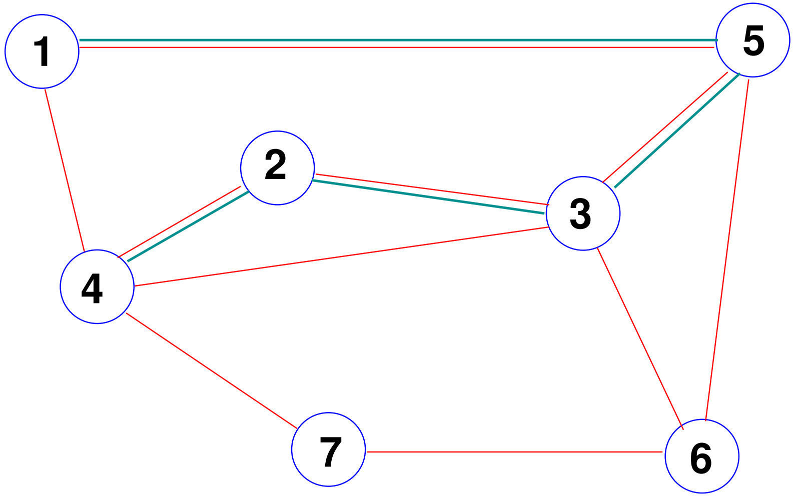
- A cycle is a closed path, i.e., a path with $w_k = w_0$.

- A graph is acyclic if it has no cycles.

✍ Find cycles in this graph:



A path in an indirected graph



- A path $w_0, \ldots, w_k$ is simple if the vertices $w_0, \ldots, w_k$ are distinct (except that we may have $w_0 = w_k$ for cycles).

- An undirected graph is connected if there is path from every vertex to every other vertex.

- A digraph with the same property is said to be strongly connected

- The undirected (or symmetrized) form of a digraph = undirected graph obtained by removing the directions of all edges

- A directed graph whose undirected form is connected is said to be weakly connected or connected.

- Tree = a graph whose undirected form, i.e., symmetrized form, is acyclic & connected – Forest = a collection of trees

GRAPH MODELS FOR SPARSE MATRICES

Graph Representations of Sparse Matrices. Recall:

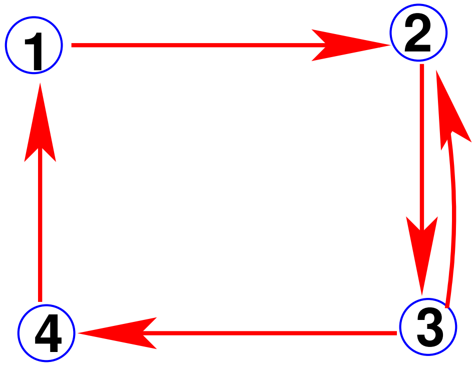
Adjacency Graph $G = (V, E)$ of an $n \times n$ matrix A :

$$V = \{1, 2, ..., N\} \qquad E = \{(i, j) | a_{ij} \neq 0\}$$

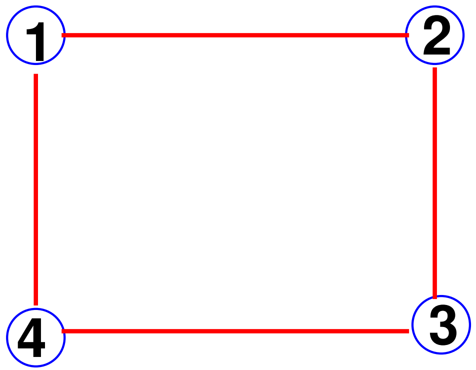
- G == undirected if A has a symmetric pattern

Example:

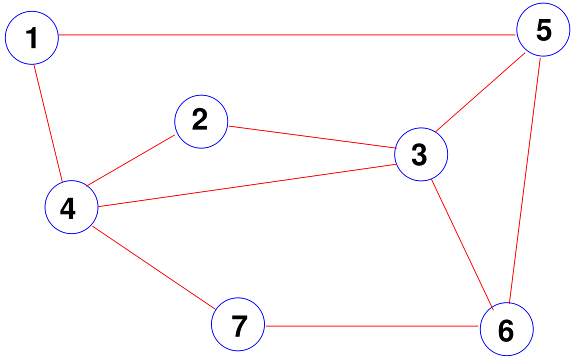
$$\begin{bmatrix} & \star & & \\ & & \star & \\ & \star & & \star \\ \star & & & \end{bmatrix}$$



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



✍ Show the matrix pattern for the graph on the right and give an interpretation of the path v_4, v_2, v_3, v_5, v_1 on the matrix



- A separator is a set Y of vertices such that the graph G_{X-Y} is disconnected.

Example: $Y = \{v_3, v_4, v_5\}$ is a separator in the above figure

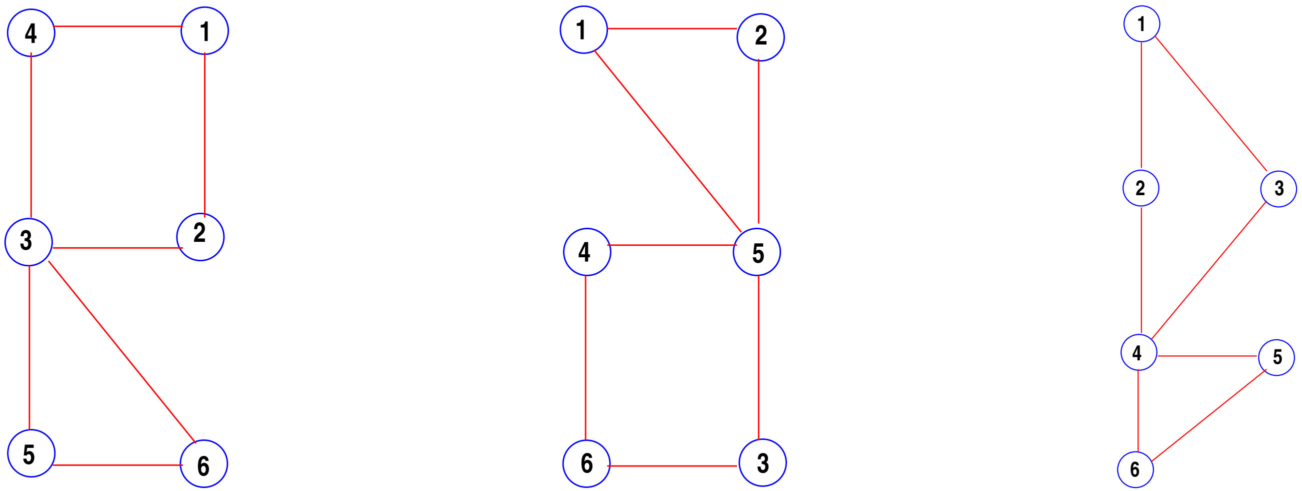
Example: Adjacency graph of:

$$A = \begin{bmatrix} & \star & & \star & & \\ \star & & \star & & & \\ & \star & & \star & \star & \star \\ \star & & \star & & & \\ & & \star & & & \star \\ & & \star & & \star & \end{bmatrix}.$$

Example: For any adjacency matrix A, what is the graph of A^2? [interpret in terms of paths in the graph of A]

- Two graphs are isomorphic is there is a mapping between the vertices of the two graphs that preserves adjacency.

✍ Are the following 3 graphs isomorphic? If yes find the mappings between them.



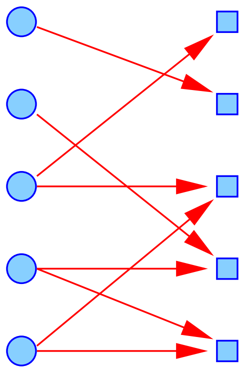
- Graphs are identical – labels are different
- Determinig graph isomorphism is a hard problem

Bipartite graph representation

- Rows and columns are (both) represented by vertices;
- Relations only between rows and columns: Row i is connected to column j if $a_{ij} \neq 0$

Example:

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



- Bipartite models used only for specific cases [e.g. rectangular matrices, ...] - By default we use the standard definition of graphs.

Interpretation of graphs of matrices

✍ What is the graph of $A + B$ (for two $n \times n$ matrices)?

✍ What is the graph of A^T ?

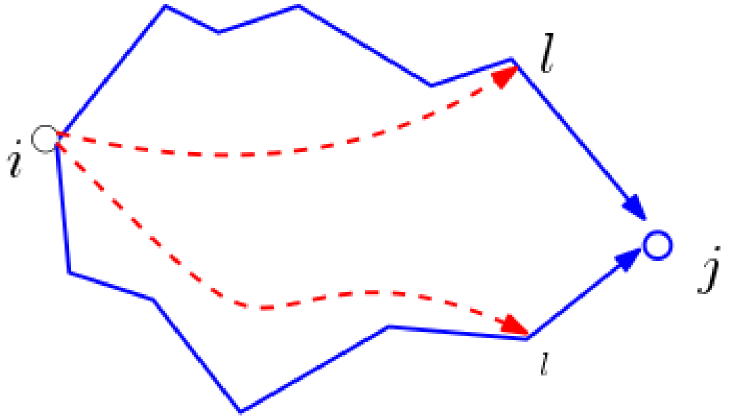
✍ What is the graph of $A.B$?

Paths in graphs

✍ What is the graph of A^k?

Theorem Let A be the adjacency matrix of a graph $G = (V, E)$. Then for $k \geq 0$ and vertices u and v of G, the number of paths of length k starting at u and ending at v is equal to $(A^k)_{u,v}$.

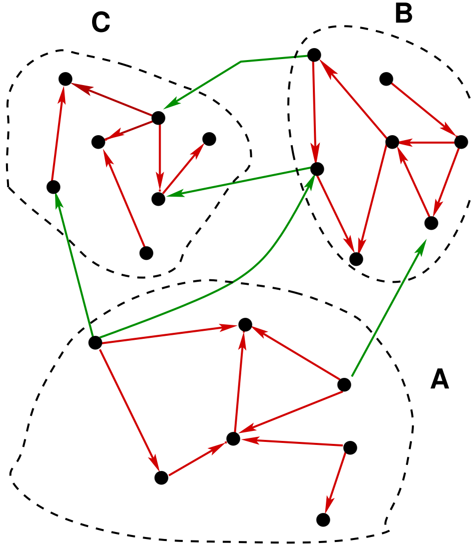
Proof: Proof is by induction.■

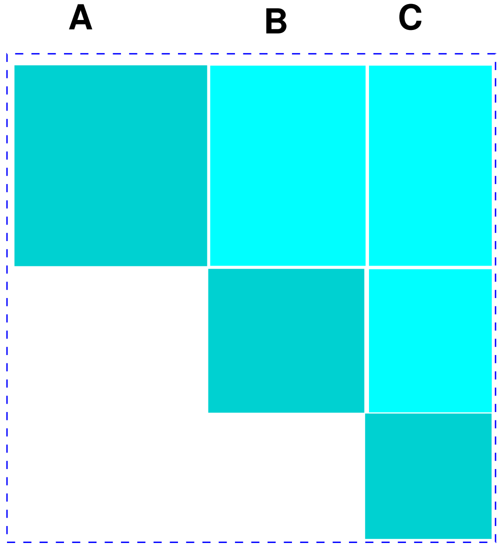


If $C = BA$ then $c_{ij} = \Sigma_l b_{il} a_{lj}$. Take $B = A^{k-1}$ and use induction. Any path of length k is formed as a path of length $k-1$ to some node l completed by an edge from l to j. Because a_{lj} is one for that last edge, c_{ij} is just the sum of all possible paths of length k from i to j

- Recall (definition): A matrix is *reducible* if it can be permuted into a block upper triangular matrix.

- Note: A matrix is reducible iff its adjacency graph is not (strongly) connected, i.e., iff it has more than one connected component.





- No edges from C to A or B. No edges from B to A.

Theorem: Perron-Frobenius An irreducible, nonnegative $n \times n$ matrix A has a real, positive eigenvalue λ_1 such that:

(i) λ_1 is a simple eigenvalue of A;

(ii) λ_1 admits a positive eigenvector u_1 ; and

(iii)$|\lambda_i| \leq \lambda_1$ for all other eigenvalues λ_i where $i > 1$.

- The spectral radius is equal to the eigenvalue λ_1

- Definition : a graph is d regular if each vertex has the same degree d.

Proposition: The spectral radius of a d regular graph is equal to d.

Proof: The vector e of all ones is an eigenvector of A associated with the eigenvalue $\lambda = d$. In addition this eigenvalue is the largest possible (consider the infinity norm of A). Therefore e is the Perron-Frobenius vector u_1. ■

Application: Markov Chains

- Read about Markov Chains in Sect. 10.9 of:

  https://www-users.cs.umn.edu/~saad/eig_book_2ndEd.pdf

- Let $\pi \equiv$ row vector of stationary probabilities
- Then π satisfies the equation → $\boxed{\pi P = \pi}$

- P is the probabilty transition matrix and it is 'stochastic':

A matrix P is said to be *stochastic* if :

(i) $p_{ij} \geq 0$ for all i, j

(ii) $\sum_{j=1}^{n} p_{ij} = 1$ for $i = 1, \cdots, n$

(iii) No column of P is a zero column.

- Spectral radius is ≤ 1

✍ Why?

- Assume P is irreducible. Then:

- Perron Frobenius → $\rho(P) = 1$ is an eigenvalue and associated eigenvector has positive entries.

- Probabilities are obtained by scaling π by its sum.

- Example: One of the 2 models used for page rank.

Example: A college Fraternity has 50 students at various stages of college (Freshman, Sophomore, Junior, Senior). There are 6 potential stages for the following year: Freshman, Sophomore, Junior, Senior, graduated, or left-without degree. Following table gives probability of transitions from one stage to next

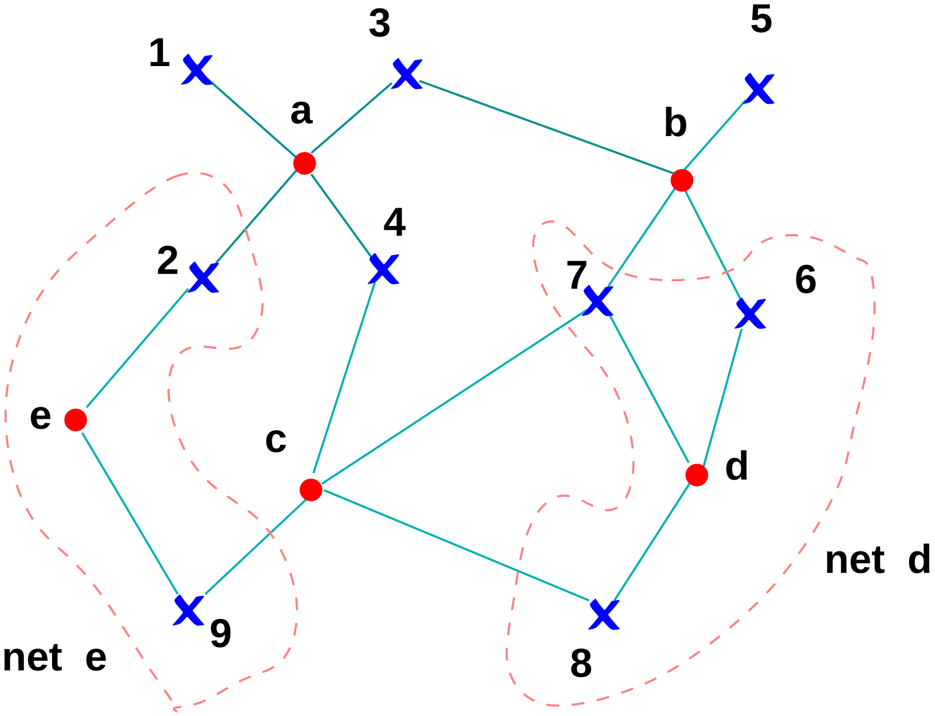
To From	Fr	So.	Ju.	Sr.	Grad	lwd
Fr.	.2	0	0	0	0	0
So.	.6	.1	0	0	0	0
Ju.	0	.7	.1	0	0	0
Sr.	0	0	.8	.1	0	0
Grad	0	0	0	.75	1	0
lwd	.2	.2	.1	.15	0	1

✍ What is P? Assume initial population is $x_0 = [10, 16, 12, 12, 0, 0]$ and do a follow the population for a few years. What is the probability that a student will graduate? What is the probability that s/he leaves without a degree?

A few words on hypergraphs

- Hypergraphs are very general.. Ideas borrowed from VLSI work
- Main motivation: to better represent communication volumes when partitioning a graph. Standard models face many limitations
- Hypergraphs can better express complex graph partitioning problems and provide better solutions.
- Example: completely nonsymmetric patterns ...
- .. Even rectangular matrices. Best illustration: Hypergraphs are ideal for text data

Example: $V = \{1, \ldots, 9\}$ and $E = \{a, \ldots, e\}$ with
$a = \{1,2,3,4\}$, $b = \{3,5,6,7\}$, $c = \{4,7,8,9\}$,
$d = \{6,7,8\}$, and $e = \{2,9\}$



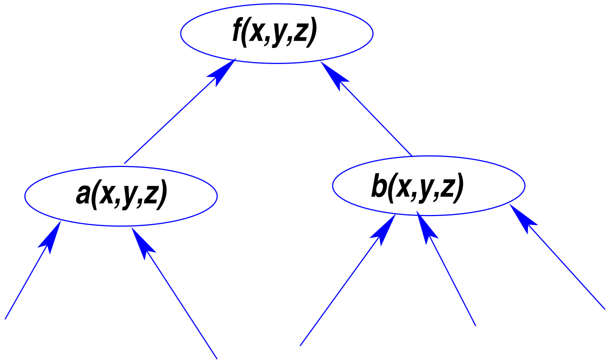
Boolean matrix:

$$A = \begin{array}{|ccccccccc|c} \multicolumn{1}{c}{1} & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \multicolumn{1}{c}{9} & \\ \hline 1 & 1 & 1 & 1 & & & & & & a \\ & & 1 & & 1 & 1 & 1 & & & b \\ & & & 1 & & & 1 & 1 & 1 & c \\ & & & & & 1 & 1 & 1 & & d \\ & 1 & & & & & & & 1 & e \\ \hline \end{array}$$

A few words on computational graphs

- Computational graphs: graphs where nodes represent computations whose evaluation depend on other (incoming) nodes.

f(x,y,z) = g(a(x,y,z), b(x,y,z))



- Consider the following expression:

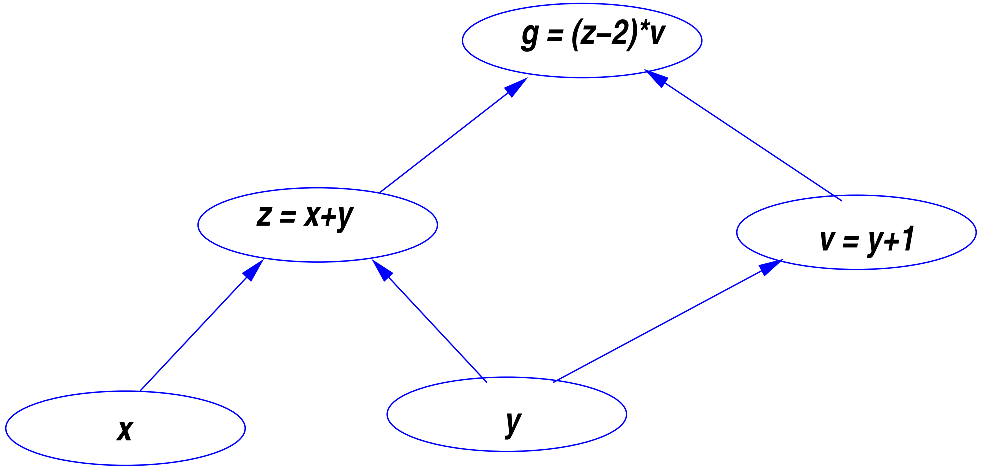
$$g(x, y) = (x + y - 2) * (y + 1)$$

We can decompose this as $\begin{cases} z &= x + y \\ v &= y + 1 \\ g &= (z - 2) * v \end{cases}$

- Computational graph →
- Given x, y we want:

(a) Evaluate the nodes and

(b) derivatives w.r.t x, y



(a) is trivial - just follow the graph up - starting from the leaves (that contain x and y)

(b): Use the chain rule – here shown for x only using previous setting

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial a}\frac{da}{dx} + \frac{\partial g}{\partial b}\frac{db}{dx}$$

✍ For the above example compute values and derivatives at all nodes when $x = -1, y = 2$.

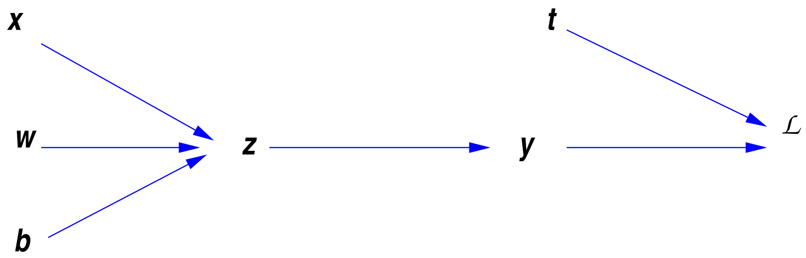
Back-Propagation

- Often we want to compute the gradient of the function at the root, once the nodes have been evaluated

- The derivatives can be calculated by going backward (or down the tree)

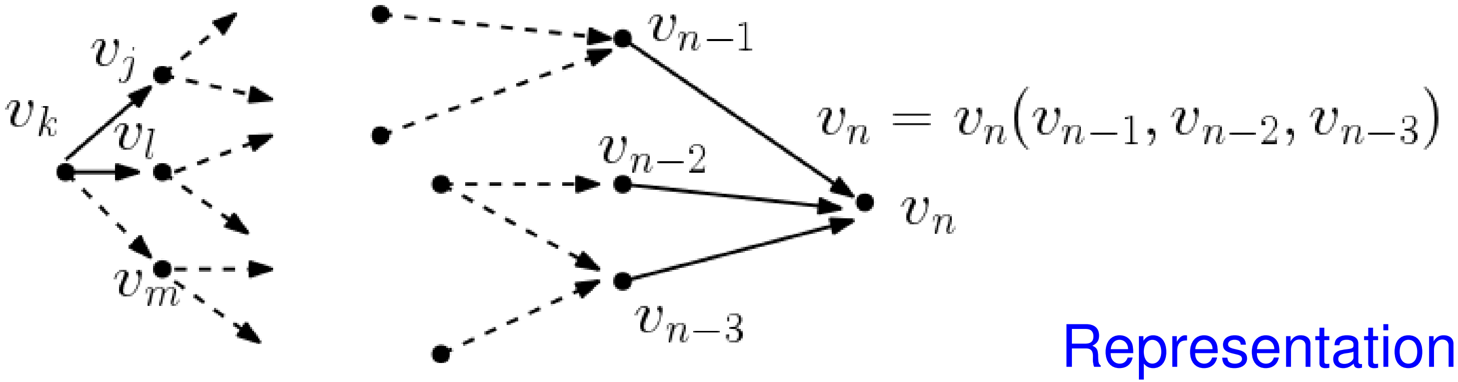
- Here is a very simple example from Neural Networks

$$\begin{cases} L = \frac{1}{2}(y-t)^2 \\ y = \sigma(z) \\ z = wx + b \end{cases}$$



- Note that t (desired output) and x (input) are constant.

Back-Propagation: General computational graphs



Representation: *a DAG*

- Last node (v_n) is the target function. Let us rename it f.
- Nodes $v_i, i = 1, \cdots, e$ with indegree 0 are the variables
- Want to compute $\partial f/\partial v_1, \partial f/\partial v_2, \cdots, \partial f/\partial v_e$
- Use the chain rule.

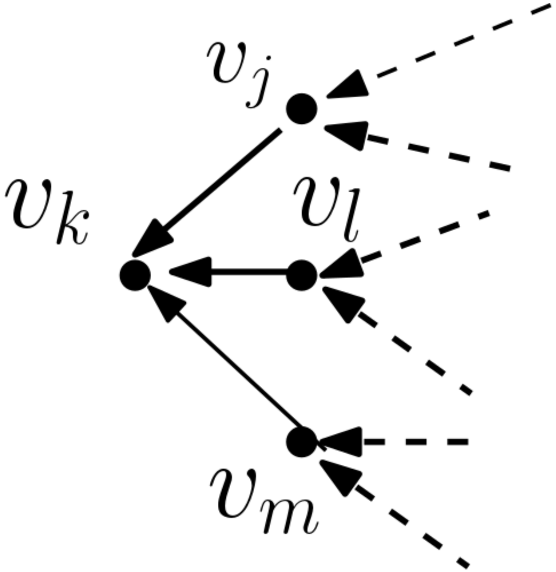
For $v_k(v_j, v_l, v_m) \longrightarrow$

$$\frac{\partial f}{\partial v_k} = \frac{\partial f}{\partial v_j}\frac{\partial v_j}{\partial v_k} + \frac{\partial f}{\partial v_l}\frac{\partial v_l}{\partial v_k} + \frac{\partial f}{\partial v_m}\frac{\partial v_m}{\partial v_k}$$

- Let $\delta_k = \frac{\partial f}{\partial v_k}$ (called 'errors'). Then

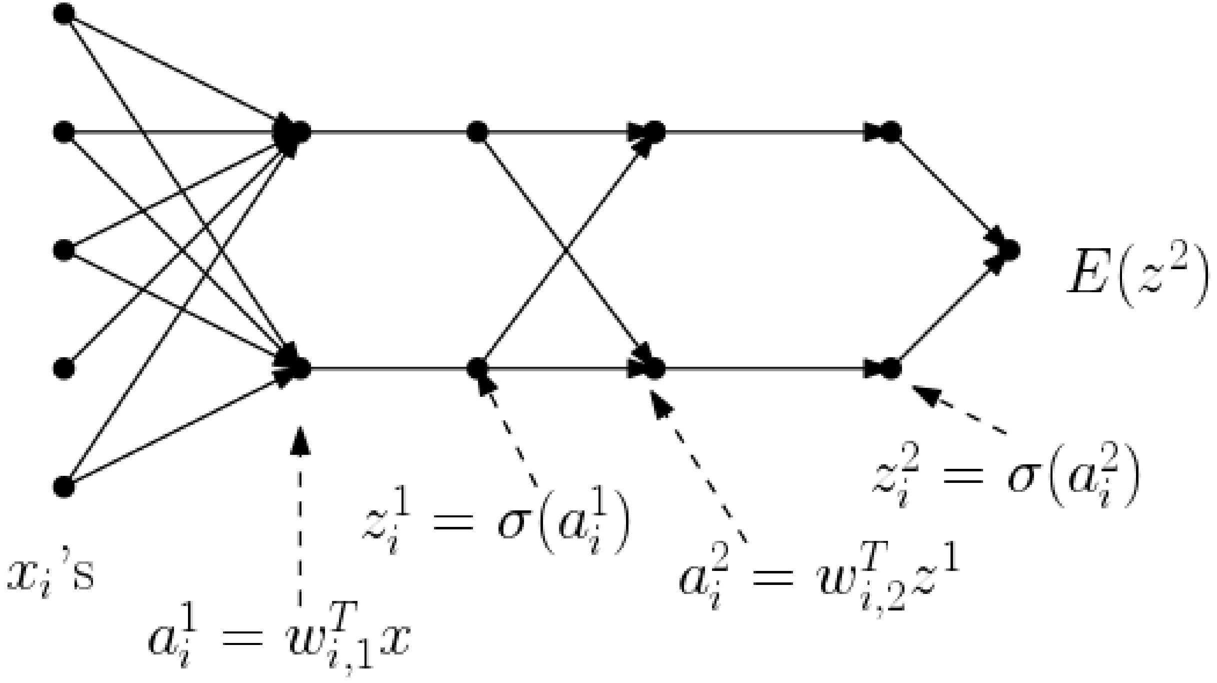
$$\boxed{\delta_k = \delta_j \frac{\partial v_j}{\partial v_k} + \delta_l \frac{\partial v_l}{\partial v_k} + \delta_m \frac{\partial v_m}{\partial v_k}}$$

- To compute the δ_k's once the v_j's have been computed (in a 'forward' propagation) – proceed backward.
- $\delta_j, \delta_l, \delta_m$ available and $\partial v_i / \partial v_k$ computable. Nore $\delta_n \equiv 1$.



- However: cannot just do this in any order. Must follow a topological order in order to obey dependencies.

Example:



GRAPH CENTRALITY

Centrality in graphs

- Goal: measure importance of a node, edge, subgraph, .. in a graph
- Many measures introduced over the years
- Early Work: Freeman '77 [introduced 3 measures] – based on 'paths in graph'
- Many different ways of defininf centrality! We will just see a few

Degree centrality: (simplest) ‘Nodes with high degree are important’

(note: scaling $n - 1$ is unimportant)

$$C_D(v) = \frac{\text{deg}(v)}{n-1}$$

Closeness centrality: ‘Nodes that are close to many other nodes are important’

$$C_C(v) = \frac{n-1}{\sum_{w \neq v} d(v,w)}$$

Betweenness centrality:

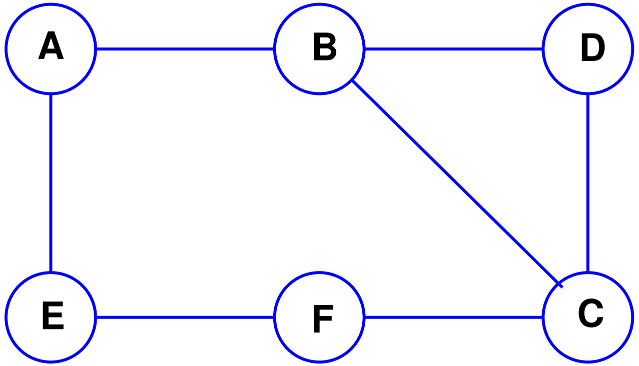
(Freeman ’77)

$$C_B(v) = \sum_{u \neq v, w \neq v} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

- σ_{uw} = total # shortest paths from u to w
- $\sigma_{uw}(v)$ = total # shortest paths from u to w passing through v

➤ ’Nodes that are on many shortest paths are important’

Example: Find $C_D(v)$; $C_C(v)$; $C_B(v)$ when $v = C$



(u,w)	$\sigma_{uw}(v)$	σ_{uw}	/	(u,w)	$\sigma_{uw}(v)$	σ_{uw}	/
(A,B)	0	1	0	(B,E)	0	1	0
(A,D)	0	1	0	(B,F)	1	1	1
(A,E)	0	1	0	(D,E)	1	2	.5
(A,F)	0	1	0	(D,F)	1	1	1
(B,D)	0	1	0	(E,F)	0	1	0

- $C_D(v) = 3/5 = 0.6$;
- $$C_C(v) = 5/[d_{CA} + d_{CB} + d_{CD} + d_{CE} + d_{CF}]$$
  $$= 5/[2 + 1 + 1 + 2 + 1] = 5/7$$
- $C_B(v) = 2.5$ (add all ratios in table)

✍ Redo this for $v = B$

Eigenvector centrality:

- Supppose we have n nodes v_j, $j = 1, \cdots, n$– each with a measure of importance ('prestige') p_j

- Principle: prestige of i depends on that of its neighbors.

- Prestige x_i = multiple of sum of prestiges of neighbors pointing to it

$$\boxed{\lambda x_i = \sum_{j \,\in \mathcal{N}(i)} x_j = \sum_{j=1}^{n} a_{ji} x_j}$$

- x_i = component of eigenvector associated with λ.

- Perron Frobenius theorem at play again: take largest eigenvalue → x_i's nonnegative

Page-rank

➤ Can be viewed as a variant of Eigenvector centrality

Main point: A page is important if it is pointed to by other important pages.

➤ Importance of your page (its PageRank) is determined by summing the page ranks of all pages which point to it. [→ same as EV centrality]

➤ Weighting: If a page points to several other pages, then the weighting should be distributed proportionally.

➤ Imagine many tokens doing a random walk on this graph:

- (δ/n) chance to follow one of the n links on a page,
- $(1-\delta)$ chance to jump to a random page.
- What's the chance a token will land on each page?

Page-Rank - definitions

If T_1, ..., T_n point to page T_i then

$$\boxed{\rho(T_i) \;=\; 1 - \delta + \delta \left[\frac{\rho(T_1)}{|T_1|} + \frac{\rho(T_2)}{|T_2|} + \cdots \frac{\rho(T_n)}{|T_n|} \right]}$$

- $|T_j|$ = count of links going out of Page T_i. So the 'vote' $\rho(T_j)$ is spread evenly among $|T_j|$ links.
- Sum of all PageRanks == 1: $\Sigma_T \rho(T) = 1$
- δ is a 'damping' parameter close to 1 – e.g. 0.85
- Defines a (possibly huge) Hyperlink matrix H

$$h_{ij} = \begin{cases} \frac{1}{|T_i|} & \text{if} \quad i \text{ points to } j \\ 0 & \text{otherwise} \end{cases}$$

✍ 4 Nodes

A points to B and D

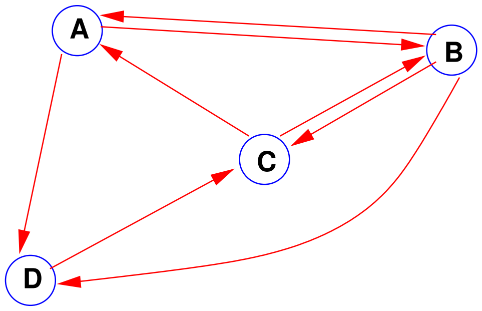
B points to A, C, and D

C points to A and B

D points to C

1) What is the H matrix?

2) the graph?



	A	B	C	D
A		$1/2$		$1/2$
B	$1/3$		$1/3$	$1/3$
C	$1/2$	$1/2$		
D			1	

- Row- sums of H are = 1.

- Sum of all PageRanks will be one:

$$\sum_{\text{All-Pages}_A} \rho(A) = 1.$$

- H is a stochastic matrix [actually it is forced to be by changing zero rows]

Algorithm (PageRank)

```
1.  Select initial row vector $v$ ($v \geq 0$)
2.  For i=1:maxitr
3       $v := (1-\delta)e^T + \delta v H$
4.  end
```

✍ Do a few steps of this algorithm for previous example with $\delta = 0.85$.

➤ This is a row iteration..

$$\boxed{\quad v \quad} = \boxed{(1-\delta)e^T} + \boxed{\quad v \quad} \cdot \boxed{\delta H}$$

A few properties:

- v will remain ≥ 0. [combines non-negative vectors]
- More general iteration is of the form

$$v := v[\underbrace{(1-\delta)E + \delta H}_{G}] \quad \textbf{with} \quad E = ez^T$$

where z is a probability vector $e^T z = 1$ [Ex. $z = \frac{1}{n}e$]

- A variant of the power method.
- e is a right-eigenvector of G associated with $\lambda = 1$. We are interested in the left eigenvector.

Kleinberg's Hubs and Authorities

➤ Idea is to put order into the web by ranking pages by their degree of Authority or ”Hubness”.

➤ An Authority is a page pointed to by many important pages.

- Authority Weight = sum of Hub Weights from In-Links.

➤ A Hub is a page that points to many important pages:

- Hub Weight = sum of Authority Weights from Out-Links.

➤ Source:

http://www.cs.cornell.edu/home/kleinber/auth.pdf

Computation of Hubs and Authorities

- Simplify computation by forcing sum of squares of weights to be 1.
- $\mathrm{Auth}_j = x_j = \sum_{i:(i,j)\in \mathrm{Edges}} \mathrm{Hub}_i$.
- $\mathrm{Hub}_i = y_i = \sum_{j:(i,j)\in \mathrm{Edges}} \mathrm{Auth}_j$.
- Let $A =$ Adjacency matrix: $a_{ij} = 1$ if $(i, j) \in \mathrm{Edges}$.
- $\mathrm{y} = A\mathrm{x}$, $\mathrm{x} = A^T\mathrm{y}$.
- Iterate . . . to leading eigenvectors of A^TA & AA^T.
- Answer: Leading Singular Vectors!

GRAPH LAPLACEANS AND THEIR APPLICATIONS

Graph Laplaceans - Definition

➤ “Laplace-type” matrices associated with general undirected graphs – useful in many applications

➤ Given a graph $G = (V, E)$ define

- A matrix W of weights w_{ij} for each edge
- Assume $w_{ij} \geq 0,$, $w_{ii} = 0$, and $w_{ij} = w_{ji} \; \forall (i, j)$
- The diagonal matrix $D = diag(d_i)$ with $d_i = \sum_{j \neq i} w_{ij}$

➤ Corresponding *graph Laplacean* of G is: $\boxed{L = D - W}$

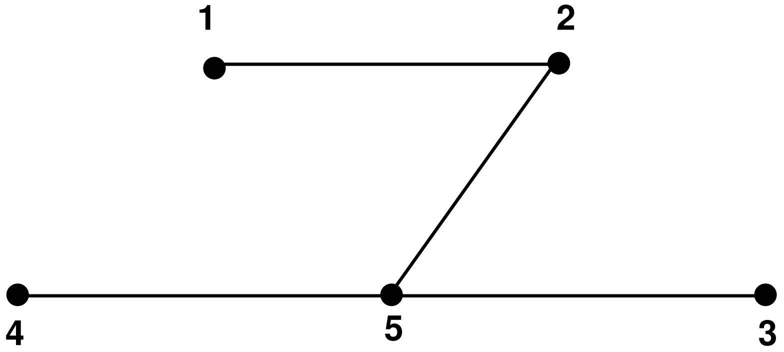
➤ Gershgorin's theorem $\rightarrow$ L is positive semidefinite.

- Simplest case:

$$w_{ij} = \begin{cases} 1 \text{ if } (i,j) \in E \& i \neq j \\ 0 \quad \text{else} \end{cases} \qquad D = \text{diag}\left[d_i = \sum_{j \neq i} w_{ij} \right]$$

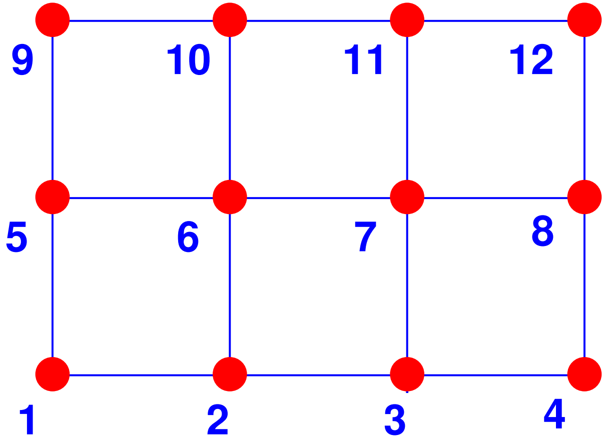
Example:

Consider the graph



$$L = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix}$$

✍ Define the graph Laplacean for the graph associated with the simple mesh shown next. [use the simple weights of 0 or 1]. What is the difference with the discretization of the Laplace operator for case when mesh is the same as this graph?



Proposition:

(i) L is symmetric semi-positive definite.

(ii) L is singular with $\mathbb{1}$ as a null vector.

(iii) If G is connected, then $\mathbf{Null}(L) = \mathbf{span}\{\mathbb{1}\}$

(iv) If G has $k > 1$ connected components $G_1, G_2, \cdots, G_k$, then the nullity of L is k and $\mathbf{Null}(L)$ is spanned by the vectors $z^{(j)}$, $j = 1, \cdots, k$ defined by:

$$(z^{(j)})_i = \begin{cases} 1 & \text{if } i \in G_j \\ 0 & \text{if not.} \end{cases}$$

Proof: (i) and (ii) seen earlier and are trivial. (iii) Clearly $u = \mathbb{1}$ is a null vector for L. The vector $D^{-1/2}u$ is an eigenvector for the matrix $D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2}$ associated with the smallest eigenvalue. It is also an eigenvector for $D^{-1/2}WD^{-1/2}$ associated with the largest eigenvalue. By the Perron Frobenius theorem this is a simple eigenvalue... (iv) Can be proved from the fact that L can be written as a direct sum of the Laplacian matrices for $G_1, \cdots, G_k$. ■

A few properties of graph Laplaceans

Define: oriented incidence matrix H: (1)First orient the edges $i \sim j$ into $i \rightarrow j$ or $j \rightarrow i$. (2) Rows of H indexed by vertices of G. Columns indexed by edges. (3) For each (i, j) in E, define the corresponding column in H as $\sqrt{w(i,j)}(e_i - e_j)$.

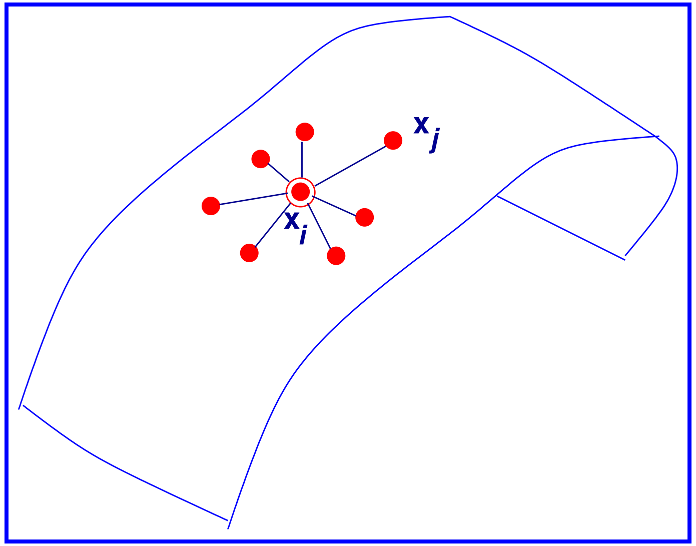
Example: In previous example (4 p. back) orient $i \rightarrow j$ so that $j > i$ [lower triangular matrix representation]. Then matrix H is:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -1 & -1 & -1 \end{bmatrix}$$

Property 1 $\boxed{L = HH^T}$

✍ Re-prove part (iv) of previous proposition by using this property.

A few properties of graph Laplaceans



Strong relation between $x^T L x$ and local distances between entries of x

- Let L = any matrix s.t. $L = D - W$, with $D = diag(d_i)$ and

$$w_{ij} \geq 0, \qquad d_i = \sum_{j \neq i} w_{ij}$$

Property 2: for any $x \in \mathbb{R}^n$:

$$x^\top L x = \frac{1}{2} \sum_{i,j} w_{ij} |x_i - x_j|^2$$

Property 3: (generalization) for any $\boldsymbol{Y} \in \mathbb{R}^{d\times n}$:

$$\text{Tr}\,[\boldsymbol{Y}\boldsymbol{L}\boldsymbol{Y}^\top] = \frac{1}{2}\sum_{i,j} w_{ij}\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2$$

➤ Note: $\boldsymbol{y}_j$ = j-th column of $\boldsymbol{Y}$. Usually $d < n$. Each column can represent a data sample.

Property 4: For the particular $\boldsymbol{L} = \boldsymbol{I} - \frac{1}{n}\mathbb{1}\,\mathbb{1}^\top$

$$\boldsymbol{X}\boldsymbol{L}\boldsymbol{X}^\top = \bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^\top == n \times \text{Covariance matrix}$$

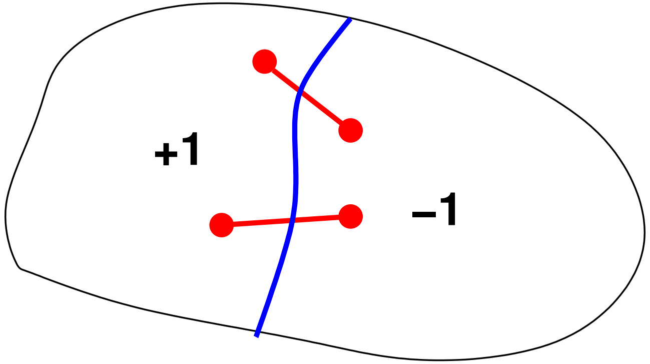
Property 5: $\boldsymbol{L}$ is singular and admits the null vector $\mathbb{1}$ =`ones(n,1)`

Property 6: (Graph partitioning) Consider situation when $w_{ij} \in \{0, 1\}$. If x is a vector of signs (± 1) then

$$x^\top L x = 4 \times (\text{'number of edge cuts'})$$

edge-cut = pair (i, j) with $x_i \neq x_j$

➤ Consequence: Can be used to partition graphs



➤ Would like to minimize (Lx, x) subject to $x \in \{-1, 1\}^n$ and $e^T x = 0$ [balanced sets]

➤ Wll solve a relaxed form of this problem

✍ What if we replace x by a vector of ones (representing one partition) and zeros (representing the other)?

✍ Let x be any vector and $y = x + \alpha \mathbb{1}$ and L a graph Laplacean. Compare (Lx, x) with (Ly, y).

- Consider any symmetric (real) matrix A with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ and eigenvectors $u_1, \cdots, u_n$

- Recall that:

(Min reached for $x = u_1$)

$$\boxed{\min_{x \in \mathbb{R}^n} \frac{(Ax, x)}{(x, x)} = \lambda_1}$$

- In addition:

(Min reached for $x = u_2$)

$$\boxed{\min_{x \perp u_1} \frac{(Ax, x)}{(x, x)} = \lambda_2}$$

- For a graph Laplacean $u_1 = \mathbb{1}$ = vector of all ones and

- ...vector u_2 is called the Fiedler vector. It solves a relaxed form of the problem -

$$\min_{x\in\{-1,1\}^n;\ \mathbb{1}^T x=0} \frac{(Lx,x)}{(x,x)} \quad \rightarrow \quad \min_{x\in\mathbb{R}^n;\ \mathbb{1}^T x=0} \frac{(Lx,x)}{(x,x)}$$

- Define $v = u_2$ then $lab = sign(v - med(v))$

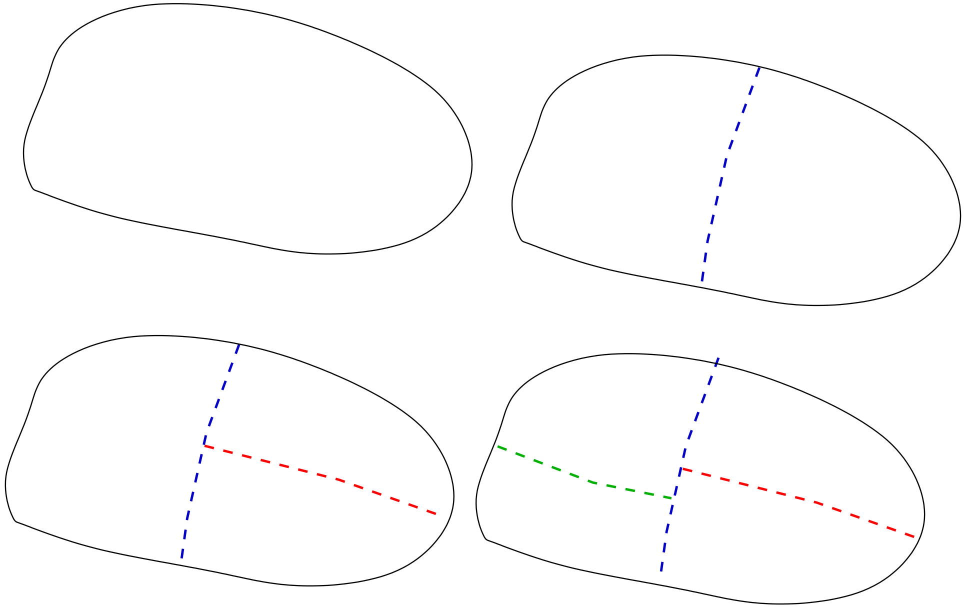
Recursive Spectral Bisection

1 Form graph Laplacean

2 Partition graph in 2 based on Fielder vector

3 Partition largest subgraph in two recursively ...

4 ... Until the desired number of partitions is reached



Three approaches to graph partitioning:

1. Spectral methods - Just seen + add Recursive Spectral Bisection.
2. Geometric techniques. Coordinates are required. [Houstis & Rice et al., Miller, Vavasis, Teng et al.]
3. Graph Theory techniques – multilevel,... [use graph, but no coordinates]
   - Currently best known technique is Metis (multi-level algorithm)
   - Simplest idea: Recursive Graph Bisection; Nested dissection (George & Liu, 1980; Liu 1992]
   - Advantages: simplicity – no coordinates required

Example of a graph theory approach

- Level Set Expansion Algorithm

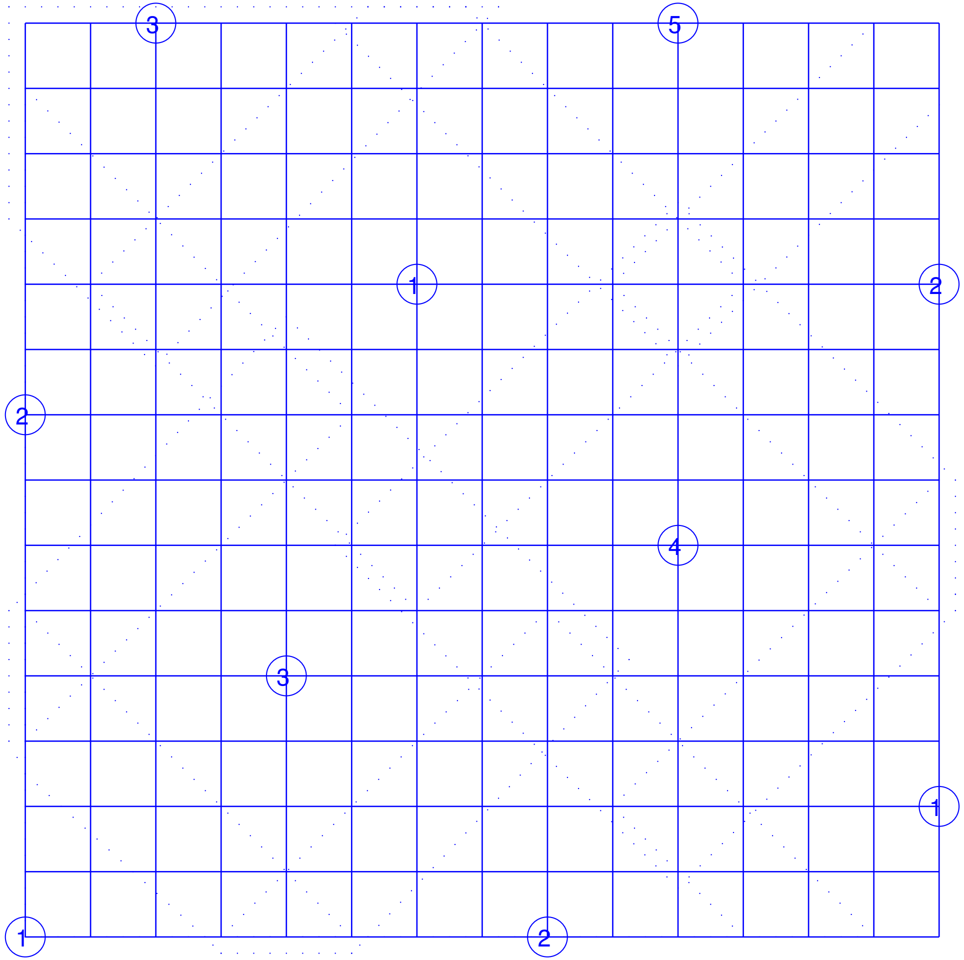
- Given: p nodes 'uniformly' spread in the graph (roughly same distance from one another).

- Method: Perform a level-set traversal (BFS) from each node simultaneously.

- Best described for an example on a 15×15 five – point Finite Difference grid.

- See [Goehring-YS '94, See Cai-YS '95]

- Approach also known under the name 'bubble' algorithm and implemented in some packages [Party, DibaP]

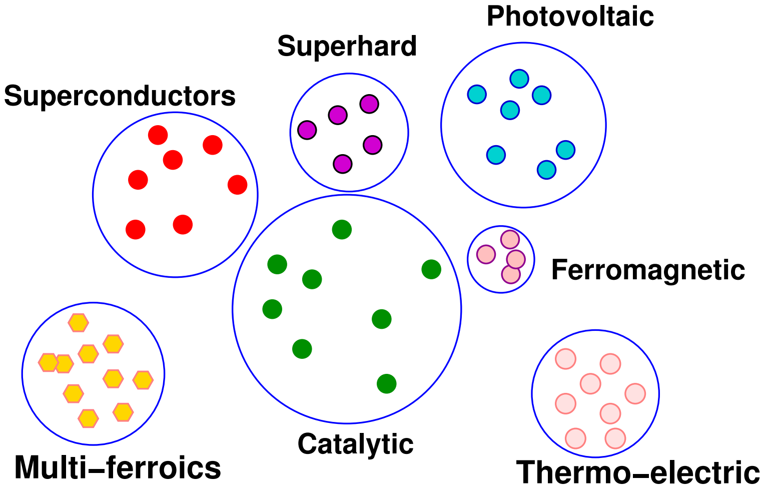

3
5
1
2
2
4
3
1
1
2

APPLICATIONS OF GRAPH LAPLACEANS: CLUSTERING

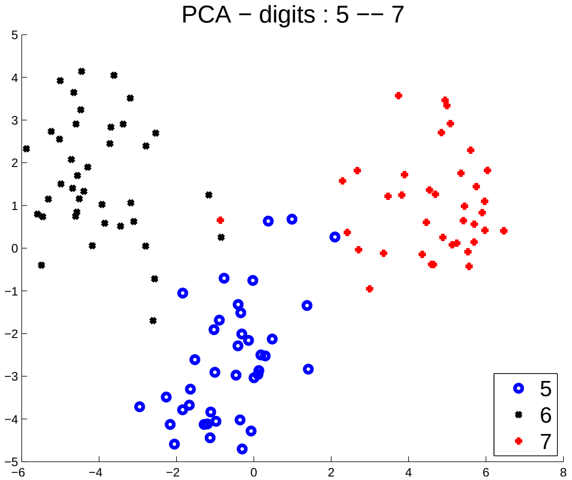
Clustering

- Problem: we are given n data items: $x_1, x_2, \cdots, x_n$. Would like to '*cluster*' them, i.e., group them so that each group or cluster contains items that are similar in some sense.

- Example: materials



- Example: Digits



- Each group is a 'cluster' or a 'class'

- 'Unsupervised learning'

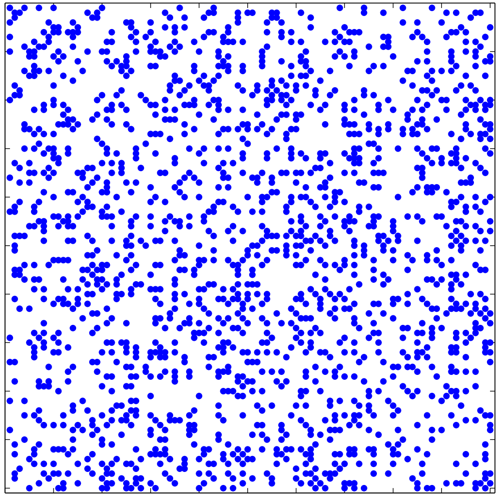
What is 'Unsupervised Learning'?

Ans: Class of methods that do not exploit labeled data

- Example of digits: perform a 2-D projection
- Images of same digit tend to cluster (more or less)
- Such 2-D representations are popular for visualization
- Can also try to find natural clusters in data, e.g., in materials
- Basic clusterning technique: K-means

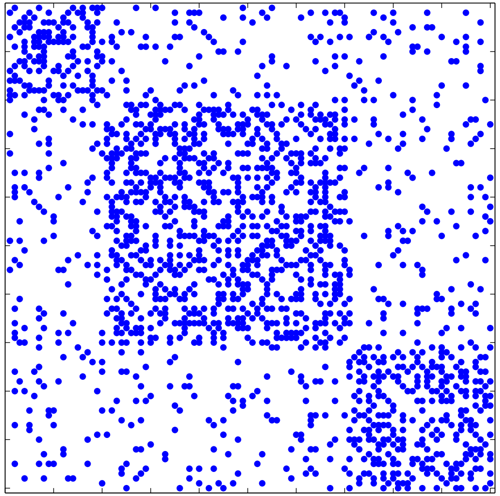
Example: Community Detection

- Communities modeled by an ‘affinity’ graph [e.g., ’user A sends frequent e-mails to user B’] . [data: www-personal.umich.edu/~mejn/netdata/]



← Original Adj. matrix

Goal: Find ordering so blocks are as dense as possible →



- Use ‘blocking’ techniques for sparse matrices
- Advantage of this viewpoint: need not know # of clusters.

Example of application

Data set from :

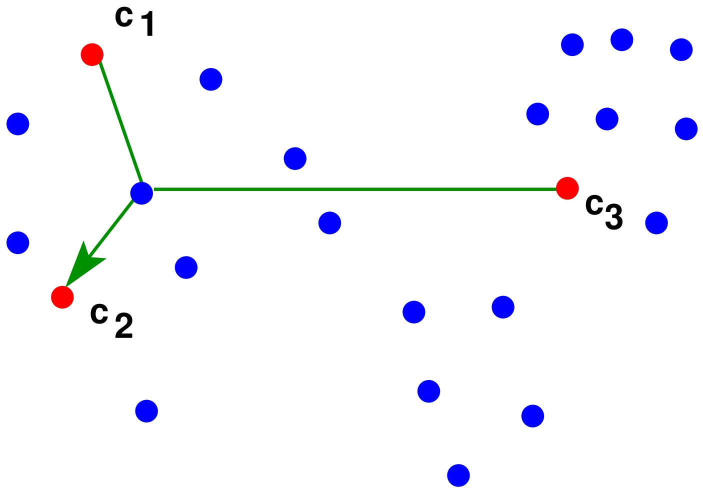
http://www-personal.umich.edu/~mejn/netdata/

- Network connecting bloggers of different political orientations [2004 US presidentual election]
- 'Communities': liberal vs. conservative
- Graph: $1,490$ vertices (blogs) : first 758: liberal, rest: conservative.
- Edge: $i \to j$: a citation between blogs i and j
- Blocking algorithm (Density theshold=0.4): subgraphs [note: density = $|E|/|V|^2$.]
- Smaller subgraph: conservative blogs, larger one: liberals

A basic method: K-means

- A basic algorithm that uses Euclidean distance

```
1  Select $p$ initial centers: $c_1, c_2, ..., c_p$ for classes $1, 2, \cdots, p$
2  For each $x_i$ do: determine *class* of $x_i$ as $\text{argmin}_k \|x_i - c_k\|$
3  Redefine each $c_k$ to be the centroid of class $k$
4  Repeat until convergence
```



- Simple algorithm
- Works well (gives good results) but can be slow
- Performance depends on initialization

Methods based on similarity graphs

➤ Class of Methods that perform clustering by exploiting a graph that describes the similarities between any two items in the data.

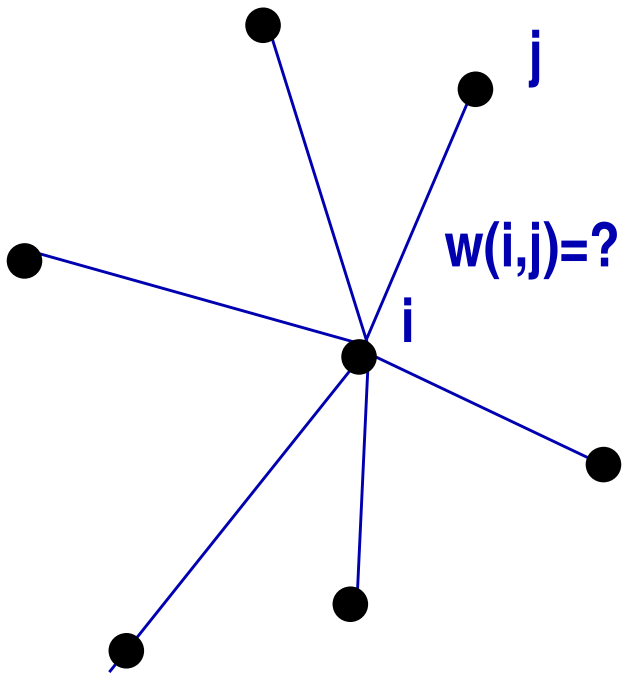
➤ Need to:

1. decide what nodes are in the neighborhood of a given node
2. quantify their similarities - by assigning a weight to any pair of nodes.

Example: For text data: Can decide that any columns i and j with a cosine greater than 0.95 are ‘similar’ and assign that cosine value to w_{ij}

First task: build a 'similarity' graph

- Goal: to build a similarity graph, i.e., a graph that captures similarity between any two items



- Two methods: K-nearest Neighbor graphs or use Gaussian ('heat') kernel

K-nearest neighbor graphs

- Given: a set of n data points $X = \{x_1, \ldots, x_n\} \rightarrow$ vertices

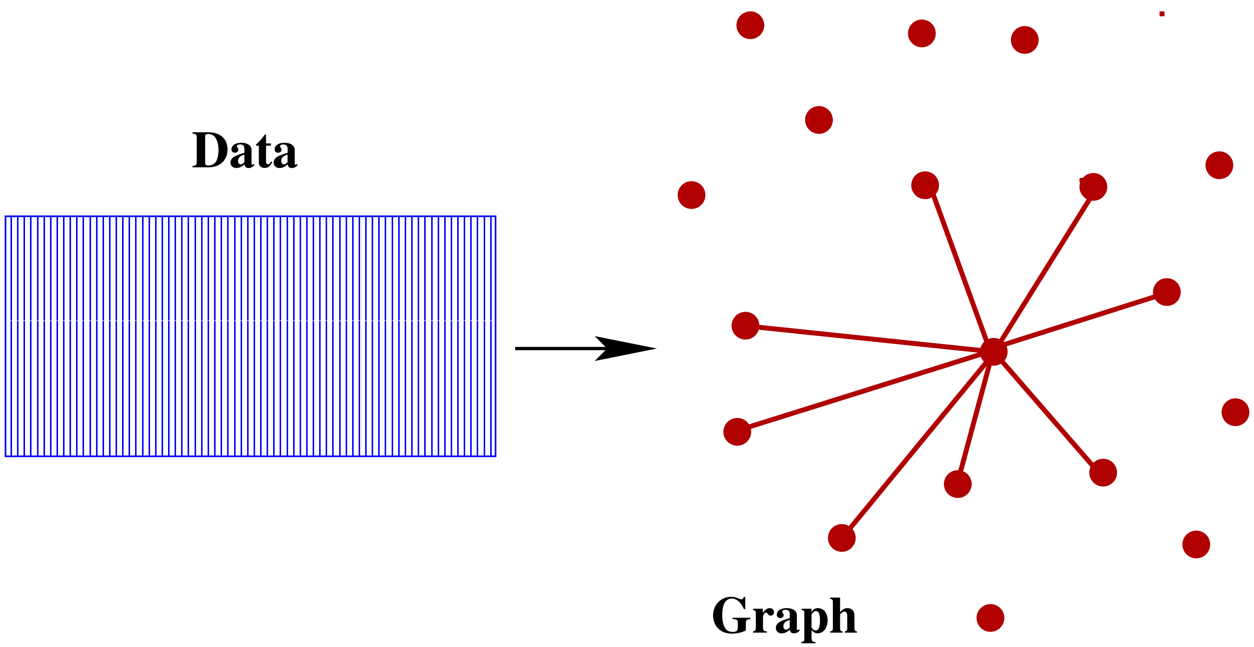
- Given: a proximity measure between two data points x_i and x_j – as measured by a quantity $dist(x_i, x_j)$

- Want: For each point x_i a list of the ‘nearest neighbors’ of x_i (edges between x_i and these nodes).

- Note: graph will usually be directed $\rightarrow$ need to symmetrize

Nearest neighbor graphs

- For each node, get a few of the nearest neighbors → Graph



- Problem: How to build a nearest-neighbor graph from given data
- We will revisit this later.

Two types of nearest neighbor graph often used:

ϵ-graph: Edges consist of pairs (x_i, x_j) such that $\rho(x_i, x_j) \le \epsilon$

kNN graph: Nodes adjacent to x_i are those nodes x_ℓ with the k with smallest distances $\rho(x_i, x_\ell)$.

- ϵ-graph is undirected and is geometrically motivated. Issues: 1) may result in disconnected components 2) what ϵ?
- kNN graphs are directed in general (can be trivially fixed).
- kNN graphs especially useful in practice.

Similarity graphs: Using ‘heat-kernels’

Define weight between i and j as:

$$w_{ij} = f_{ij} \ \times \begin{cases} e^{\frac{-\|x_i - x_j\|^2}{\sigma_X^2}} & \text{if } \|x_i - x_j\| < r \\ 0 & \text{if not} \end{cases}$$

- Note $\|x_i - x_j\|$ could be any measure of distance...
- f_{ij} = optional = some measure of similarity - other than distance
- Only nearby points kept.
- Sparsity depends on parameters

Edge cuts, ratio cuts, normalized cuts, ...

- Assume now that we have built a ‘similarity graph’
- Setting is identical with that of graph partitioning.
- Need a Graph Laplacean: $L = D - W$ with $w_{ii} = 0, w_{ij} \geq 0$ and $D = diag(W * ones(n, 1))$ [in matlab notation]
- Partition vertex set V in two sets A and B with

$$A \cup B = V, \quad A \cap B = \emptyset$$

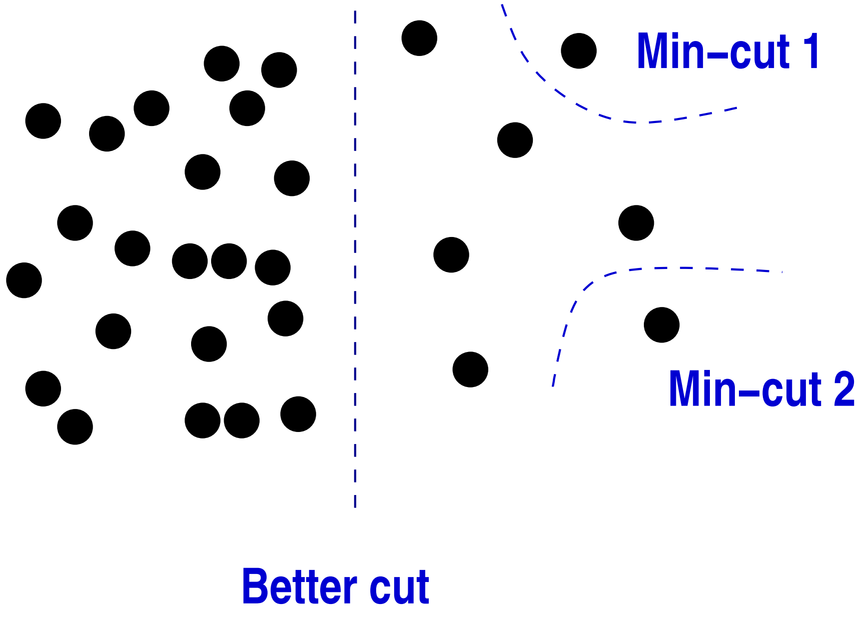
- Define

$$\boxed{cut(A, B) = \sum_{u\ \in A, v \in B} w(u, v)}$$

- First (naive) approach: use this measure to partition graph, i.e.,

... Find A and B that minimize $cut(A, B)$.

- Issue: Small sets, isolated nodes, big imbalances,



Ratio-cuts

- Standard Graph Partitioning approach: Find A, B by solving

$$\boxed{\text{Minimize} \quad cut(A, B), \text{ subject to } |A| = |B|}$$

- Condition $|A| = |B|$ not too meaningful in some applications - too restrictive in others.

- Minimum Ratio Cut approach. Find A, B by solving:

$$\boxed{\text{Minimize} \quad \frac{cut(A,B)}{|A|.|B|}}$$

- Difficult to find solution (original paper [Wei-Cheng '91] proposes several heuristics)

- Approximate solution : spectral .

Theorem [Hagen-Kahng, 91] If λ_2 is the 2nd smallest eigenvalue of L, then a lower bound for the cost c of the optimal ratio cut partition, is:

$$c \geq \frac{\lambda_2}{n}.$$

Proof: Consider an optimal partition A, B and let $p = |A|/n, q = |B|/n$. Note that $p + q = 1$. Let x be the vector with coordinates

$$x_i = \begin{cases} q & \text{if } i \in A \\ -p & \text{if } i \in B \end{cases}$$

Note that $x \perp \mathbb{1}$. Also if (i, j) == an edge-cut then $|x_i - x_j| = |q - (-p)| = |q + p| = 1$, otherwise $x_i - x_j = 0$. Therefore, $x^T L x = \sum_{(i,j)\in E}(x_i - x_j)^2 = w(A, B)$. In addition:

$\|x\|^2 = pq^2n + qp^2n = pq(p + q)n = pqn = \frac{|A|.|B|}{n}$.

Therefore, by the Courant-Fischer theorem:

$$\lambda_2 \leq \frac{(Lx, x)}{(x, x)} = n \times \frac{w(A, B)}{|A|.|B|} = n \times c.$$

Hence result. ■

➤ Idea is to use eigenvector associated with λ_2 to determine partition, e.g., based on sign of entries. Use the ratio-cut measure to actually determine where to split.

Normalized cuts [Shi-Malik,2000]

- Recall notation $w(X, Y) = \sum_{x\in X, y\in Y} w(x, y)$ - then define:

$$\boxed{\text{ncut}(A, B) = \frac{cut(A,B)}{w(A,V)} + \frac{cut(A,B)}{w(B,V)}}$$

- Goal is to avoid small sets A, B

✍ What is $w(A, V)$ in the case when $w_{ij} == 1$?

- Let x be an indicator vector:

$$x_i = \begin{cases} 1 \ if \ i \in A \\ 0 \ if \ i \in B \end{cases}$$

- Recall that: $x^T L x = \sum_{(i,j)\in E} w_{ij} |x_i - x_j|^2$ (note: each edge counted once)

- Therefore:

$$cut(A,B) = \sum_{x_i=1, x_j=0} w_{ij} = x^T L x$$

$$w(A,V) = \sum_{x_i=1} d_i = x^T W \mathbb{1} = x^T D \mathbb{1}$$

$$w(B,V) = \sum_{x_j=0} d_j = (\mathbb{1} - x)^T W \mathbb{1} = (\mathbb{1} - x)^T D \mathbb{1}$$

- Goal now: to minimize ncut

$$\min_{A,B} \text{ncut}(A,B) = \min_{x_i \in \{0,1\}} \frac{x^T L x}{x^T D x} + \frac{x^T L x}{(\mathbb{1} - x)^T D x}$$

- Let
$$\boxed{\begin{aligned}\beta = \frac{w(A,V)}{w(B,V)} &= \frac{x^T D \mathbb{1}}{(\mathbb{1} - x)^T D \mathbb{1}} \\ y = x - \beta(\mathbb{1} - x)\end{aligned}}$$

- Then we need to solve:
$$\boxed{\begin{array}{cc} \min\limits_{y_i\ \{0,-\beta\}} & \dfrac{y^T L y}{y^T D y} \\ \text{Subject to} & y^T D \mathbb{1} = 0 \end{array}}$$

- + Relax → need to solve Generalized eigenvalue problem

$$Ly = \lambda D y$$

- $y_1 = \mathbb{1}$ is eigenvector associated with eigenvalue $\lambda_1 = 0$

- y_2 associated with second eigenvalue solves problem.

A few properties

✍ Show that

$$\boldsymbol{ncut(A,B) = \sigma \times \frac{cut(A,B)}{w(A,V) \times w(B,V)}}$$

where $\boldsymbol{\sigma}$ is a constant

✍ How do ratio-cuts and normalized cuts compare when the graph is $\boldsymbol{d}$-regular (same degree for each node).

Extension to more than 2 clusters

- Just like graph partitioning we can:

1. Apply the method recursively [Repeat clustering on the resulted parts]
2. or compute a few eigenvectors and run K-means clustering on these eigenvectors to get the clustering.

Application: Image segmentation

- First task: obtain a graph from pixels.
- Common idea: use “Heat kernels”
- Let $F_j =$ feature value (e.g., brightness), and Let $X_j =$ spatial position.

Then define

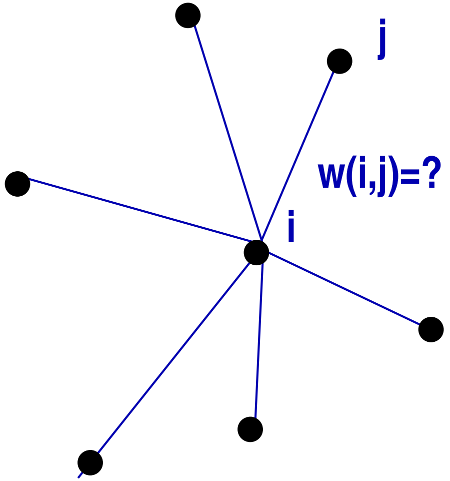
$$w_{ij} = e^{\frac{-\|F_i - F_j\|^2}{\sigma_I^2}} \times \begin{cases} e^{\frac{-\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if} \|X_i - X_j\| < r \\ 0 & \text{else} \end{cases}$$

- Sparsity depends on parameters

Spectral clustering: General approach

1 Given: Collection of data samples $\{x_1, x_2, \cdots, x_n\}$

2 Build a similarity graph between items

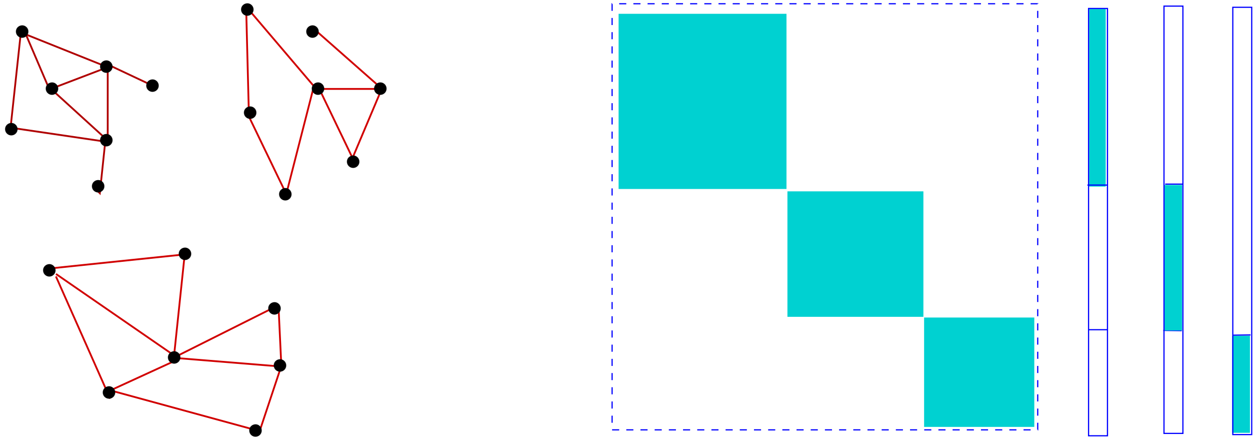


3 Compute (smallest) eigenvector (s) of resulting graph Laplacean

4 Use k-means on eigenvector (s) of Laplacean

➤ For Normalized cuts solve generalized eigen problem.

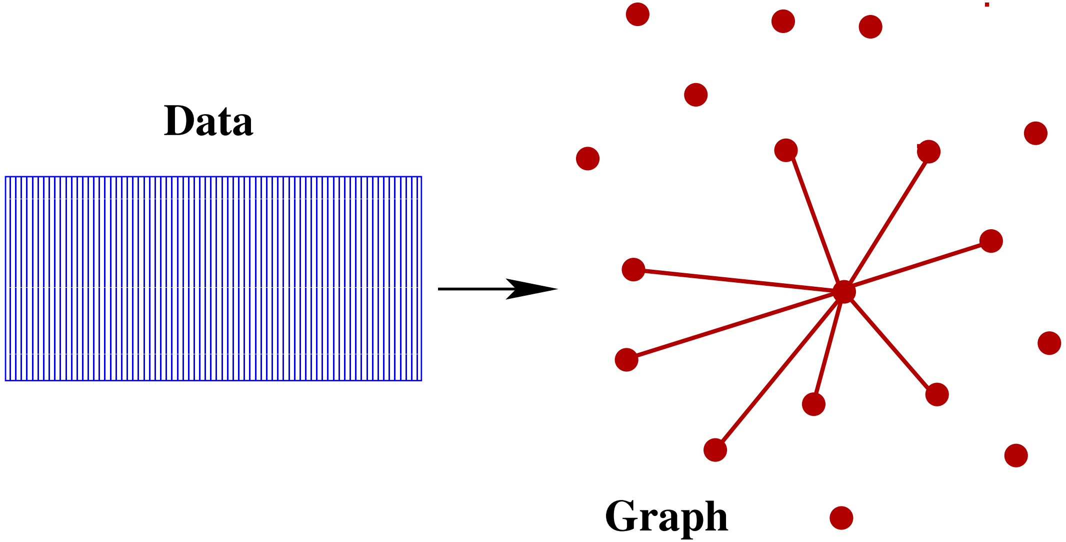
- Recall observation made earlier:



- Alg. Multiplicity of eigenvalue zero = # connected components.

Building a nearest neighbor graph

- Question: How to build a nearest-neighbor graph from given data?



- Will demonstrate the power of a divide a conquer approach combined with the Lanczos algorithm.

- Note: The Lanczos algortithm will be covered in detail later

Recall: Two common types of nearest neighbor graphs

ϵ-graph: Edges consist of pairs (x_i, x_j) such that $\rho(x_i, x_j) \leq \epsilon$

kNN graph: Nodes adjacent to x_i are those nodes x_ℓ with the k with smallest distances $\rho(x_i, x_\ell)$.

- ϵ-graph is undirected and is geometrically motivated. Issues: 1) may result in disconnected components 2) what ϵ?
- kNN graphs are directed in general (can be trivially fixed).
- kNN graphs especially useful in practice.

Divide and conquer KNN: key ingredient

- Key ingredient is *Spectral bisection*
- Let the data matrix $\boldsymbol{X = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}}$
- Each column == a data point.
- Center the data: $\boldsymbol{\hat{X} = [\hat{x}_1, \dots, \hat{x}_n] = X - ce^T}$

  where $\boldsymbol{c}$ == centroid; $\boldsymbol{e = ones(d, 1)}$ (matlab)

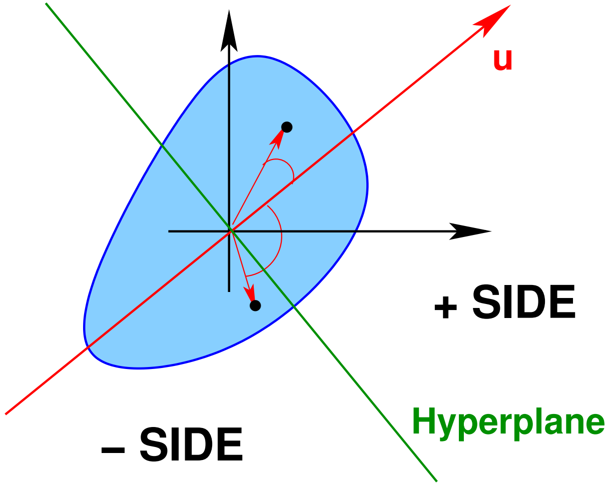
Goal: Split $\hat{X}$ into halves using a hyperplane.

Method: Principal Direction Divisive Partitioning D. Boley, '98.

Idea: Use the $(\boldsymbol{\sigma, u, v})$ = largest singular triplet of $\hat{X}$ with: $\boldsymbol{u^T \hat{X} = \sigma v^T}.$

- Hyperplane is defined as $\langle u, x\rangle = 0$, i.e., it splits the set of data points into two subsets:

$$\boldsymbol{X_+ = \{x_i \mid u^T\hat{x}_i \geq 0\}} \quad \text{and} \qquad \boldsymbol{X_- = \{x_i \mid u^T\hat{x}_i < 0\}}.$$



- Note that $\boldsymbol{u^T\hat{x}_i = u^T\hat{X}e_i = \sigma v^T e_i} \rightarrow$

$$\boldsymbol{X}_+ = \{\boldsymbol{x}_i \mid \boldsymbol{v}_i \geq 0\} \quad \text{and} \quad \boldsymbol{X}_- = \{\boldsymbol{x}_i \mid \boldsymbol{v}_i < 0\},$$

where v_i is the i-th entry of v.

- In practice: replace above criterion by

$$\boxed{\boldsymbol{X}_+ = \{\boldsymbol{x}_i \mid \boldsymbol{v}_i \geq \text{med}(\boldsymbol{v})\} \ \& \ \boldsymbol{X}_- = \{\boldsymbol{x}_i \mid \boldsymbol{v}_i < \text{med}(\boldsymbol{v})\}}$$

where *med*(v) == *median of the entries of* v.

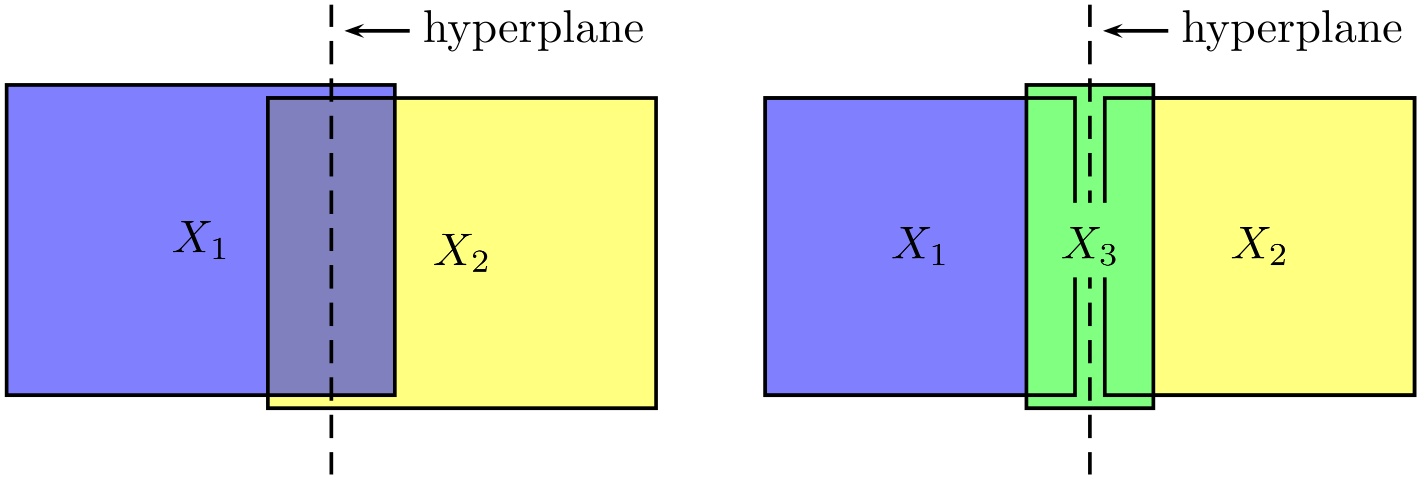
- For largest singular triplet (σ, u, v) of $\hat{X}$: use Golub-Kahan-Lanczos algorithm or Lanczos applied to $\hat{X}\hat{X}^T$ or $\hat{X}^T\hat{X}$

- Cost (assuming s Lanczos steps) : $O(n \times d \times s)$; Usually: d very small

Two divide and conquer algorithms

Overlap method: divide current set into two overlapping subsets $\boldsymbol{X_1, X_2}$

Glue method: divide current set into two disjoint subsets $\boldsymbol{X_1, X_2}$ plus a third set $\boldsymbol{X_3}$ called gluing set.



➤ Exploit recursivity

The Overlap Method

➤ Divide current set X into two overlapping subsets:

$$X_1 = \{x_i \mid v_i \geq -h_\alpha(S_v)\} \quad \text{and} \quad X_2 = \{x_i \mid v_i < h_\alpha(S_v)\},$$

- where $S_v = \{|v_i| \mid i = 1, 2, \ldots, n\}$.
- and $h_\alpha(\cdot)$ is a function that returns an element larger than $(100\alpha)\%$ of those in S_v.

➤ Rationale: to ensure that the two subsets overlap $(100\alpha)\%$ of the data, i.e.,

$$|X_1 \cap X_2| = \lceil \alpha |X| \rceil \,.$$

The Glue Method

Divide the set X into two disjoint subsets X_1 and X_2 with a gluing subset X_3:

$$X_1 \cup X_2 = X, \quad X_1 \cap X_2 = \emptyset, \quad X_1 \cap X_3 \neq \emptyset, \quad X_2 \cap X_3 \neq \emptyset.$$

Criterion used for splitting:

$$X_1 = \{x_i \mid v_i \geq 0\}, \quad X_2 = \{x_i \mid v_i < 0\},$$
$$X_3 = \{x_i \mid -h_\alpha(S_v) \leq v_i < h_\alpha(S_v)\}.$$

Note: gluing subset X_3 here is just the intersection of the sets X_1, X_2 of the overlap method.

Theorem The time complexity for the overlap method is

$$T_{\text{o}}(n) = \Theta(dn^{t_{\text{o}}}), \qquad \text{where:} \qquad t_{\text{o}} = \log_{2/(1+\alpha)} 2 = \frac{1}{1 - \log_2(1+\alpha)}.$$

Theorem The time complexity for the glue method is

$$T_{\text{g}}(n) = \Theta(dn^{t_{\text{g}}}/\alpha), \quad \text{where} \quad t_{\text{g}} \equiv \text{sol. to the equ.:} \quad \frac{2}{2^t} + \alpha^t = 1.$$

Example: When $\alpha = 0.1$, then $t_{\text{o}} = 1.16$ while $t_{\text{g}} = 1.12$.

Reference:

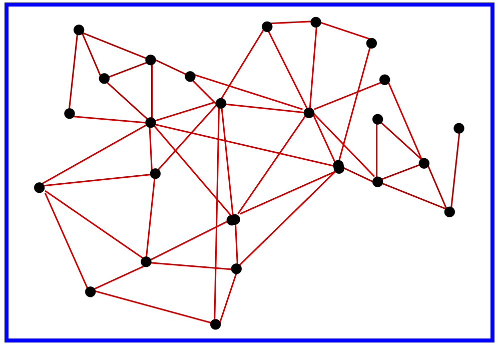
Jie Chen, Haw-Ren Fang and YS, "Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection" JMLR, vol. 10, pp. 1989-2012 (2009).

APPLICATIONS OF GRAPH LAPLACEANS: GRAPH EMBEDDINGS

Graph embeddings

- We have seen how to build a graph to represent data
- *Graph embedding* does the opposite: maps a graph to data

Given: a graph that models some data (e.g., a kNN graph)

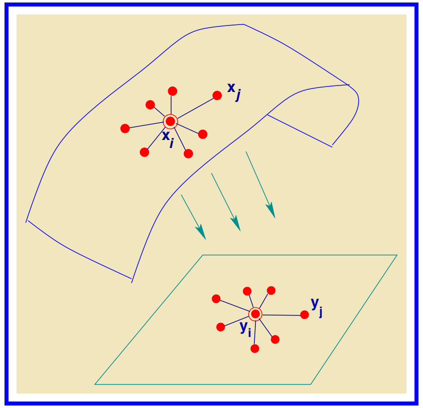


$\longrightarrow$ Data: $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_n]$ in $\mathbb{R}^d$

- Trivial use: visualize a graph ($\boldsymbol{d = 2}$)
- Wish: mapping should preserve *similarities* in graph.

Vertex embedding: map every vertex x_i to a vector $y_i \in \mathbb{R}^d$

- Many applications [clustering, finding missing link, semi-supervised learning, community detection, ...]



- Graph captures similarities, closeness, ..., in data

Objective: Build a mapping of each vertex i to a data point $y_i \in \mathbb{R}^d$

- Many methods do this

- Eigenmaps and LLE are two of the best known

- Eigenmaps uses the *graph Laplacean*

- Recall: Graph Laplacean is a matrix defined by :

$$\boxed{L = D - W}$$

$$\begin{cases} w_{ij} \geq 0 & \text{if } j \in Adj(i) \\ w_{ij} = 0 & \text{else} \end{cases} \qquad D = \operatorname{diag}\left[d_{ii} = \sum_{j \neq i} w_{ij} \right]$$

with $Adj(i)$ = neighborhood of i (excludes i)

- Remember that vertex i represents data item x_i. We will use i or x_i to refer to the vertex.

- We will find the y_i's by solving an optimization problem.

The Laplacean eigenmaps approach

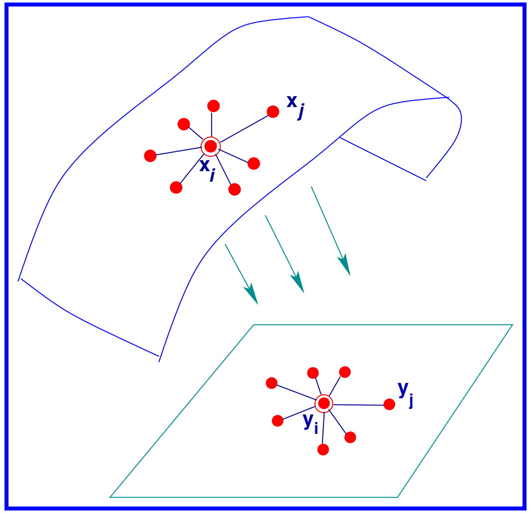
Laplacean Eigenmaps [Belkin-Niyogi '01] *minimizes*

$$\mathcal{F}(Y) = \sum_{i,j=1}^{n} w_{ij} \|y_i - y_j\|^2 \quad \text{subject to} \quad YDY^\top = I$$

Motivation: if $\|x_i - x_j\|$ is small (orig. data), we want $\|y_i - y_j\|$ to be also small (low-Dim. data)

➤ Original data used indirectly through its graph

➤ Objective function can be translated to a trace (see Property 3 in Lecture notes 9) and will yield a sparse eigenvalue problem



- Problem translates to:

$$\min_{\begin{cases} \boldsymbol{Y} \in \mathbb{R}^{d\times n} \\ \boldsymbol{Y}\boldsymbol{D}\,\boldsymbol{Y}^\top = \boldsymbol{I}\end{cases}} \quad \text{Tr}\left[\boldsymbol{Y}(\boldsymbol{D}-\boldsymbol{W})\boldsymbol{Y}^\top\right] .$$

- Solution (sort eigenvalues increasingly):

$$\boldsymbol{(D-W)u_i = \lambda_i D u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d}$$

- An $n \times n$ sparse eigenvalue problem [In 'sample' space]

- Note: can assume $\boldsymbol{D} = \boldsymbol{I}$. Amounts to rescaling data. Problem becomes

$$\boxed{\boldsymbol{(I-W)u_i = \lambda_i u_i \; ; \quad y_i = u_i^\top ; \quad i = 1, \cdots, d}}$$

Locally Linear Embedding (Roweis-Saul-00)

➤ LLE is very similar to Eigenmaps. Main differences:

1) Graph Laplacean matrix is replaced by an 'affinity' graph

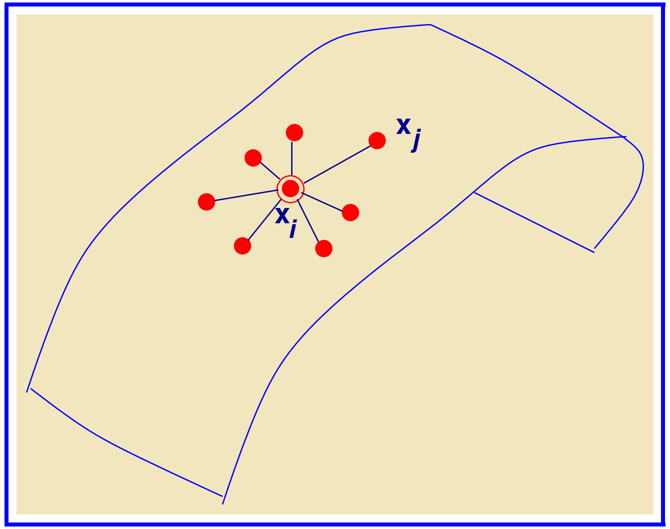
2) Objective function is changed: want to preserve graph

1. Graph: Each x_i is written as a convex combination of its k nearest neighbors:

$x_i \approx \Sigma w_{ij} x_j, \quad \sum_{j \in N_i} w_{ij} = 1$

➤ Optimal weights computed ('local calculation') by minimizing

$$\|x_i - \Sigma w_{ij} x_j\| \quad \text{for} \quad i = 1, \cdots, n$$



2. Mapping:

The y_i's should obey the same 'affinity' as x_i's $\rightsquigarrow$

Minimize:

$$\sum_i \left\| y_i - \sum_j w_{ij} y_j \right\|^2 \quad \text{subject to:} \quad Y \mathbb{1} = 0, \quad YY^\top = I$$

Solution:

$$\boxed{(I - W^\top)(I - W)u_i = \lambda_i u_i; \qquad y_i = u_i^\top \, .}$$

- $(I - W^\top)(I - W)$ replaces the graph Laplacean of eigenmaps

Implicit vs explicit mappings

- In Eigenmaps and LLE we only determine a set of $y_i's$ in $\mathbb{R}^d$ from the data points $\{x_i\}$.

- The mapping $\boxed{y_i = \phi(x_i), i = 1, \cdots , n}$ is implicit

- Difficult to compute a y for an x that is not one of the x_i's

- Inconvenient for classification. Thus is known as the "The out-of-sample extension" problem

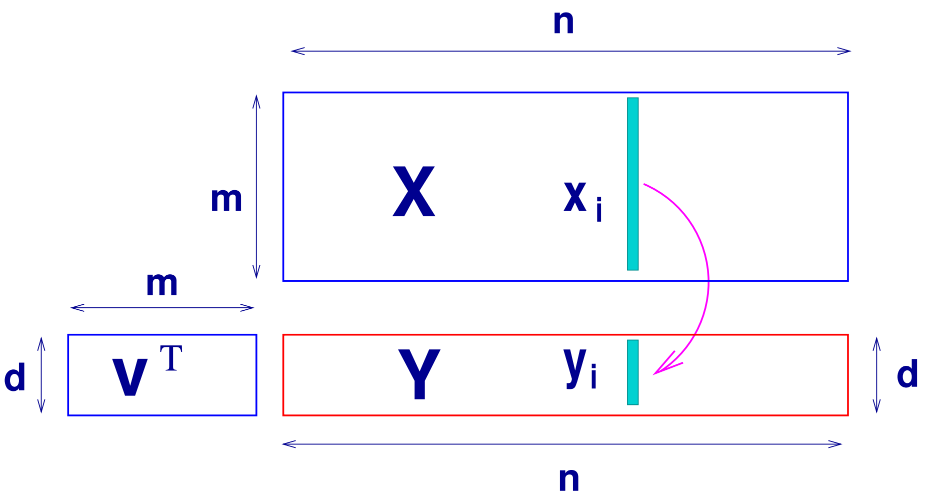
- In Explicit (also known as linear) methods: mapping ϕ is known explicitly (and it is linear.)

Locally Preserving Projections (He-Niyogi-03)

- LPP is a linear dimensionality reduction technique

- Recall the setting:

Want $V \in \mathbb{R}^{m \times d}$; $Y = V^\top X$



- Starts with the same neighborhood graph as Eigenmaps: $L \equiv D - W$ = graph 'Laplacean'; with $D \equiv diag(\{\Sigma_i w_{ij}\})$.

- Optimization problem is to solve

$$\min_{Y\in\mathbb{R}^{d\times n},\ YDY^\top=I} \quad \Sigma_{i,j} w_{ij} \left\|y_i - y_j\right\|^2, \quad Y = V^\top X.$$

- Difference with eigenmaps: Y is an explicit projection of X
- Solution (sort eigenvalues increasingly)

$$XLX^\top v_i = \lambda_i XDX^\top v_i \quad y_{i,:} = v_i^\top X$$

- Note: essentially same method in [Koren-Carmel'04] called 'weighted PCA' [viewed from the angle of improving PCA]

ONPP (Kokiopoulou and YS ’05)

- Orthogonal Neighborhood Preserving Projections

- A linear (orthogonoal) version of LLE obtained by writing $\boldsymbol{Y}$ in the form $\boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X}$

- Same graph as LLE. Objective: preserve the affinity graph (as in LLE) *but* with the constraint $\boldsymbol{Y} = \boldsymbol{V}^\top \boldsymbol{X}$

- Problem solved to obtain mapping:

$$\min_{\boldsymbol{V}} \text{Tr} \left[\boldsymbol{V}^\top \boldsymbol{X}(\boldsymbol{I} - \boldsymbol{W}^\top)(\boldsymbol{I} - \boldsymbol{W})\boldsymbol{X}^\top \boldsymbol{V}\right]$$

s.t. $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$

- In LLE replace $\boldsymbol{V}^\top \boldsymbol{X}$ by $\boldsymbol{Y}$

More recent methods

- Quite a bit of recent work - e.g., methods: node2vec, DeepWalk, GraRep, ... See the following papers ... among many others :

[1] *William L. Hamilton, Rex Ying, and Jure Leskovec* *Representation Learning on Graphs: Methods and Applications* arXiv:1709.05584v3

[2] *Shaosheng Cao, Wei Lu, and Qiongkai Xu* *GraRep: Learning Graph Representations with Global Structural Information*, CIKM, ACM Conference on Information and Knowledge Management, 24

[3] *Amr Ahmed, Nino Shervashidze, and Shravan Narayanamurthy*, *Distributed Large-scale Natural Graph Factorization* [Proc. WWW 2013, May 1317, 2013, Rio de Janeiro, Brazil]

Example: Graph factorization

- Line of work in Papers [1] and [3] above + others
- Instead of minimizing $\sum w_{ij} \|y_i - y_j\|_2^2$ as before

... try to minimize

$$\boxed{\sum_{ij} |w_{ij} - y_i^T y_j|^2}$$

- In other words solve: $\min_Y \|W - Y^T Y\|_F^2$
- Referred to as *Graph factorization*
- Common in knowledge graphs

Major tool of Data Mining: Dimension reduction

- Eigenmaps and LLE are a form of dimension reduction:

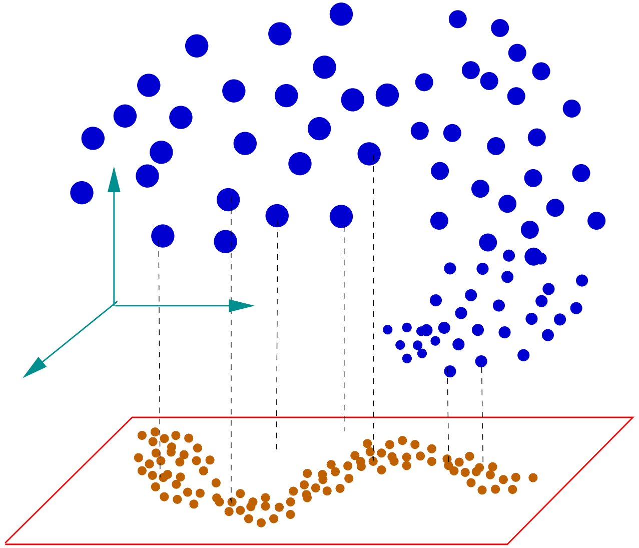
Data in $\mathbb{R}^m \rightarrow$ graph $\rightarrow$ Data in $\mathbb{R}^d$

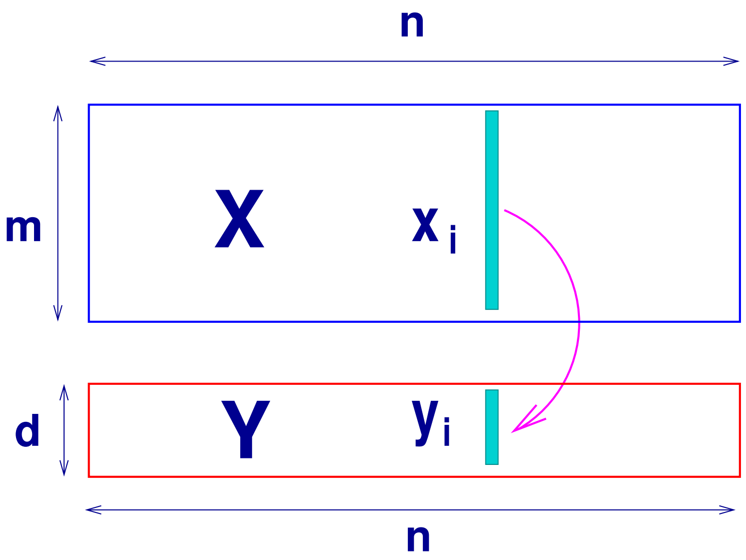
- So are the explicit (linear) methods (LPP, ONPP), ...,

Dimenson reduction: Given: $X = [x_1, \cdots, x_n] \in \mathbb{R}^{m\times n}$, find a low-dimens. representation $Y = [y_1, \cdots, y_n] \in \mathbb{R}^{d\times n}$ of X

- Achieved by a mapping $\Phi : x \in \mathbb{R}^m \longrightarrow y \in \mathbb{R}^d$ so:

$$\phi(x_i) = y_i, \quad i = 1, \cdots, n$$





- Φ may be linear : $y_j = W^\top x_j, \; \forall j$, or, $Y = W^\top X$

- ... or nonlinear (implicit).

- Mapping Φ required to: Preserve proximity? Maximize variance? Preserve a certain graph?

Basics: Principal Component Analysis (PCA)

In *Principal Component Analysis* $\boldsymbol{W}$ is computed to:

Maximize variance of projected data:

$$\max_{\boldsymbol{W}\in\mathbb{R}^{m\times d};\boldsymbol{W}^\top\boldsymbol{W}=\boldsymbol{I}} \quad \sum_{i=1}^{n}\left\|\boldsymbol{y}_i-\frac{1}{n}\sum_{j=1}^{n}\boldsymbol{y}_j\right\|_2^2, \quad \boldsymbol{y}_i=\boldsymbol{W}^\top\boldsymbol{x}_i.$$

➤ Leads to maximizing

$$\text{Tr}\left[\boldsymbol{W}^\top(\boldsymbol{X}-\boldsymbol{\mu}\boldsymbol{e}^\top)(\boldsymbol{X}-\boldsymbol{\mu}\boldsymbol{e}^\top)^\top\boldsymbol{W}\right], \quad \boldsymbol{\mu}=\tfrac{1}{n}\Sigma_{i=1}^{n}\boldsymbol{x}_i$$

➤ Solution $\boldsymbol{W}=\{$ dominant eigenvectors $\}$ of the covariance matrix $\equiv$ Set of left singular vectors of $\bar{\boldsymbol{X}}=\boldsymbol{X}-\boldsymbol{\mu}\boldsymbol{e}^\top$

SVD:

$$\bar{X} = U\Sigma V^\top, \quad U^\top U = I, \quad V^\top V = I, \quad \Sigma = \text{Diag}$$

- Optimal $W = U_d \equiv$ matrix of first d columns of U
- Solution W also minimizes ‘reconstruction error’ ..

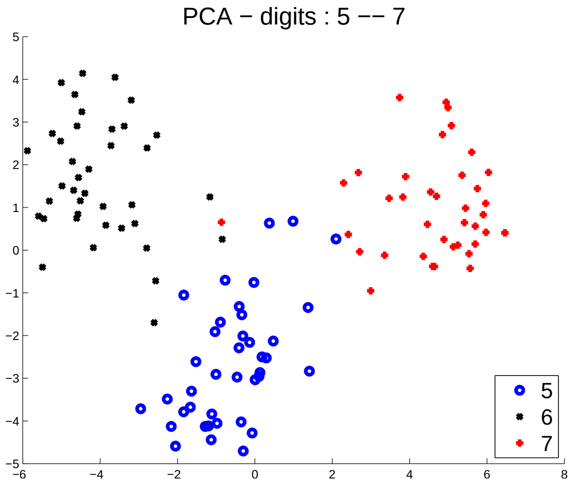
$$\sum_i \|x_i - WW^T x_i\|^2 = \sum_i \|x_i - Wy_i\|^2$$

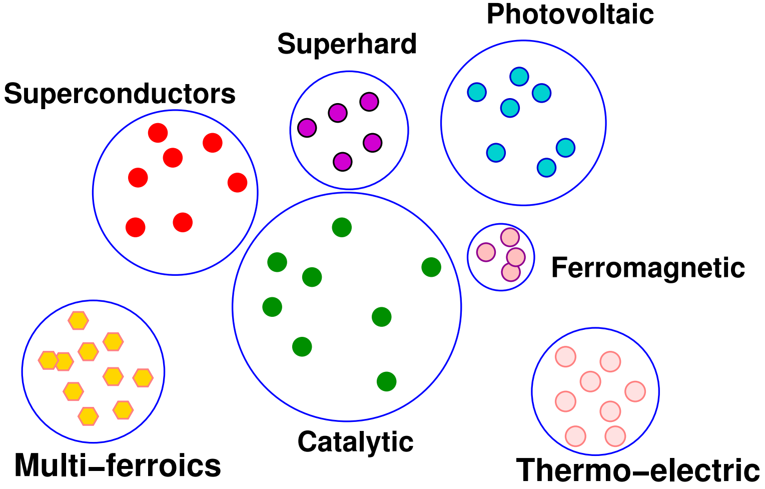
- In some methods recentering to zero is not done, i.e., $\bar{X}$ replaced by X.

Recall: Unsupervised learning

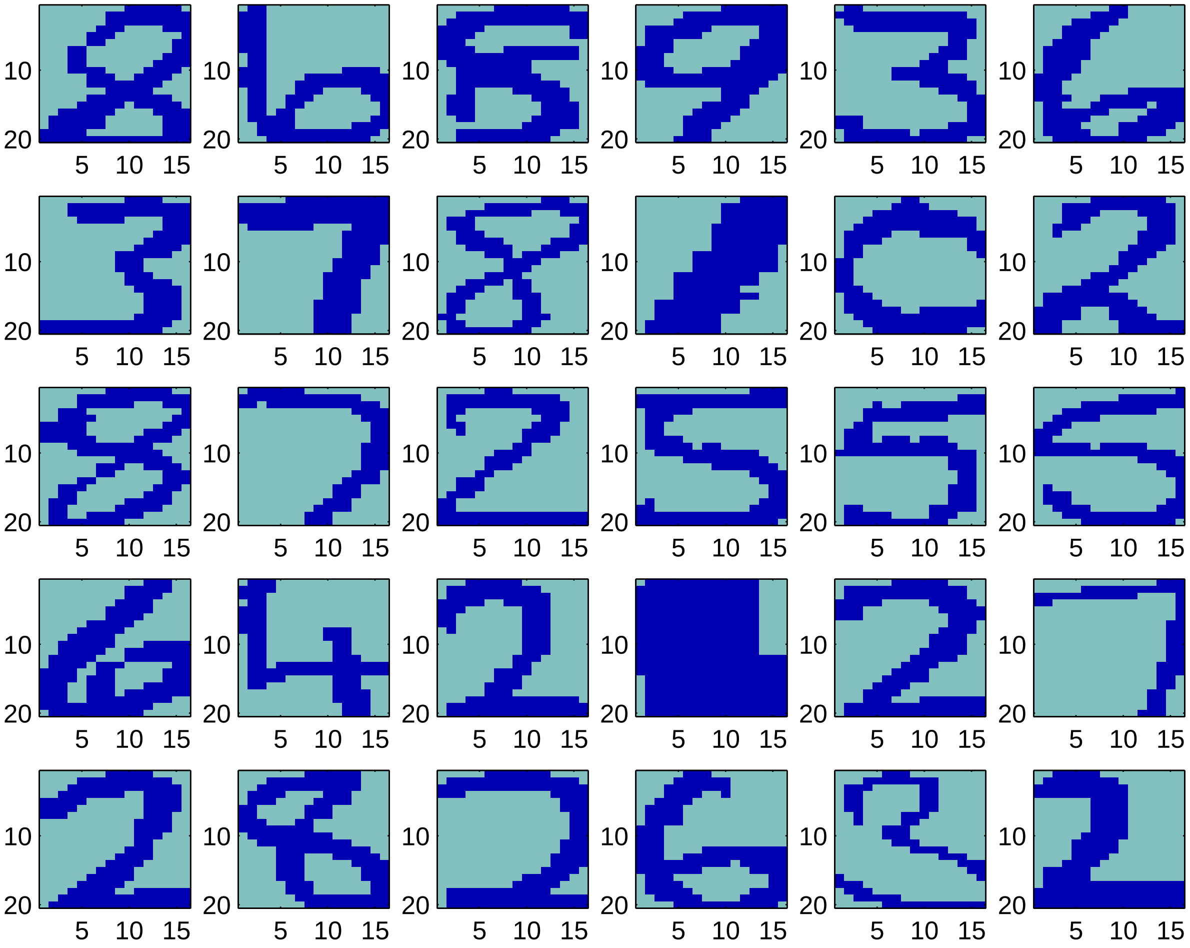
"Unsupervised learning" : methods do not exploit labeled data

- ➤ Example of digits: perform a 2-D projection
- ➤ Images of same digit tend to cluster (more or less)
- ➤ Such 2-D representations are popular for visualization
- ➤ Can also try to find natural clusters in data, e.g., in materials
- ➤ Basic clusterning technique: K-means

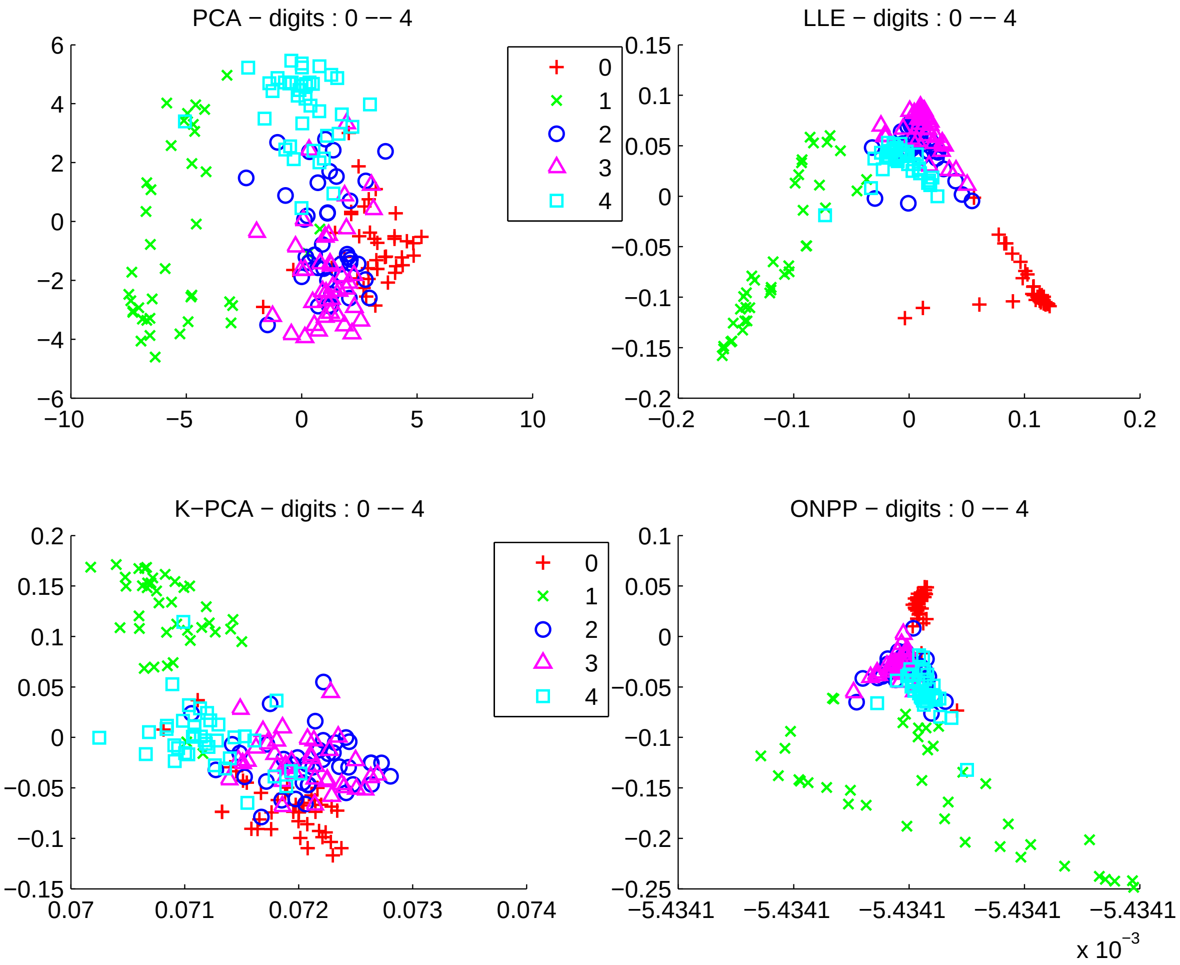




Example: Digit images (a random sample of 30)



2-D 'reductions':



SPECTRAL DENSITIES AND RANK ESTIMATION

What dimension to use in dimension reduction?

- Important question – but a hard one.
- Often, dimension k is selected in an ad-hoc way.
- k = intrinsic rank of data.
- Can we estimate it?

Two scenarios:

1. We know the magnitude of the noise, say τ. Then, ignore any singular value below τ and count the others.

2. We have no idea on the magnitude of noise. Determine a good threshold τ to use and count singular values $> \tau$.

Determining rank by eigenvalue counts

- Idea: count eigenvalues of A^TA (or AA^T) that are $> \tau$.

- Use technique in [E. Di Napoli, E. Polizzi, and Y.S., 2013] based on trace estimators.

- Summarized next for general situation of a symmetric real (or Hermitian complex) matrix A

Eigenvalue counts [E. Di Napoli, E. Polizzi, YS]

The problem:

Estimate number of eigenvalues of A in given interval $[a, \quad b]$

Motivation:

➤ Eigensolvers based on splitting the spectrum intervals and extracting eigenpairs from each interval independently.

- Contour integration-type methods, e,g., FEAST [Polizzi 2011], Sakurai-Sugiura - method [2003, 2007, ..]
- Polynomial filtering, e.g.,: Schofield, Chelikowsky, YS'2011.

➤ Problems related to rank estimation in many applications

Eigenvalue counts: Standard approach and an alternative

Problem: $\boldsymbol{A}$ Hermitian with eigenpairs $\boldsymbol{\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n}$. *Want:* number $\boldsymbol{\mu_{[a,b]}}$ of $\boldsymbol{\lambda_i}$'s $\in$ $[\boldsymbol{a}, \ \boldsymbol{b}]$ - where $\boldsymbol{\lambda_1 \le a \le b \le \lambda_n}$.

- Standard method: Use Sylvester inertia theorem. → Expensive

- Alternative: Exploit trace of the eigen-projector:

$$\boldsymbol{P} = \sum_{\lambda_i \, \in \, [a \ b]} \boldsymbol{u_i u_i^T}.$$

- We know that :

$$\text{Tr}\,(\boldsymbol{P}) = \boldsymbol{\mu_{[a,b]}}$$

- Goal: calculate an approximation to : Tr $(\boldsymbol{P})$

- P is not available ... but can be approximated by: ● (1) a polynomial in A, or ● (2) a rational function in A.

Approximation theory viewpoint:

- Interpret P as a step function of A, namely:

$$P = h(A) \quad \text{where} \quad h(t) = \begin{cases} 1 & \text{if } t \in [a\ b] \\ 0 & \text{otherwise} \end{cases}$$

- Approximate $h(t)$ by a polynomial ψ. Then use statistical estimator for approximating Tr $(\psi(A))$ – to be discussed next

- Hutchinson's unbiased estimator uses only matrix-vector products to approximate the trace of a generic matrix A.

How to estimate the trace of a matrix

Trace estimation: A few examples of applications

Problem 1: Compute Tr[inv[A]] the trace of the inverse.

➤ Arises in cross validation methods [Stats]. Motivation for the work [Golub & Meurant, “Matrices, Moments, and Quadrature”, 1993, Book with same title in 2009]

Problem 2: Compute Tr [f (A)], f a certain function

➤ Arises in many applications in Physics, e.g., Stochastic estimations of Tr (f(A)) extensively used by quantum chemists to estimate Density of States, see

[H. Röder, R. N. Silver, D. A. Drabold, J. J. Dong, Phys. Rev. B. 55, 15382 (1997)].

Problem 3: Compute diag[inv(A)] the diagonal of the inverse

- Dynamic Mean Field Theory [DMFT, motivation for our work on this topic]. Related approach: Non Equilibrium Green's Function (NEGF) approach used to model nanoscale transistors.

- Uncertainty quantification: diagonal of the inverse of a covariance matrix needed [Bekas, Curioni, Fedulova '09]

Problem 4: Compute diag[f (A)] ; f = a certain function.

- Arises in density matrix approaches in quantum modeling

$$f(\epsilon) = \frac{1}{1 + \exp(\frac{\epsilon-\mu}{k_B T})}$$

Here, f = Fermi-Dirac operator

Note: when $T \to 0$ then $f \to$ a step function.

- Linear-Scaling methods

Problem 5: Estimate the numerical rank.

➤ Amounts to counting the number of singular values above a certain threshold τ == Trace $(\phi_\tau(A^TA))$..

$\phi_\tau(t)$ is a certain step function.

Problem 6: Estimate the log-determinant (common in statistics)

$$\log\det(A) = \text{Trace}(\log(A)) = \sum_{i=1}^n \log(\lambda_i).$$

.... many others

Important tool: Stochastic Trace Estimator

➤ To estimate diagonal of $B = f(A)$ (e.g., $B = A^{-1}$), let:

Notation:

- $d(B) = \text{diag}(B)$ [matlab notation]
- $\odot$ and $\oslash$: Elementwise multiplication and division of vectors
- $\{v_j\}$: Sequence of s random vectors

Result:

$$d(B) \approx \left[\sum_{j=1}^{s} v_j \odot B v_j\right] \oslash \left[\sum_{j=1}^{s} v_j \odot v_j\right]$$

C. Bekas , E. Kokiopoulou & YS ('05); C. Bekas, A. Curioni, I. Fedulova '09; ...

Trace of a matrix

- For the trace - take vectors of unit norm and

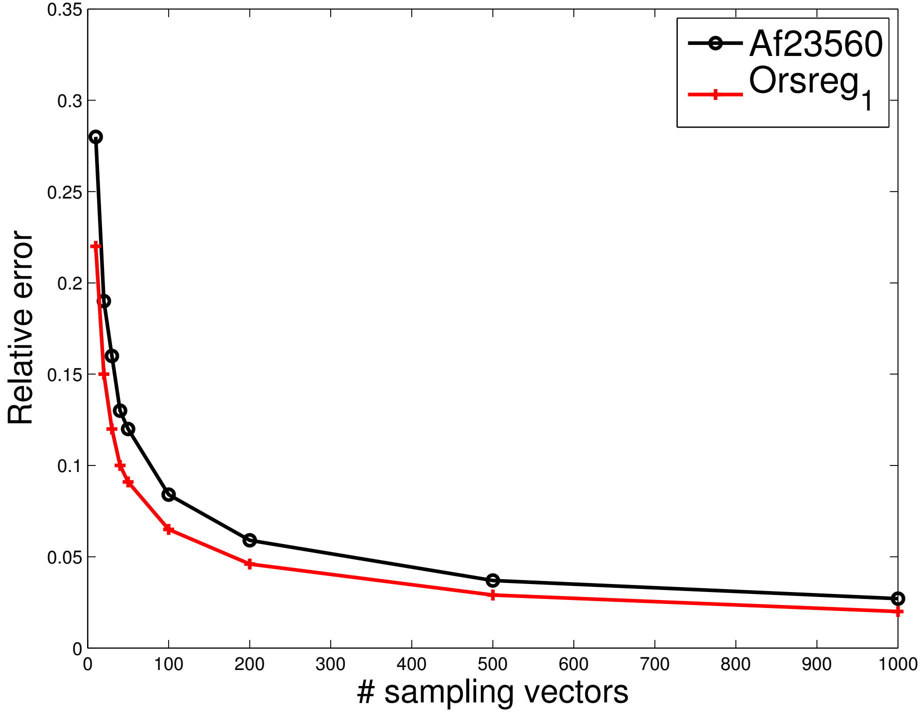
$$\text{Trace}(B) \approx \frac{1}{s}\sum_{j=1}^{s} v_j^T B v_j$$

- Hutchinson's estimator : take random vectors with components of the form $\pm 1/\sqrt{n}$ [Rademacher vectors]

- Extensively studied in literature. See e.g.: *Hutchinson '89; H. Avron and S. Toledo '11; G.H. Golub & U. Von Matt '97; Roosta-Khorasani & U. Ascher '15; ...*

Typical convergence curve for stochastic estimator

- Estimating the diagonal of inverse of two sample matrices



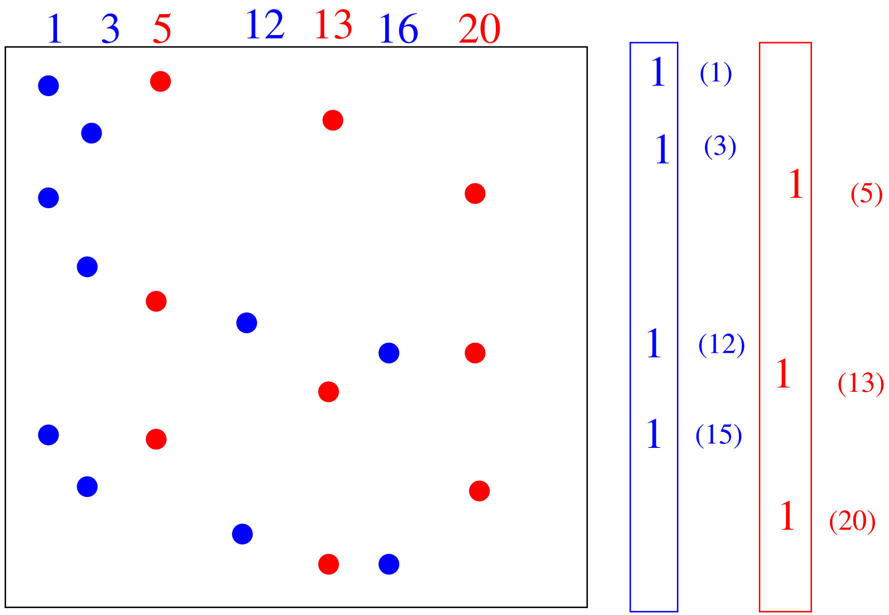
Alternative: standard probing

Basis of the method: Color columns of matrix so that no two columns of the same color overlap.

Entries of same color can be computed with 1 matvec

➤ Corresponds to coloring graph of $A^T A$.

➤ For problem of diag(A) need only color graph of A



In summary:

- Probing much more powerful when $f(A)$ is known to be nearly sparse (e.g. banded)..
- Approximate pattern (graph) can be obtained inexpensively
- Generally just a handful of probing vectors needed – Can be obtained by coloring graph
- However:
- Not as general: need $f(A)$ to be ‘ ϵ – sparse ’

References:

- *J. M. Tang and YS*, *A probing method for computing the diagonal of a matrix inverse*, Numer. Lin. Alg. Appl., 19 (2012), pp. 485–501.

See also (improvements)

- *Andreas Stathopoulos, Jesse Laeuchli, and Kostas Orginos* *Hierarchical Probing for Estimating the Trace of the Matrix Inverse on Toroidal Lattices* SISC, 2012. [somewhat specific to Lattice QCD]

- *E. Aune, D. P. Simpson, J. Eidsvik* [Statistics and Computing 2012] combine probing with stochastic estimation. Good improvements reported.

SPECTRAL DENSITIES

Spectral Densities - Introduction

- Spectral density == function that provides a global representation of the spectrum of a Hermitian matrix

- Known in solid state physics as *‘Density of States’* (DOS)

- Very useful in physics

- Almost unknown (as a tool) in numerical linear algebra

Density of States

- Formally, the Density Of States (DOS) of a matrix A is

$$\phi(t) = \frac{1}{n}\sum_{j=1}^{n} \delta(t - \lambda_j),$$

where:
- δ is the Dirac δ-function or Dirac distribution
- $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of A

- DOS is also referred to as the spectral density

- Note: number of eigenvalues in an interval $[a, b]$ is

$$\mu_{[a,b]} = \int_a^b \sum_j \delta(t - \lambda_j)\, dt \equiv \int_a^b n\phi(t) dt \ .$$

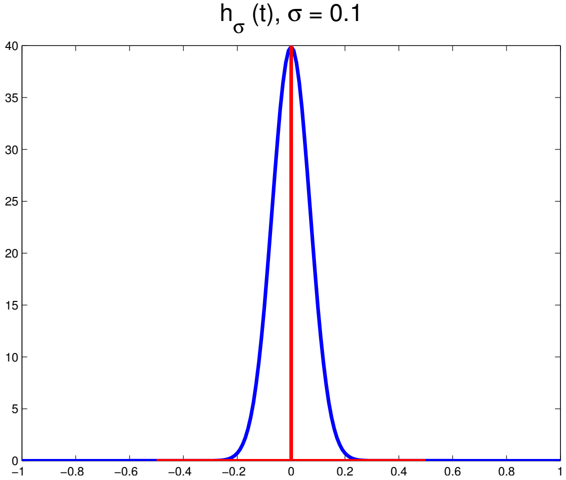
Issue: How to deal with distributions?

- Highly ‘discontinuous’, not easy to handle numerically

- Solution for practical and theoretical purposes: replace ϕ by a regularized (‘blurred’) version ϕ_σ:

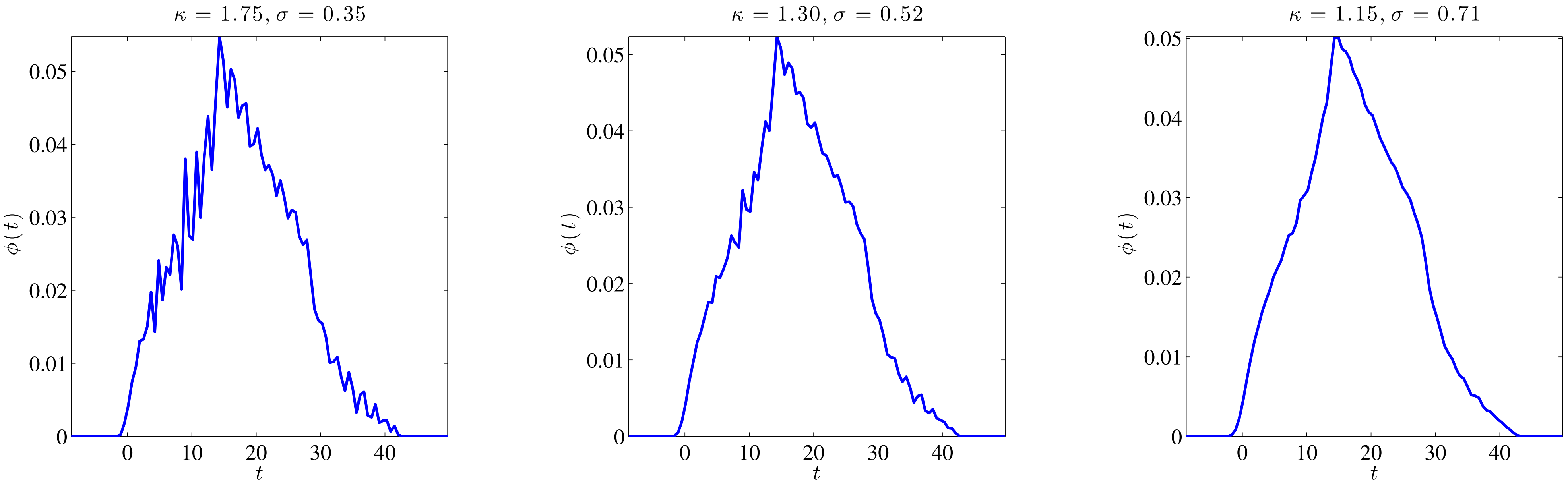
$$\phi_\sigma(t) = \frac{1}{n}\sum_{j=1}^{n} h_\sigma(t-\lambda_j),$$

Where, for example: $h_\sigma(t) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{t^2}{2\sigma^2}}$.



- Smoothed $\phi(t)$ can be viewed as a probability distribution function for the spectrum

How to select smoothing parameter σ? Example for Si_2

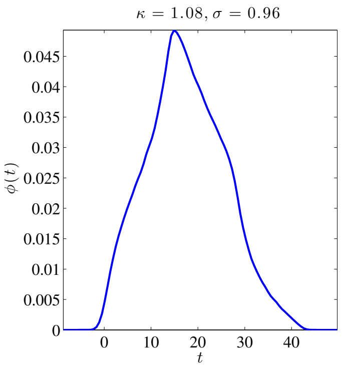


$\kappa = 1.75, \sigma = 0.35$

$\kappa = 1.30, \sigma = 0.52$

$\kappa = 1.15, \sigma = 0.71$

- Higher $\sigma \to$ smoother curve
- But loss of detail ..
- Compromise: $\sigma = \dfrac{h}{2\sqrt{2\log(\kappa)}}$,
- $h =$ resolution, $\kappa =$ parameter > 1



$\kappa = 1.08, \sigma = 0.96$

Computing the DOS: The Kernel Polynomial Method

- Used by Chemists to calculate the DOS – see Silver and Röder'94 , Wang '94, Drabold-Sankey'93, + others

- Basic idea: expand DOS into Chebyshev polynomials

- Use trace estimator to get traces needed in calculations ➤ Assume change of variable done so eigenvalues lie in $[-1,\ 1]$.

- To avoid weight function expand $\sqrt{1-t^2}\phi \rightarrow$

$$\hat{\phi}(t) = \sqrt{1-t^2} \times \frac{1}{n}\sum_{j=1}^{n} \delta(t-\lambda_j).$$

- Then, (full) expansion is: $\hat{\phi}(t) = \sum_{k=0}^{\infty} \mu_k T_k(t)$. Question: $\mu_k =??$

- Expansion coefficients μ_k are formally defined by:

$$\begin{aligned}\mu_k &= \frac{2-\delta_{k0}}{\pi}\int_{-1}^{1}\frac{1}{\sqrt{1-t^2}}T_k(t)\hat{\phi}(t)dt\\ &= \frac{2-\delta_{k0}}{\pi}\int_{-1}^{1}\frac{1}{\sqrt{1-t^2}}T_k(t)\sqrt{1-t^2}\phi(t)dt\\ &= \frac{2-\delta_{k0}}{n\pi}\sum_{j=1}^{n}T_k(\lambda_j).\end{aligned}$$

- Here $2-\delta_{k0}$ == 1 when $k = 0$ and == 2 otherwise.

- Note: $\boxed{\sum T_k(\lambda_i) = \textit{Trace}[T_k(A)]}$ ⟶ Estimate this, e.g., via stochastic estimator

- Generate random vectors $v^{(1)}, v^{(2)}, \cdots, v^{(n_{\text{vec}})}$

- Each vector is normalized so that $\|v^{(l)}\| = 1, l = 1, \ldots, n_{\text{vec}}$.

- Estimate the trace of $T_k(A)$ with stochastisc estimator:

$$\text{Trace}(T_k(A)) \approx \frac{1}{n_{\text{vec}}} \sum_{l=1}^{n_{\text{vec}}} \left(v^{(l)}\right)^T T_k(A) v^{(l)}.$$

- Will lead to the desired estimate:

$$\mu_k \approx \frac{2 - \delta_{k0}}{n \pi n_{\text{vec}}} \sum_{l=1}^{n_{\text{vec}}} \left(v^{(l)}\right)^T T_k(A) v^{(l)}.$$

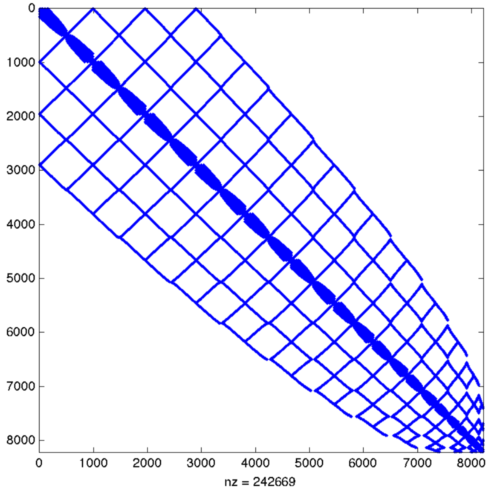
- To compute scalars of the form $v^T T_k(A) v$, exploit 3-term recurrence of the Chebyshev polynomial: $T_{k+1}(A)v = 2AT_k(A)v - T_{k-1}(A)v$

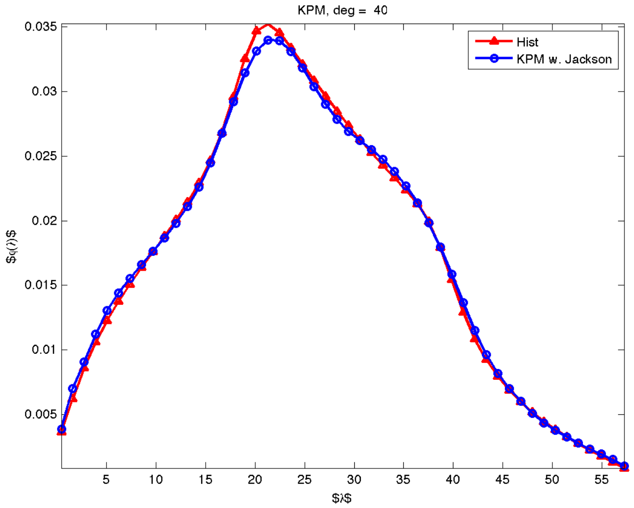
- If we let $v_k \equiv T_k(A)v$, we have

$$v_{k+1} = 2Av_k - v_{k-1}$$

An example: The Benzene matrix

```
>> TestKpmDos
 Matrix Benzene n =8219  nnz = 242669
Degree =  40  # sample vectors =  10
Elapsed time is 0.235189 seconds.
```





Use of the Lanczos Algorithm

- Recall: The Lanczos algorithm generates an orthonormal basis $V_m = [v_1, v_2, \cdots, v_m]$ for the Krylov subspace:

$$\boxed{\mathbf{span}\{v_1, Av_1, \cdots, A^{m-1}v_1\}}$$

- ... such that:

$V_m^H A V_m = T_m$ - with

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{pmatrix}$$

- Lanczos process builds orthogonal polynomials wrt to dot product:

$$\int p(t)q(t)dt \equiv (p(A)v_1, q(A)v_1)$$

- Let $\theta_i, \ i = 1 \cdots , m$ be the eigenvalues of T_m [Ritz values]
- y_i's associated eigenvectors; Ritz vectors: $\{V_m y_i\}_{i=1:m}$
- Ritz values approximate eigenvalues
- Could compute θ_i's then get approximate DOS from these
- Problem: θ_i not good enough approximations – especially inside the spectrum.

Better idea: exploit relation of Lanczos with (discrete) orthogonal polynomials and related Gaussian quadrature:

$$\int p(t)dt \approx \sum_{i=1}^{m} a_i p(\theta_i) \quad a_i = \left[e_1^T y_i\right]^2$$

- See, e.g., Golub & Meurant '93, and also Gautschi'81, Golub and Welsch '69.

- Formula exact when p is a polynomial of degree $\leq 2m+1$

- Consider now $\int p(t)dt = < p, 1 > =$ (Stieljes) integral $\equiv$

$$\boxed{(p(A)v, v) = \sum \beta_i^2 p(\lambda_i) \equiv < \phi_v, p >}$$

- Then $\langle \phi_v, p \rangle \approx \sum a_i p(\theta_i) = \sum a_i \langle \delta_{\theta_i}, p \rangle \rightarrow$

$$\boxed{\phi_v \approx \sum a_i \delta_{\theta_i}}$$

- To mimick the effect of $\beta_i = 1, \forall i$, use several vectors v and average the result of the above formula over them..

• *Approximating spectral densities of large matrices*, Lin Lin, YS, and Chao Yang - SIAM Review '16. Also in:
[arXiv: http://arxiv.org/abs/1308.5467]

Application 1: Eigenvalue counts

Problem: Given A (Hermitian) find an estimate of the number $\mu_{[a,b]}$ of eigenvalues of A in $[a, \quad b]$.

Standard method: Sylvester inertia theorem → expensive!

First alternative: integrate the Spectral Density in $[a, \ b]$.

$$\mu_{[a,b]} \approx n \sum_{k=0}^{m} \mu_k \left(\int_a^b \frac{T_k(t)}{\sqrt{1-t^2}} dt \right) = \ldots$$

Second method: Estimate trace of the related spectral projector P (→ u_i's = eigenvectors ↔ λ_i's)

$$P = \sum_{\lambda_i \, \in \, [a \ b]} u_i u_i^T.$$

- It turns out that the 2 methods are identical.

Application 3: Estimating the rank

- Very important problem in signal processing applications, machine learning, etc.

- Often: a certain rank is selected ad-hoc. Dimension reduction is application with this “guessed” rank.

- Can be viewed as a particular case of the eigenvalue count problem - but need a cutoff value..

Approximate rank, Numerical rank

- Notion defined in various ways. A common one:

$$r_\epsilon = \min\{rank(B) : B \in \mathbb{R}^{m\times n}, \|A - B\|_2 \le \epsilon\},$$

$$r_\epsilon = \text{ Number of sing. values } \ge \epsilon$$

- Two distinct problems:

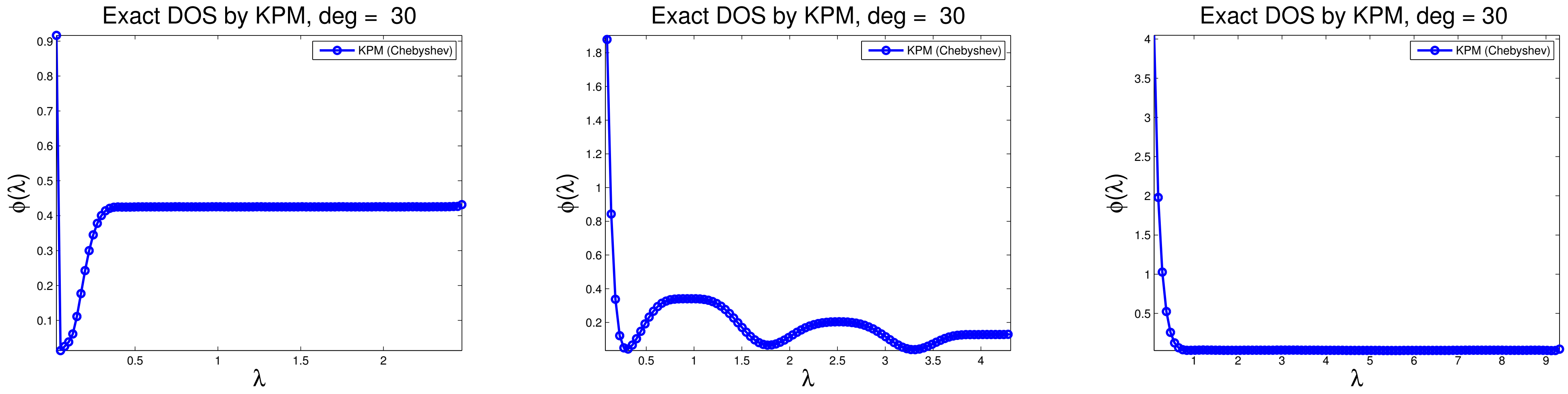
1. Get a good ϵ 2. Estimate number of sing. values $\ge \epsilon$

- We will need a cut-off value ('threshold') ϵ.

- Could use 'noise level' for ϵ, but not always available

Threshold selection

- How to select a good threshold?
- Answer: Obtain it from the DOS function



(A) (B) (C)

Exact DOS plots for three different types of matrices.

- To find: point immediatly following the initial sharp drop observed.

- Simple idea: use derivative of DOS function ϕ

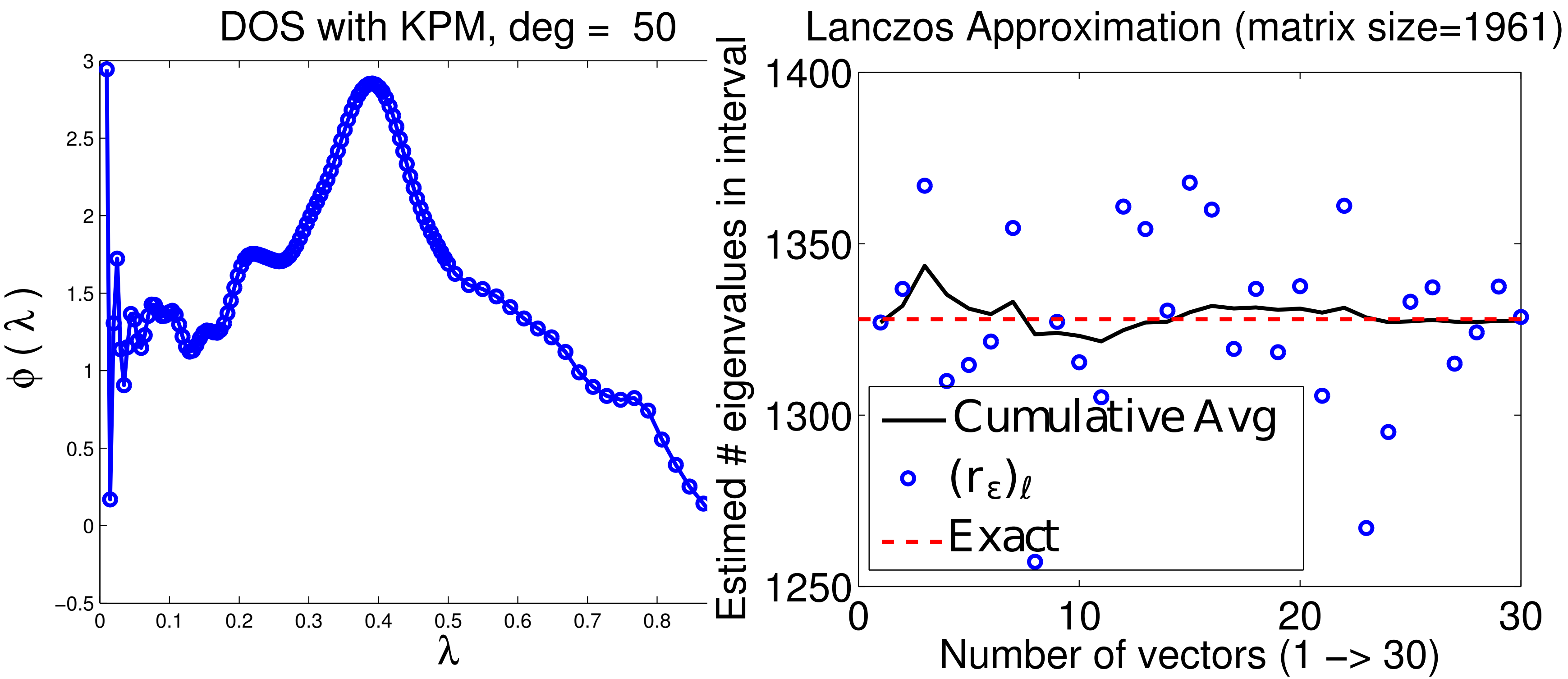
- For an $n \times n$ matrix with eigenvalues $\lambda_n \le \lambda_{n-1} \le \cdots \le \lambda_1$:

$$\epsilon = \min\{t : \lambda_n \le t \le \lambda_1, \phi'(t) = 0\}.$$

- In practice replace by

$$\epsilon = \min\{t : \lambda_n \le t \le \lambda_1, |\phi'(t)| \ge \text{tol}\}$$

Experiment: estimated rank by Lanczos for matrix `netz4504`*.*



(A) (B)

(A) The DOS found by KPM. (B) Approximate rank estimation by Lanczos

Tests with Matérn covariance matrices for grids

- Important in statistical applications

Approximate Rank Estimation of Matérn covariance matrices

Type of Grid (dimension)	Matrix	# λ_i's	r_ϵ	
	Size	$\geq \epsilon$	KPM	Lanczos
1D regular Grid (2048×1)	2048	16	16.75	15.80
1D no structure Grid (2048×1)	2048	20	20.10	20.46
2D regular Grid (64×64)	4096	72	72.71	72.90
2D no structure Grid (64×64)	4096	70	69.20	71.23
2D deformed Grid (64×64)	4096	69	68.11	69.45

- For all test $M(deg) = 50, n_v$=30

Application 4: The LogDeterminant

Evaluate the Log-determinant of A:

$$\log\det(A) = \text{Trace}(\log(A)) = \sum_{i=1}^{n} \log(\lambda_i).$$

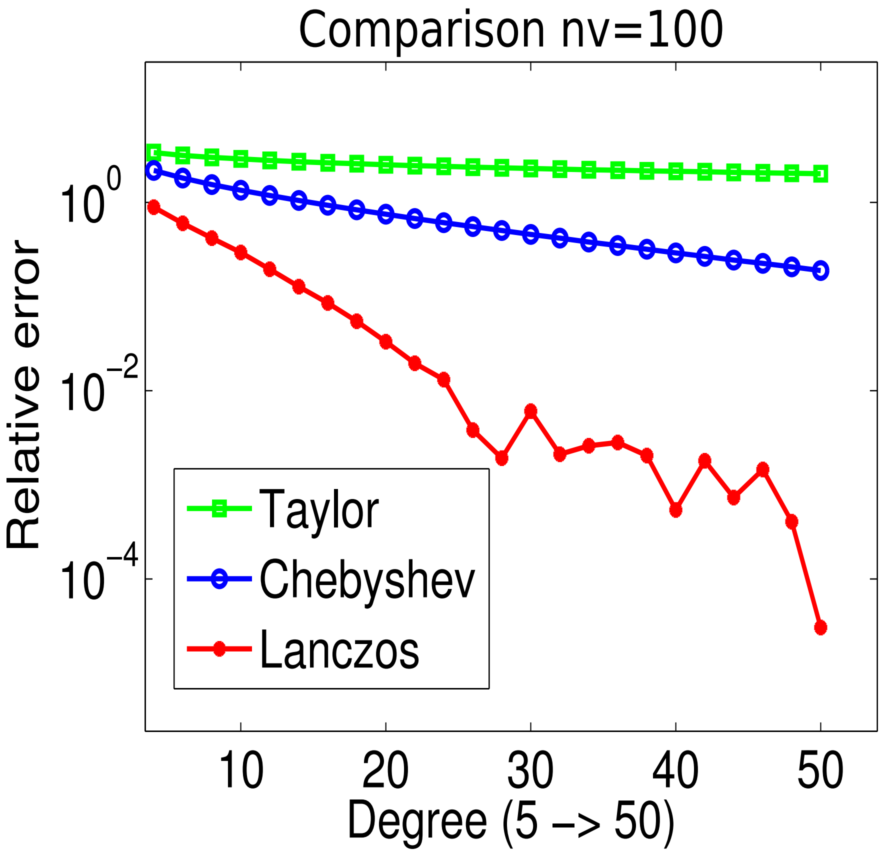
A is SPD.

➤ Estimating the log-determinant of a matrix equivalent to estimating the trace of the matrix function $f(A) = \log(A)$.

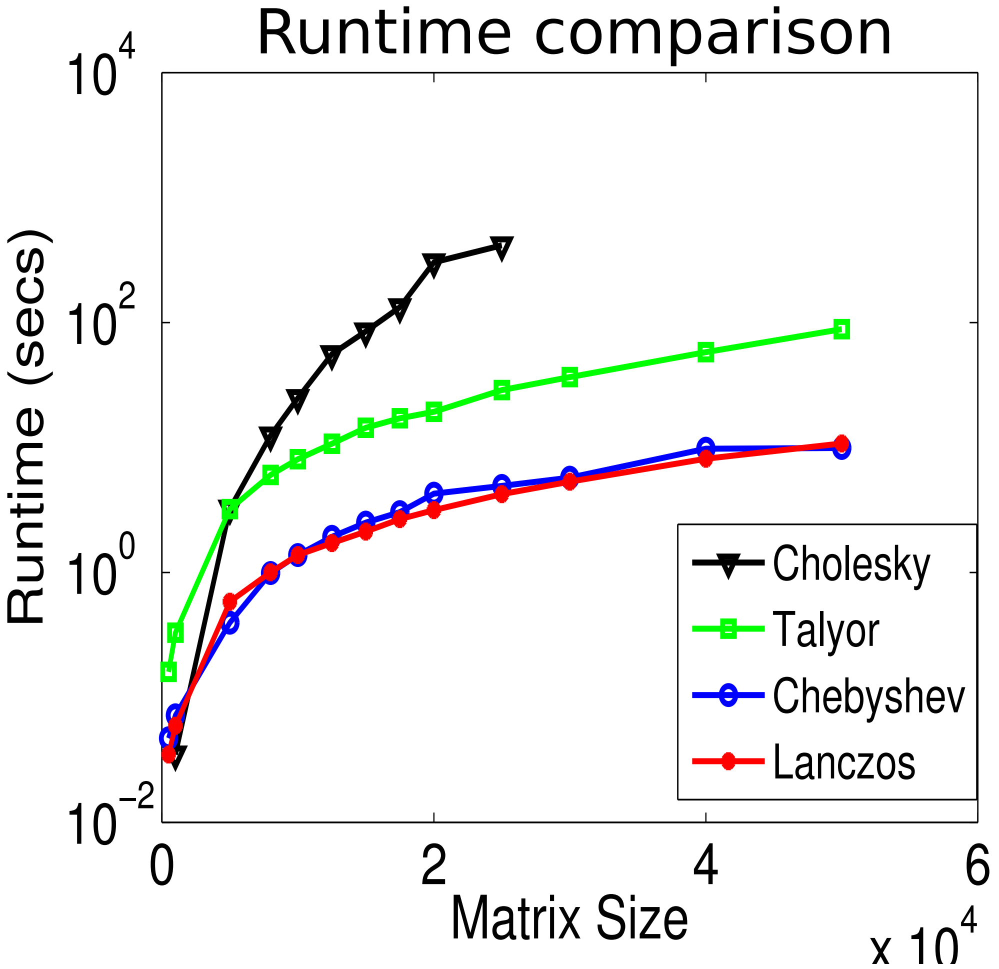
➤ Can invoke Stochastic Lanczos Quadrature (SLQ) to estimate this trace.

Numerical example: A graph Laplacian `california` of size 9664×9664, $nz \approx 10^5$ from the Univ. of Florida collection.

Rel. error vs degree



- 3 methods: Taylor Series, Chebyshev expansion, SLQ
- # starting vectors $nv = 100$ in all three cases.



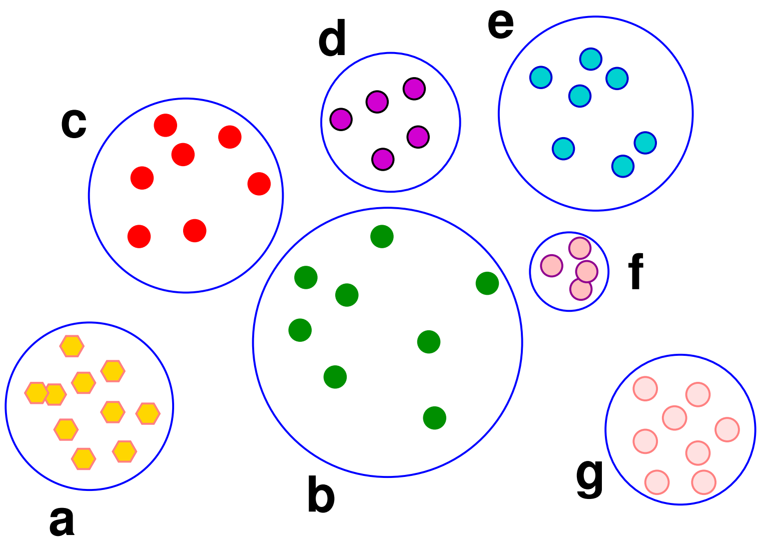
- ➤ Many more applications!

SUPERVISED LEARNING

Supervised learning

- We now have data that is ‘labeled’

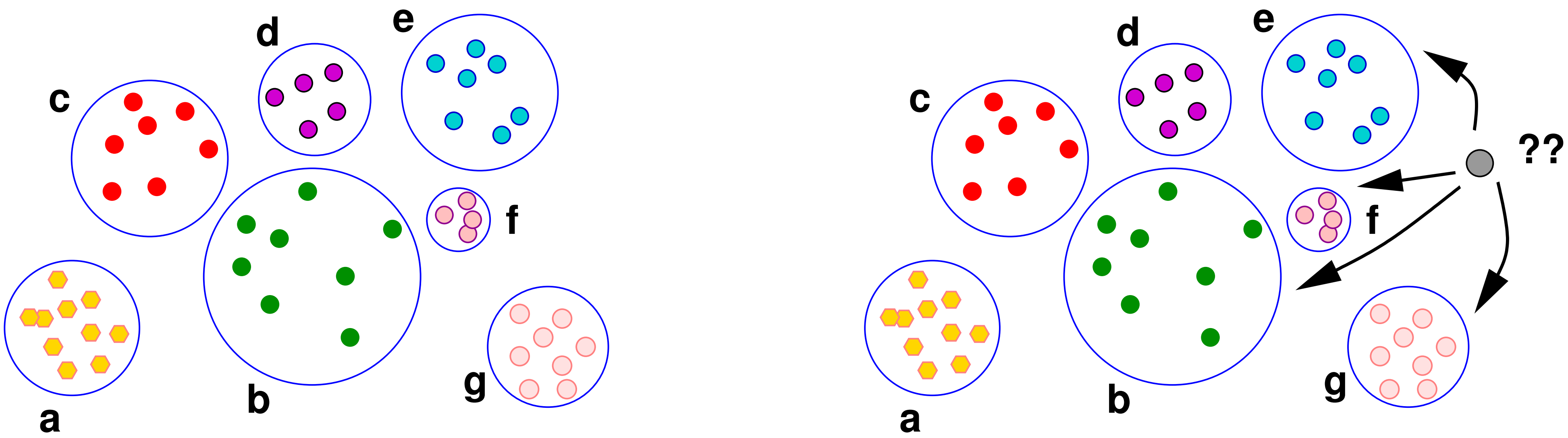
Examples: Health Sciences (‘malignant’- ’non malignant’) ; Materials (’photovoltaic’, ’hard’, ’conductor’, ...) ; Digit Recognition (’0’, ’1’, ..., ’9’)



Supervised learning

- We now have data that is ‘labeled’

Examples: Health Sciences (‘malignant’- ’non malignant’) ; Materials (’photovoltaic’, ’hard’, ’conductor’, ...) ; Digit Recognition (’0’, ’1’, ..., ’9’)

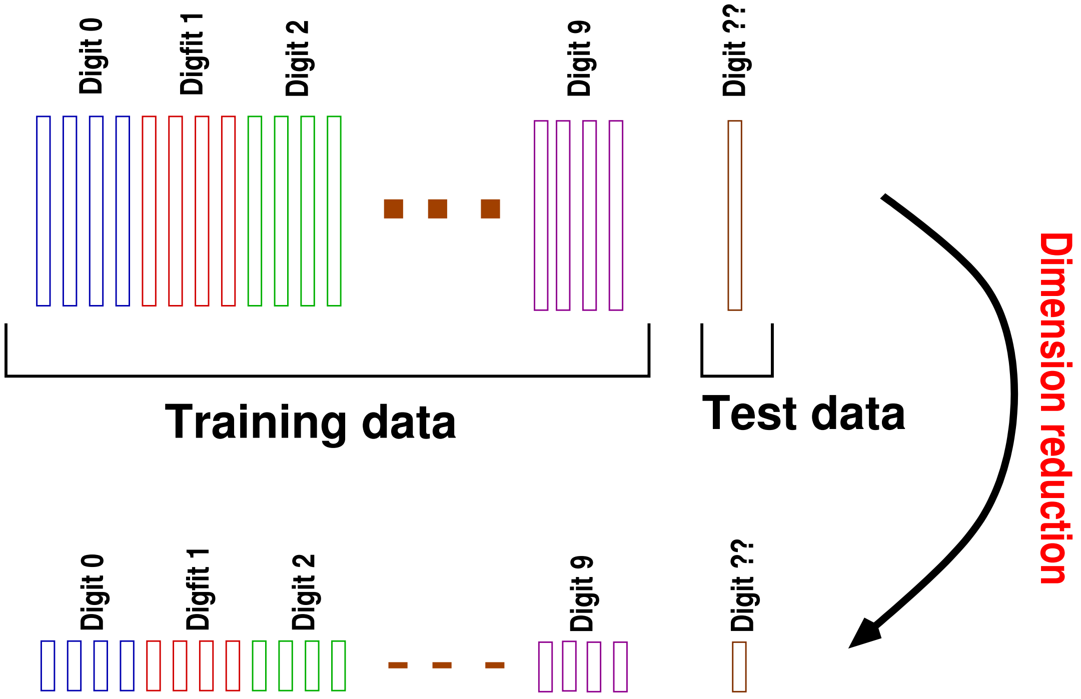


Supervised learning: classification

- Best illustration: written digits recognition example

Given: set of labeled samples (training set), and an (unlabeled) test image x.

Problem: label of x =?



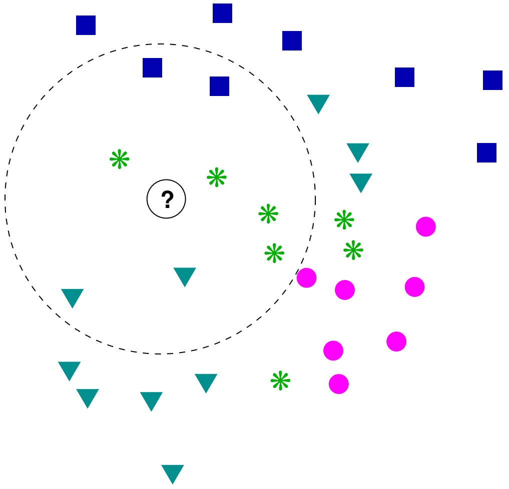
- Roughly speaking: we seek dimension reduction so that recognition is 'more effective' in low-dim. space

Basic method: K-nearest neighbors (KNN) classification

- Idea of a voting system: get distances between test sample and training samples

- Get the k nearest neighbors (here $k = 8$)

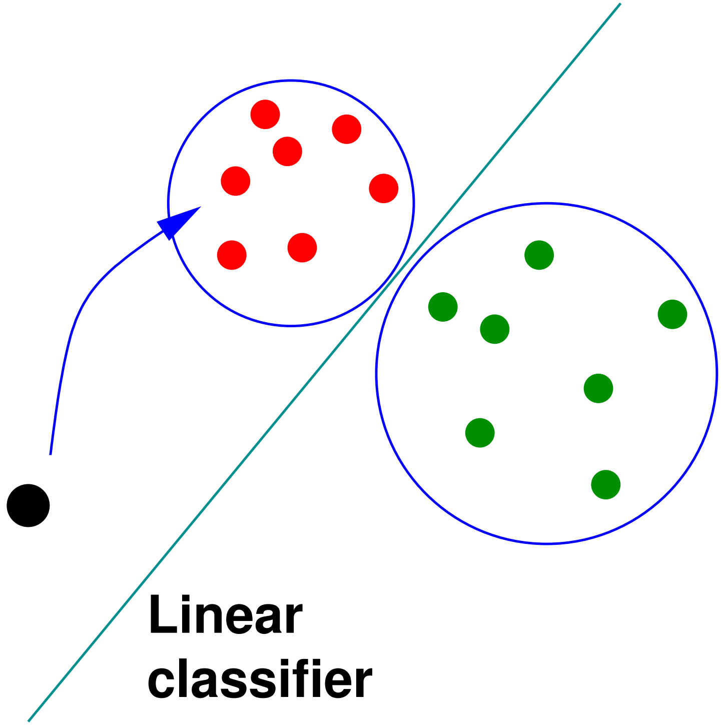
- Predominant class among these k items is assigned to the test sample ("$*$" here)



Supervised learning: Linear classification

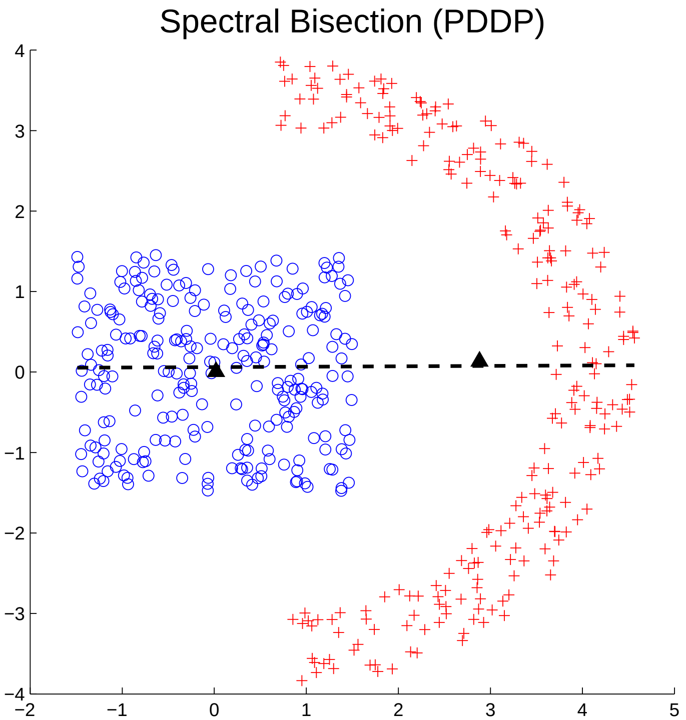
Linear classifiers: Find a hyperplane which best separates the data in classes A and B.

➤ Example of application: Distinguish between SPAM and non-SPAM e-mails

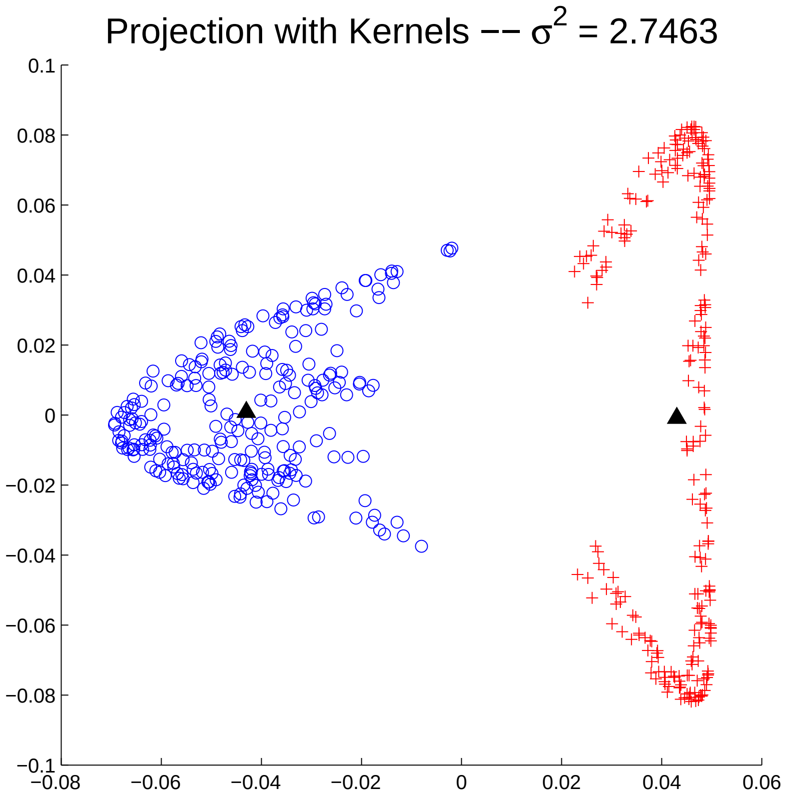


➤ Note: The world in non-linear. Often this is combined with Kernels – amounts to changing the inner product

A harder case:



- ➤ Use kernels to transform



Transformed data with a Gaussian Kernel

Simple linear classifiers

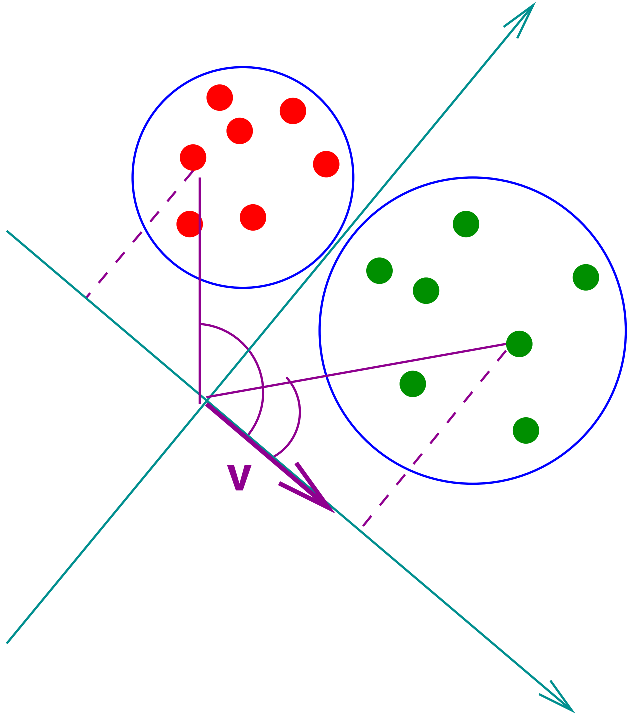
➤ Let $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n]$ be the data matrix.

➤ and $\boldsymbol{L} = [\boldsymbol{l}_1, \cdots, \boldsymbol{l}_n]$== labels. $\boldsymbol{l}_i = \pm\boldsymbol{1}$

➤ 1st Solution: Find a vector $\boldsymbol{u}$ such that $\boldsymbol{u}^T\boldsymbol{x}_i$ close to $\boldsymbol{l}_i$, $\forall \boldsymbol{i}$

➤ Common solution: SVD to reduce dimension of data [e.g. 2-D] then do comparison in this space. e.g.

A: $\boldsymbol{u}^T\boldsymbol{x}_i \geq 0$, B: $\boldsymbol{u}^T\boldsymbol{x}_i < 0$



[For clarity: principal axis $\boldsymbol{u}$ drawn below where it should be]

Fisher's Linear Discriminant Analysis (LDA)

Principle: Use label information to build a good projector, i.e., one that can 'discriminate' well between classes

- ➤ Define "between scatter": a measure of how well separated two distinct classes are.

- ➤ Define "within scatter": a measure of how well clustered items of the same class are.

- ➤ Objective: make "between scatter" measure large and "within scatter" small.

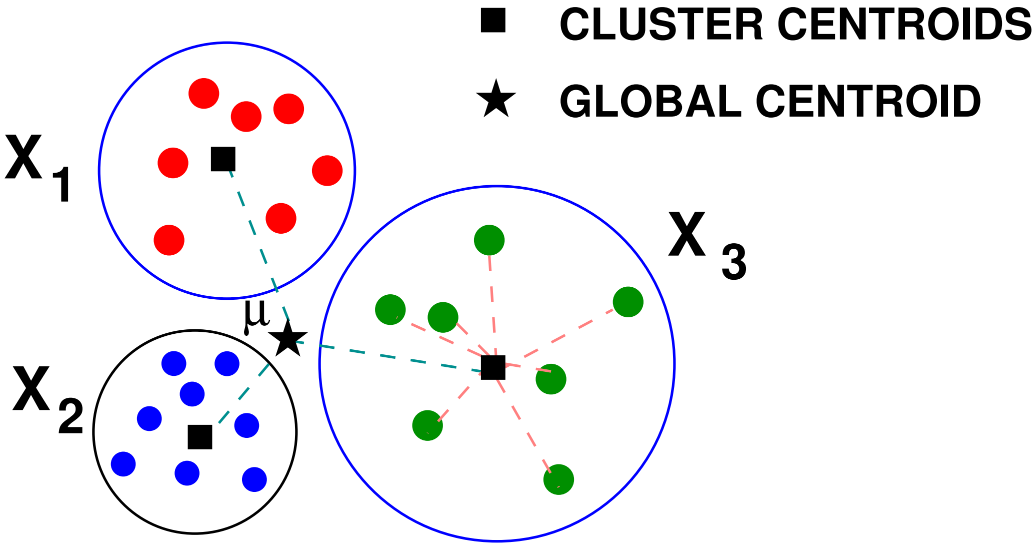
Idea: Find projector that maximizes the ratio of the "between scatter" measure over "within scatter" measure

$$S_B = \sum_{k=1}^{c} n_k(\mu^{(k)} - \mu)(\mu^{(k)} - \mu)^T,$$

$$S_W = \sum_{k=1}^{c} \sum_{x_i \in X_k} (x_i - \mu^{(k)})(x_i - \mu^{(k)})^T$$

where:

- $\mu = \text{mean}\ (X)$
- $\mu^{(k)} = \text{mean}\ (X_k)$
- $X_k = k$-th class
- $n_k = |X_k|$



- Consider 2nd moments for a vector $\boldsymbol{a}$:

$$\boldsymbol{a}^T S_B \boldsymbol{a} = \sum_{i=1}^{c} n_k \, |\boldsymbol{a}^T(\boldsymbol{\mu}^{(k)} - \boldsymbol{\mu})|^2,$$

$$\boldsymbol{a}^T S_W \boldsymbol{a} = \sum_{k=1}^{c} \sum_{x_i \in X_k} |\boldsymbol{a}^T(\boldsymbol{x}_i - \boldsymbol{\mu}^{(k)})|^2$$

- $\boldsymbol{a}^T S_B \boldsymbol{a} \equiv$ weighted variance of projected $\boldsymbol{\mu}_j$'s

- $\boldsymbol{a}^T S_W \boldsymbol{a} \equiv$ w. sum of variances of projected classes X_j's

- LDA projects the data so as to maximize the ratio of these two numbers:

$$\boxed{\max_{a} \frac{\boldsymbol{a}^T S_B \boldsymbol{a}}{\boldsymbol{a}^T S_W \boldsymbol{a}}}$$

- Optimal $\boldsymbol{a}$ = eigenvector associated with top eigenvalue of:

$$S_B \boldsymbol{u}_i = \lambda_i S_W \boldsymbol{u}_i \; .$$

LDA – Extension to arbitrary dimensions

- Criterion: maximize the ratio of two traces: $\frac{\text{Trace}[U^T S_B U]}{\text{Trace}[U^T S_W U]}$

- Constraint: $U^T U = I$ (orthogonal projector).

- Reduced dimension data: $Y = U^T X$.

Common viewpoint: hard to maximize, therefore ...

- ... alternative: Solve instead the ('easier') problem: $\max_{U^T S_W U = I} \text{Trace}[U^T S_B U]$

- Solution: largest eigenvectors of $S_B u_i = \lambda_i S_W u_i$.

In Brief: Support Vector Machines (SVM)

- Similar in spirit to LDA. Formally, SVM finds a hyperplane that best separates two training sets belonging to two classes.

- If the hyperplane is: $\boxed{w^T x + b = 0}$

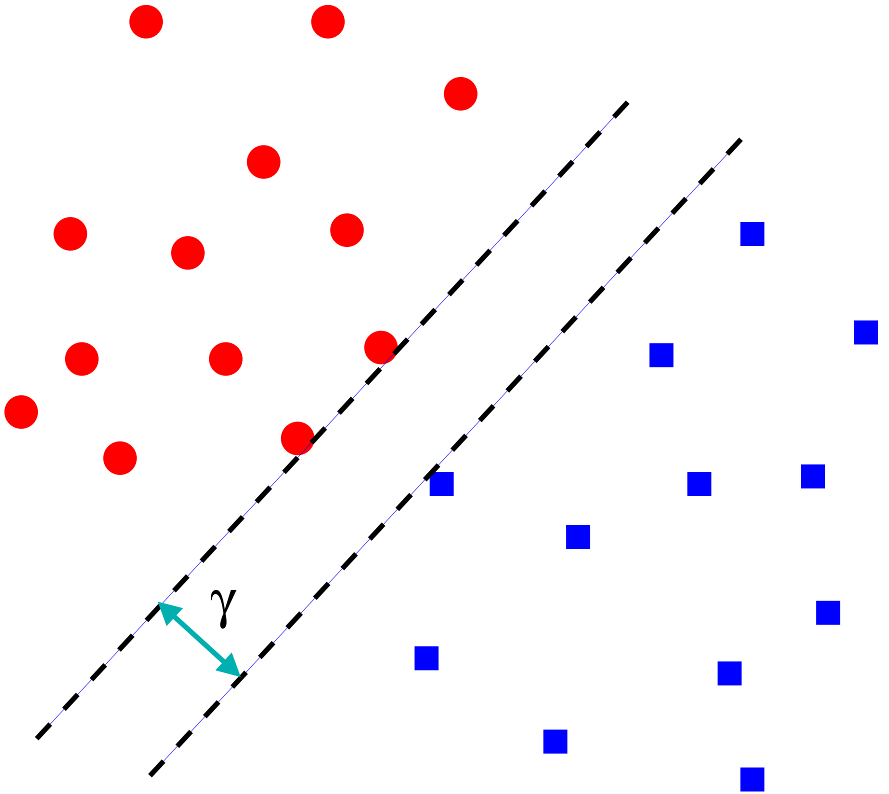
- Then the classifier is $f(x) = \textit{sign}(w^T x + b)$: assigns $y = +1$ to one class and $y = -1$ to other

- Normalize parameters w, b by looking for hyperplanes of the form $w^T x + b \geq 1$ to include one set and $w^T x + b \leq -1$ to include the other.

- With $y_i = +1$ for one class and $y_i = -1$ for the other, we can write the constraints as $y_i(w^T x_i + b) \geq 1$.

- The margin is the maximum distance between two such planes: goal find w, b to maximize margin.

- Maximize margin subject to the constraint $y_i(w^T x_i + b) \geq 1$.



- As it turns out the margin is equal to: $\boxed{\gamma = \frac{2}{\|w\|_2}}$

✍ Prove it.

- Need to solve the constrained quadratic programming problem:

$$\boxed{\begin{array}{ll} \min\limits_{w.b} & \frac{1}{2}\|w\|_2^2 \\ \text{s.t.} & y_i(w^Tx_i + b) \geq 1, \quad \forall x_i. \end{array}}$$

Modification 1: Soft margin. Consider hinge loss: $\max\{0, 1 - y_i[w^Tx_i + b]\}$

- Zero if constraint satisfied for pair x_i, y_i. Otherwise proportional to distance from corresponding hyperplane. Hence we can minimize

$$\lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i[w^Tx_i + b]\}$$

✍ Suppose $y_i = +1$ and let $d_i = 1 - y_i[w^Tx_i + b]$. Show that the distance between x_i and hyperplane $\boxed{w^Tx_i + b = +1}$ is $d_i/\|w\|$.

Modification 2 : Use in combination with a Kernel to improve separability

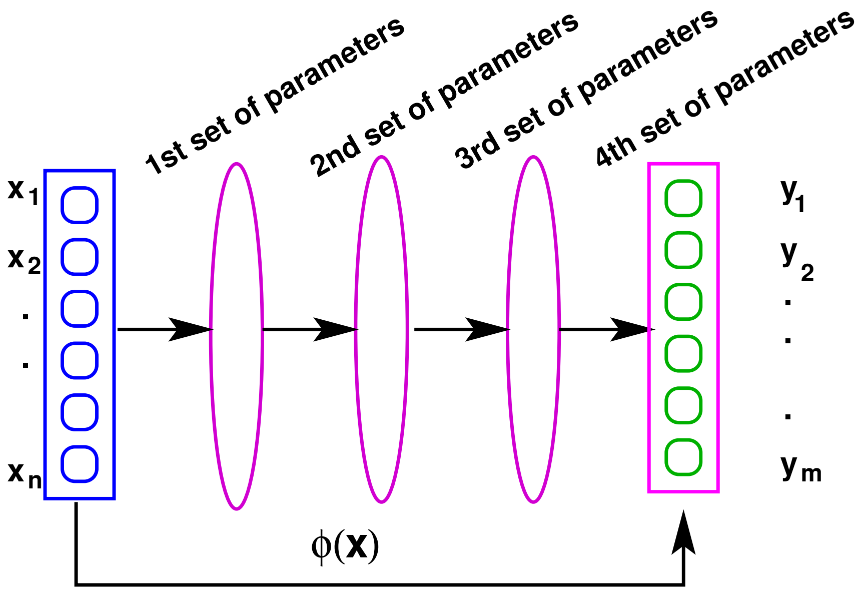
A few words on Deep Neural Networks (DNNs)

➤ Ideas of neural networks goes back to the 1960s - were popularized in early 1990s – then laid dormant until recently.

➤ Two reasons for the come-back:

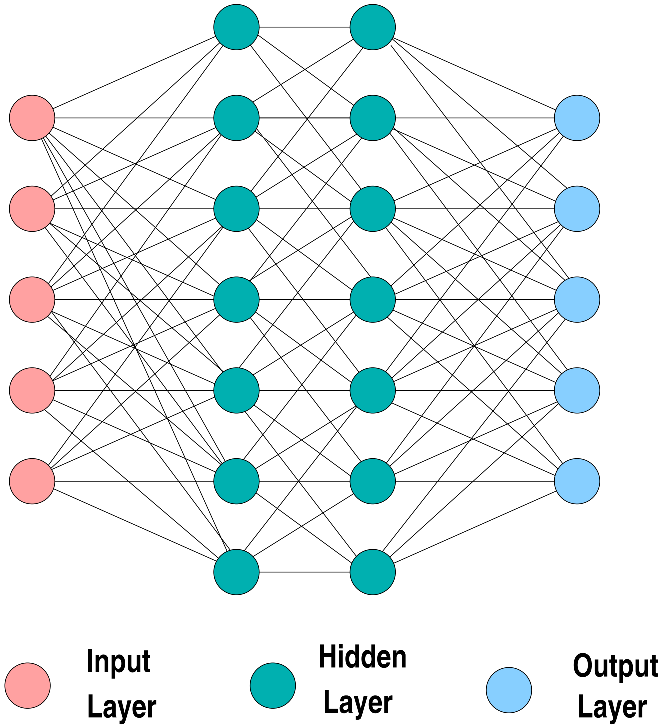
- DNN are remarkably effective in some applications
- big progress made in hardware [$\rightarrow$ affordable ‘training cost’]

➤ Training a neural network can be viewed as a problem of approximating a function ϕ which is defined via sets of parameters:



Problem: find sets of parameters such that $\phi(x) \approx y$

```
Input: $x$, Output: $y$
Set: $z_0 = x$
For $l = 1 : \mathtt{L+1}$ Do:
    $z_l = \sigma(W_l^T z_{l-1} + b_l)$
End
Set: $y = \phi(x) := z_{L+1}$
```



- layer # 0 = input layer
- layer # $(L + 1)$ = output layer

➤ A matrix W_l is associated with layers 1,2, $L + 1$.

➤ Problem: Find ϕ (i.e., matrices W_l) s.t. $\phi(x) \approx y$

DNN (continued)

- Problem is not convex, highly parameterized, ...,
- .. Main method used: Stochastic gradient descent [basic]
- It all looks like alchemy... but it works well for certain applications
- Training is still quite expensive – GPUs can help
- *Very* active area of research

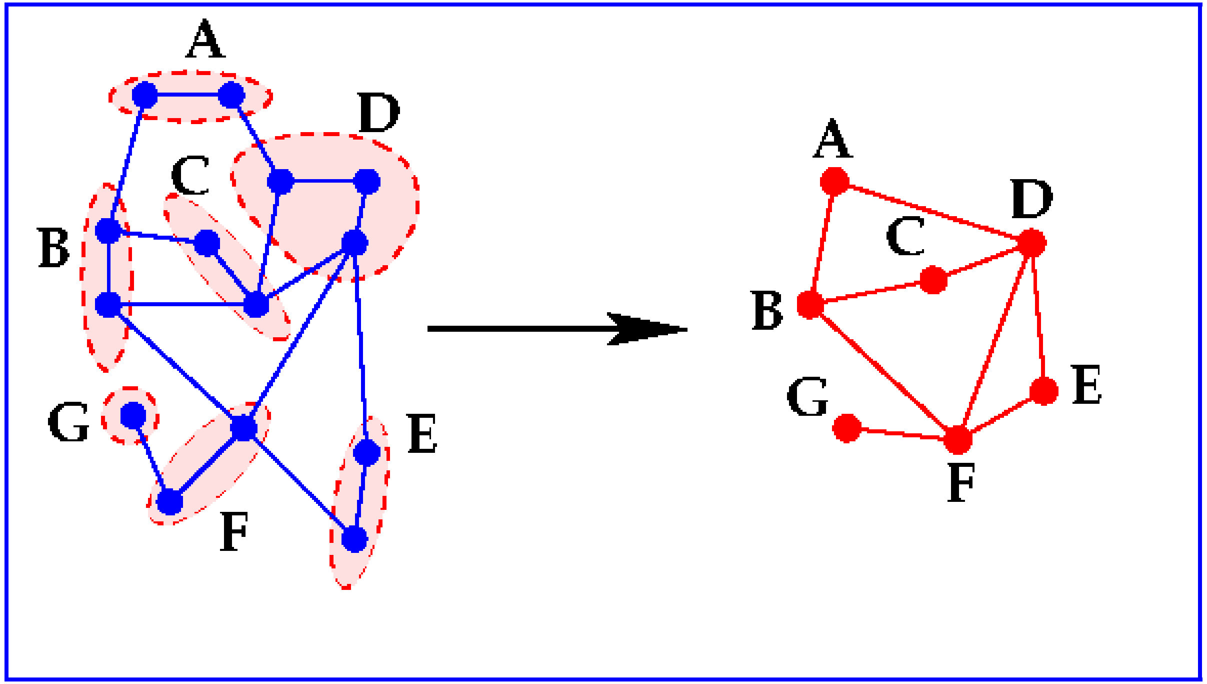
GRAPH COARSENING

Graph coarsening

Given a graph $G = (V, E)$, goal of graph coarsening is to find a smaller graph $G_c = (V_c, E_c)$ with n_c nodes and m_c edges, where $n_c < n$, which is a faithful approximation of G in some sense.

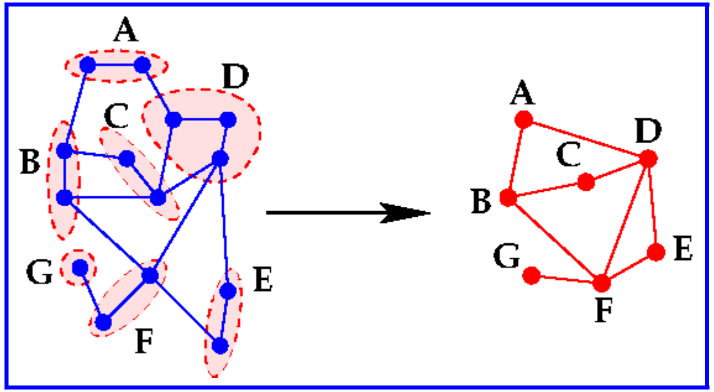
Notation:

- A_c = adjacency matrix of G_c;
- L_c = graph Laplacian of G_c



Graph Coarsening in scientific computing

➤ Goal : exploit coarse representation of problem to solve linear systems

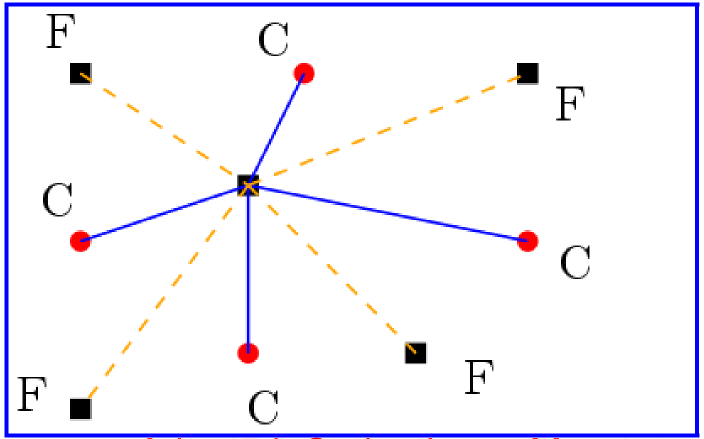


➤ Fewer nodes so: cheaper

➤ Can be used recursively

➤ Success story: *Multigrid, Algebraic Multigrid*

➤ *AMG:* Define coarse / fine nodes based on 'strength of coupling' →



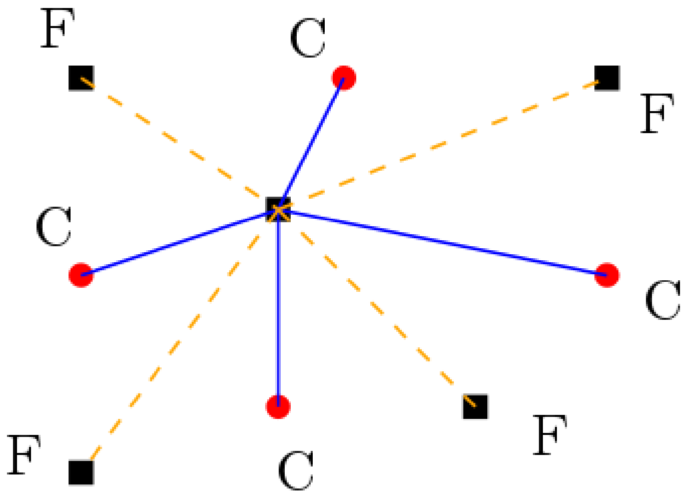
Graph coarsening in scientific computing: (A) MG

Algebraic multigrid Main idea: generalize the *interpolation* and *restriction* operations of standard MG.

➤ For each fine node select a set of nearest neighbors from the coarse set

➤ Then express a fine node i as a linear combination of a selected number of nearest neighbors that form a set C_i:

Fine nodes: ■. Coarse: ● In coarsening: central fine node is expressed as a combination of its coarse neighbors.



- Classical Ruge-Stüben strategy: selection based on ‘strong connections’ of node (i and j are strongly connected if a_{ij} has a large magnitude relative to others)

- Let C == set of coarse nodes; F == set of fine nodes

- Can define ‘interpolation operator’ P:

$$[Px]_i = \begin{cases} x_i & \text{if } i \ \in C, \\ \sum_{j \in C_i} p_{ij} x_j & \text{otherwise.} \end{cases}$$

- Expand into a multilevel framework by repeating the process on the graph of coarse set.

➤ Let $G_0 \equiv G$ (orig.) and $G_1, G_2, \ldots, G_h$ be sequence of coarse graphs: $G_\ell = (V_\ell, E_\ell)$ is obtained by coarsening on $G_{\ell-1}$ for $1 \leq \ell < L$.

➤ Let $A^{(0)} \equiv A$ and $A^{(\ell)} \equiv$ matrix associated of ℓ-th level.

➤ Linear system at the ℓ-th level, can be reordered as:

$$A^{(\ell)} = \begin{bmatrix} A_{CC}^{(\ell)} & A_{CF}^{(\ell)} \\ A_{FC}^{(\ell)} & A_{FF}^{(\ell)} \end{bmatrix}, \quad f^{(\ell)} = \begin{bmatrix} f_C^{(\ell)} \\ f_F^{(\ell)} \end{bmatrix}.$$

➤ AMG: exploit different levels to building approximate solution. An AMG scheme depends entirely on defining a sequence of interpolation operators P_ℓ for $\ell = 0, 1, \ldots$

➤ Once the P_ℓ's are defined, one can design various ‘cycles’ : processes of going back and forth between levels

Multilevel ILU preconditioner based on coarsening

- Method: find a good ordering for ILU preconditioner
- Example: technique presented in [D. Osei-Kuffuor et al, '06]:
- Ingredient: ordering based on coarsening + apply recursively
- Matrix is ordered in block form - then $A_{22}^{(0)}$ is in turn reordered:

$$\begin{bmatrix} A_{11}^{(0)} & A_{12}^{(0)} \\ A_{21}^{(0)} & A_{22}^{(0)} \end{bmatrix} \quad \rightarrow \quad \left[\begin{array}{c|cc} A_{11}^{(0)} & \multicolumn{2}{c}{A_{12}^{(0)}} \\ \hline A_{21}^{(0)} & \begin{array}{c|c} A_{11}^{(1)} & A_{12}^{(1)} \\ \hline A_{21}^{(1)} & A_{22}^{(1)} \end{array} \end{array}\right] .$$

- Repeat with $A_{22}^{(1)}$ and further down for a few levels.
- Do ILU factorization of the resulting reordered system.

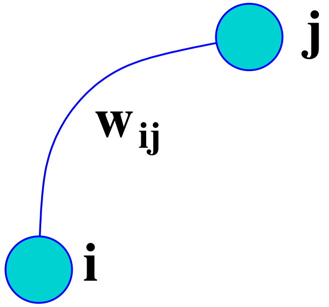
Example: Multilevel ILU [D. Osei-Kuffuor, R. Li, YS, '15]

Goal: Form of ILU preconditioning with improved robustness

- Traverse edges $(i,j) \in Nz(A)$ in decreasing order of the weights:

$$w_{ij} = \min\left\{ \frac{|a_{ij}|}{\delta_r(i)}, \frac{|a_{ij}|}{\delta_c(j)} \right\} \text{ where:}$$

$$\delta_r(i) = \frac{\|A_{i,:}\|_1}{nz(A_{i,:})} \quad \text{and} \quad \delta_c(j) = \frac{\|A_{:,j}\|_1}{nz(A_{:,j})}$$



- Select i as 'coarse' if $\sigma_i > \sigma_j$ and j otherwise, where →

$$\boxed{\sigma_k = \frac{|a_{kk}|}{\delta_r(k)\delta_c(k)}}$$

- Goal: to put large entries in the blocks $(A^{(\ell)}_{CF})$ and $(A^{(\ell)}_{FC})$ $\begin{bmatrix} A^{(\ell)}_{CC} & A^{(\ell)}_{CF} \\ A^{(\ell)}_{FC} & A^{(\ell)}_{FF} \end{bmatrix}$

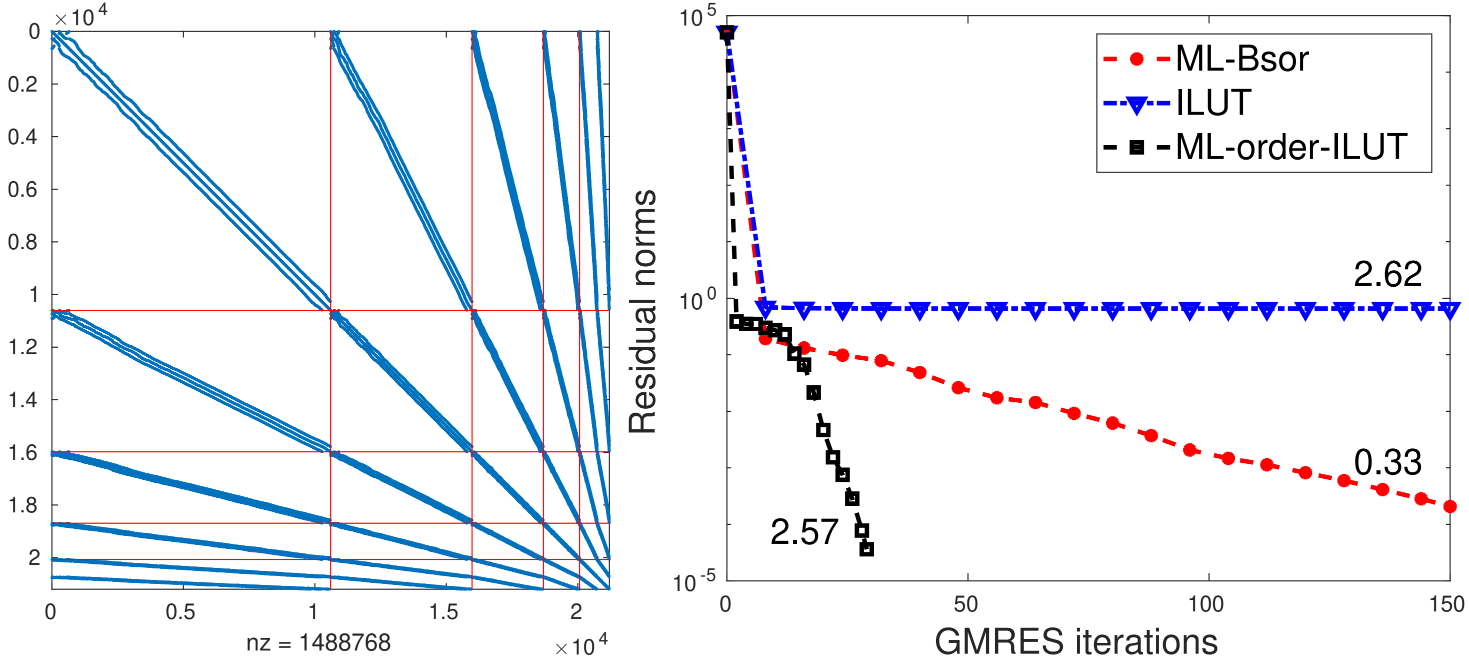
- Model: very rough approximation of Gaussian Elimination.

- Next: (Matlab) Test with matrix *Raefsky3* [1]

- 4 levels of coarsening. Then reorder matrix and:

- Solve with ILUT- GMRES(50) or BSOR - GMRES(50)

[1] SparseSuite collection. $n = 21,200, nnz \approx 1,5M$, Turbulence model.



Left: The matrix Raefsky3 after the reordering obtained from four levels of coarsening. Right: Performance of various coarsening based preconditioners for solving a linear system with the matrix.

COARSENING APPROACHES

Coarsening by matching: Pairwise aggregation

➤ Strategy: coalesce (collapse) two adjacent nodes in a graph into a single node, based on some measure of nearness or similarity.

➤ A *matching* of a graph $G = (V, E)$ is a set of edges $\widetilde{E}$, $\widetilde{E} \subseteq E$, such that no two edges in $\widetilde{E}$ have a node in common.

➤ Matching is maximal if it cant be augmented by additional edges

➤ Edge collapsing: usually selected using maximal matching

➤ Such edge matching techniques are common in AMG literature

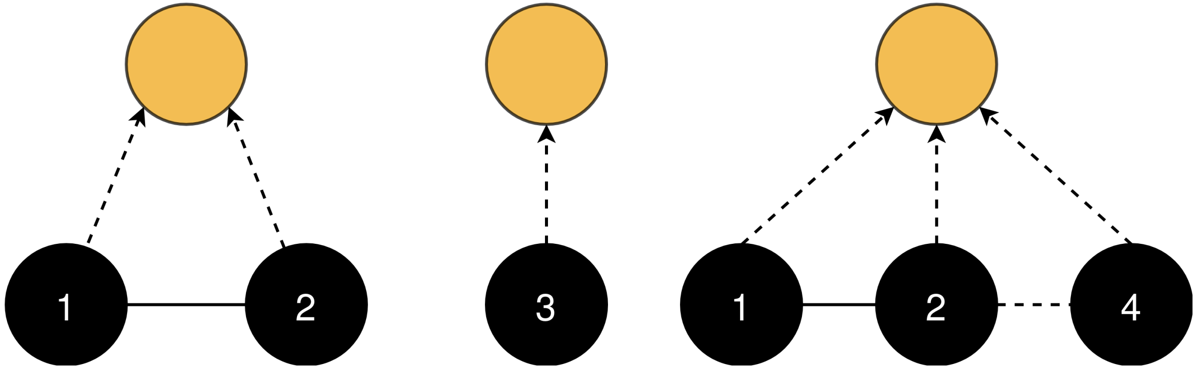
- For each node i, build a set S_i of nodes that are ‘strongly connected’ to i
- Traverse graph nodes in a certain order of preference
- Next unmarked node in this order, say j, selected as a *coarse* node.
- Priority measure of traversal updated after each insertion of a coarse node

Heavy-edge matching (HEM) : matches a node i with its largest off-diagonal neighbor j_{max};

$$\boxed{|a_{ij_{max}}| = \max_{j \in adj(i), j \neq i} |a_{ij}|}$$

- There will be left-over nodes - called ‘singletons’

Heavy Edge Matching (HEM)



1. Visit edges (i, j) in decreasing value of their weight $w_{i,j}$
2. If both i and j have no parents yet (left), create a new coarse node ('new'). Set parents of i and j to be new.
3. At completion of traversal: deal with unassigned nodes: Either (middle) add as a coarse nodes if disconnected ("singleton") or (right) lump as a child to an existing coarse node

ALGORITHM : 9. *Heavy Edge Matching (HEM)*

```
Input: Weighted graph $G = (V, E, A)$
Output: Coarse nodes; $Prnt$ list
Init: $Prnt(i) = 0\ \forall i \in V$; $new = 0$
for $\max$ to $\min$ edge $(i, j)$ do
    if $Prnt(i) == 0$, $Prnt(j) == 0$ then
        $new = new + 1$
        $Prnt(i) = Prnt(j) = new$
    end if
end for
for Node $v$ with $Prnt(v) == 0$ do
    if $v$ has no neighbor then
        $new = new + 1$; $Prnt(v) = new$
    else
        $Prnt(v) = Prnt(j)$ where $j = \text{argmax}_i(a_{iv})$
    end if
end for
```

Coarsening by independent sets

Recall: Independent set: $\mathcal{S} \subseteq V$ is a set of vertices that are not adjacent to each other: $\boxed{i, j \in \mathcal{S} \implies a_{ij} = 0}$. It is maximal if it can't be augmented

- Can take $V_c = \mathcal{S}$ as a coarse set. Need to define edges.

- Let L = reordered graph Laplacian (n_c vertices of V_c listed first): (note: D_c is *diagonal*)

$$L = \begin{pmatrix} D_c & -F \\ -F^T & B \end{pmatrix}$$

- Replace B by $D_f = F^T \mathbb{1}$ and define G_c = graph of $S_c \to$

$$S_c = D_c - F D_f^{-1} F^T$$

Property: S_c = Graph Laplacian of coarse graph G_c

Coarsening by ‘algebraic distance’

- Motivated by “bootstrap algebraic multigrid” (BAMG) [Brandt’01]
- In BAMG notion of closeness (used for coarsening) defined from a few steps of Gauss-Seidel for solving $\boldsymbol{Ax = 0}$
- Speed of convergence of the iterate determines an ‘algebraic distance’ between variables.
- Exploited to aggregate the unknowns and define restriction and interpolation operators. Analysis in [Chen-Safro’11]

Coarsening by 'kron' decomposition

- Kron reduction of networks proposed back in 1939 by Kron
- Revived by Dorfler and Bullo(2013) and Shuman et al. (2016)

Main idea:

- Select a coarse set V_1: Shuman-al use eigenvectors
- Reorder matrix so that nodes of V_1 come 1st.

Laplacean becomes →

$$L = \begin{bmatrix} L_{11} & L_{12} \\ L_{12}^T & L_{22} \end{bmatrix}$$

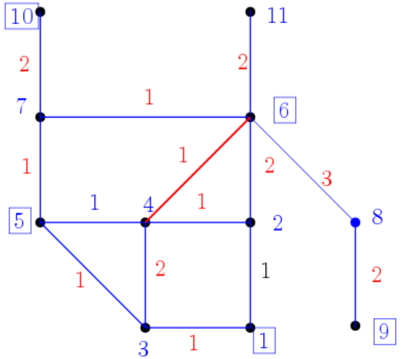
- Kron reduction of L defined as the

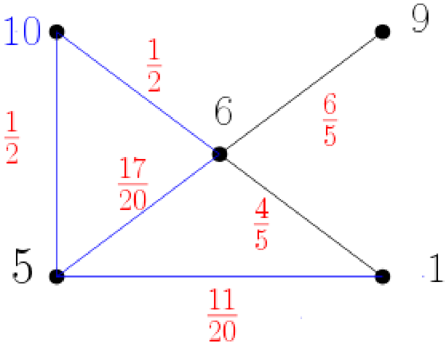
Schur complement:

$$L(V_1) = L_{11} - L_{12}L_{22}^{-1}L_{12}^T$$

Property $L(V_1)$ == graph Laplacian of V_1 [Dorfler-Bullo]

Example:





Two ways of using independent sets for coarsening.

Kron coarsening

Independent set coarsening

$$\left[\begin{array}{c|c} D_I & -F \\ \hline -F^T & B \end{array}\right] \longrightarrow \left[\begin{array}{c|c} D_I & -F \\ \hline -F^T & D_f \end{array}\right]$$

$$\downarrow \qquad\qquad\qquad\qquad \downarrow$$

$$L_c = D_I - FB^{-1}F^T \qquad\qquad L_c = D_I - FD_f^{-1}F^T$$

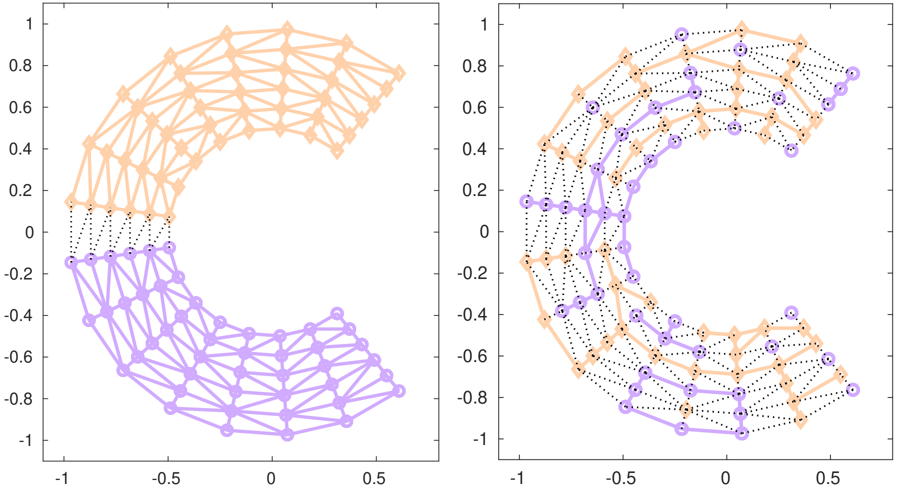
Q. 1: How to deal with 'denser' graph?

A Sparsify - using spectral sparsificaition

Q. 2: How to select V_1?

A Use signs of largest eigenvector of original Laplacian L

- If $u_1 = [\xi_1, \xi_2, \cdots, \xi_n]^T$ = the largest eigenvector.
- Define $V_+ = \{i | \xi_i \geq 0\}$ and $V_- = \{i | \xi_i < 0\}$
- Then select one of V_+, V_- as V_1.
- Opposite of what is done in spectral graph partitioning



Left side: spectral graph partitioning. Right: Coarsening withlargest eigenvector

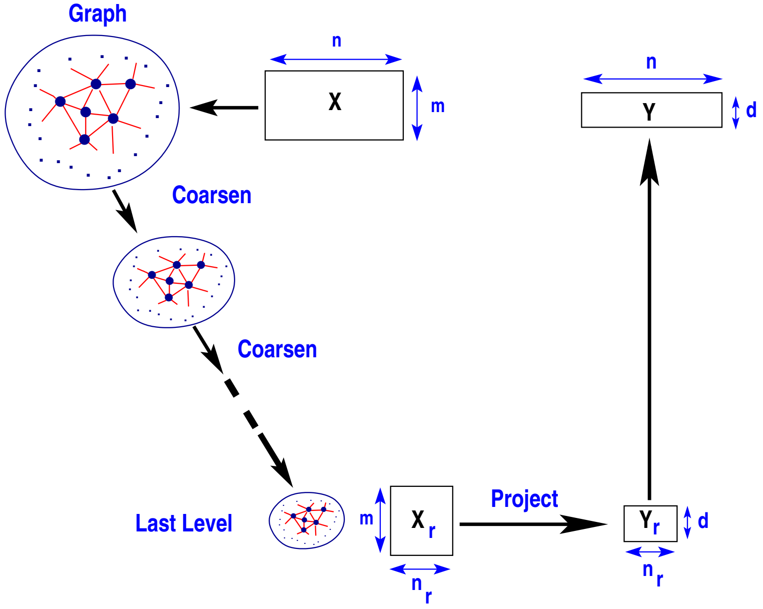
- Easy to show: (under mild condition on eigenvector) Each node of V_+ (resp. V_-) must have at least one nearest neighbor node from V_- (resp. V_+).

GRAPH COARSENING IN MACHINE LEARNING

Multilevel Dimension Reduction

Idea:

Coarsen for a few levels. Use resulting data set $\hat{X}$ to find a projector P from $\mathbb{R}^m$ to $\mathbb{R}^d$. Use this P to project data items.



- ➤ Gain: Dimension reduction is done with a much smaller set.
- ➤ Wish: not much loss compared to using whole data

Multilevel Dimension Reduction (for sparse data- e.g., text)

- Use Hypergraph Coarsening with *column matching* – similar to a common one used in graph partitioning

- Compute the non-zero inner product $\langle a^{(i)}, a^{(j)} \rangle$ between two vertices i and j, i.e., the ith and jth columns of A.

- Note: $\langle a^{(i)}, a^{(j)} \rangle = \|a^{(i)}\| \|a^{(j)}\| \cos \theta_{ij}$

Modif. 1: Parameter: $0 < \epsilon < 1$. Match columns i & j only if angle satisfies:

$$\tan \theta_{ij} \le \epsilon$$

Modif. 2: Re-Scale. If i and j match and $\|a^{(i)}\|_0 \ge \|a^{(j)}\|_0$ replace $a^{(i)}$ and $a^{(j)}$ by

$$c^{(\ell)} = \left(1 + \cos^2 \theta_{ij}\right)^{\frac{1}{2}} a^{(i)}$$

- Call C the coarsened matrix obtained from A using the approach just described

Lemma: Let $C \in \mathbb{R}^{m\times c}$ be the coarsened matrix of A obtained by one level of coarsening of $A \in \mathbb{R}^{m\times n}$, with columns $a^{(i)}$ and $a^{(j)}$ matched if $\tan\theta_i \le \epsilon$. Then

$$|x^T AA^T x - x^T CC^T x| \le 3\epsilon \|A\|_F^2,$$

for any $x \in \mathbb{R}^m$ with $\|x\|_2 = 1$.

- Very simple bound for Rayleigh quotients for any x.
- Implies some bounds on singular values and norms - skipped.
- See details + experiments in [Ubaru-YS '19]

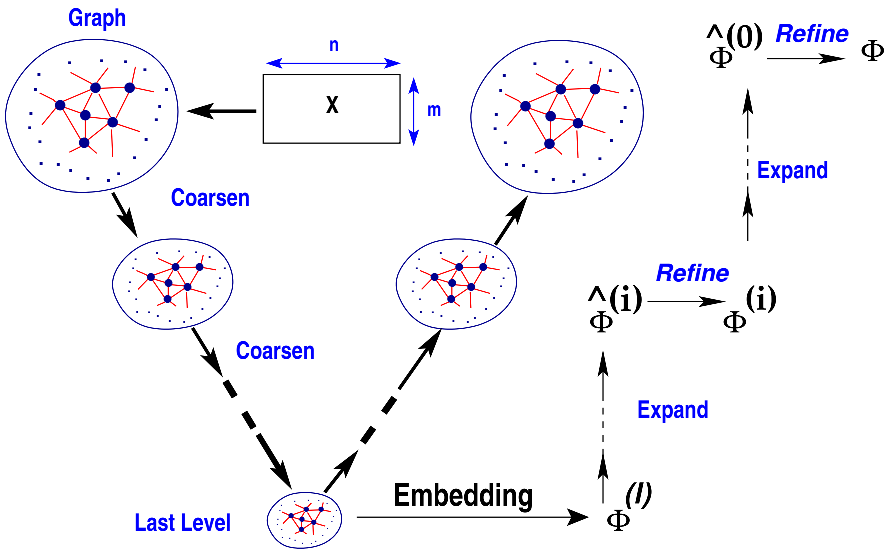
Graph coarsening for graph embeddings: HARP and MILE

- Recall Vertex embedding: Given $G = (V, E)$ find mapping Φ:

$$\boxed{\Phi : v \in V \longrightarrow \Phi(v) \ \in \ \mathbb{R}^d}$$

d is small: $d \ll n$

Hierarchical Representation Learning for Networks (HARP): (Chen et al. '18) coarsen for a few levels. Find embedding $\Phi^{(\ell)}$ for coarsest graph (level ℓ). Then a succession of expansions to higher level + refinement.



- Gain: Embedding done with a much smaller set.

- MILE approach [Liang et al. ’18] very similar (difference in refinement).

Experiment to evaluate the effectiveness of HARP.

- Baseline. Three embedding algorithms: *DeepWalk* [Perozzi-al’14], *LINE* [Tang-al’15] and *Node2vec* [Grover-Leskovec’16]

- Combined with Coarsening methods:

1. Heavy Edge Matching (HEM) - sketched earlier
2. Algebraic distance (ALG) - sketched earlier
3. Leverage Score Coarsening (LESC) – variant of HEM

Coarsening with eigenvectors

- It is possible to coarsen a graph with the goal of exactly preserving a few eigenvectors.
- This has turned out not to be too useful in practice.
- Instead we use eigenvectors to define ‘importance of nodes’ for the graph traversal

Leverage Scores

➤ $A = U \Sigma V^T$ (ran (A) = ran (U))

➤ Leverage score of i-th row →

$$\eta_i = \|U_{i,:}\|_2^2$$

● Used to measure importance of row i in random sampling methods [e.g. El-Aloui & Mahonney ’15]

- Let A now be a graph Laplacian and $A = U \Lambda U^T$ with $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$

In *Leverage-score coarsening (LESC)* we dampen lower sing. vectors →

$$\boxed{\eta_i = \sum_{k=1}^{r} (e^{-\tau\lambda_k} U_{ik})^2}$$

- Use η_i to decide order of traversal in coarsening algorithm

Note: Consider case when $r = n$ (or simply r is large)

$$\eta_i = \sum_{k=1}^{n} (e^{-\tau\lambda_k} U_{ik})^2 = \sum_{k=1}^{n} e^{-2\tau\lambda_k} |U_{ik}|^2 = e_i^T e^{-2\tau L} e_i.$$

➤ η_i equals the i-th diagonal entry of the matrix $H \equiv \exp(-2\tau L)$

- Next: visualization with 5 different coarsening methods on a graph with $n = 312$ nodes and $ne = 761$ edges

Final words

- ➤ *Many* interesting new matrix problems in areas that involve the effective exploitation of data

- ➤ Unlike in Forsythe's time: change happens fast - because we are better connected

- ➤ In particular: many many resources available online.

- ➤ Huge potential for making a good impact by looking at a topic from new perspective

- ➤ To a researcher in computational linear algebra : Tsunami of change on types or problems, algorithms, frameworks, culture,..

- My favorite quote. Alexander Graham Bell (1847-1922) said:

> *When one door closes, another opens; but we often look so long and so regretfully upon the closed door that we do not see the one which has opened for us.*

- Visit my web-site at *www.cs.umn.edu/~saad*

- More complete version of this material will available in course csci-8314 (S23) - notes (and more) are open to all.

Thank you !