

Lecture 14: Compressed sensing

Tuesday, October 13, 2015 9:30 AM

Outline: Haar wavelet basis & sparsity
Randomized dimension reduction
Compressed sensing:
Restricted isometry property (RIP)
 l_1 and l_2 norms, and sparsity
Image reconstruction

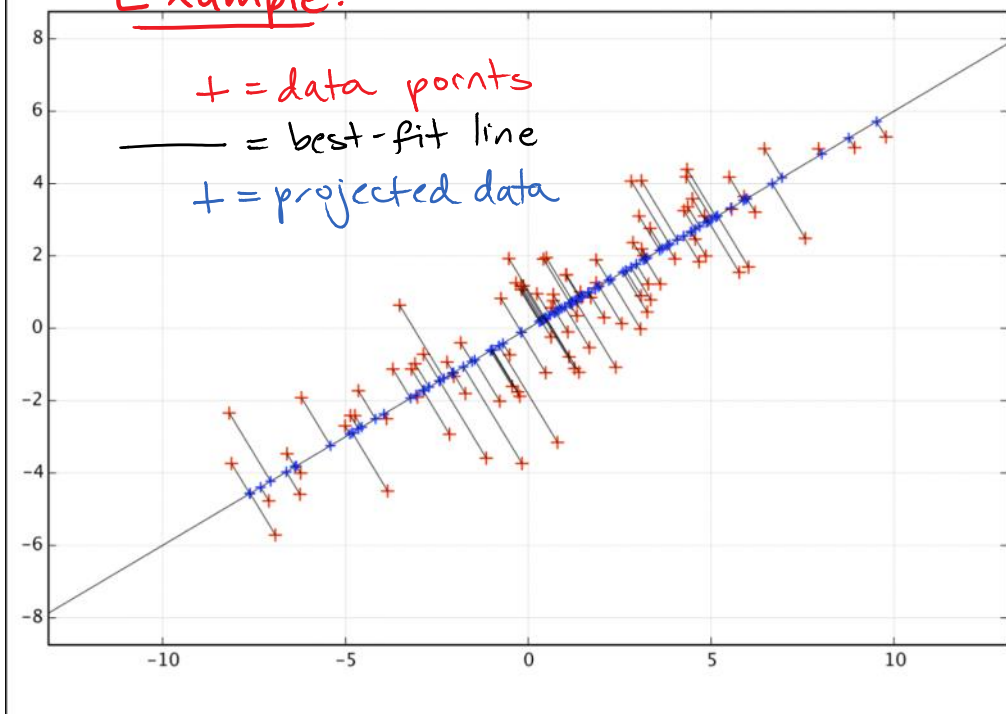
DIMENSION REDUCTION



TRADITIONAL APPROACH:

Structured data can be projected onto a lower-dimensional subspace. "Principal component analysis" [Pearson 1901]

Example:



1.2e. will go over this later!

We will go over this later!

- Problems:
1. Finding and projecting onto a good-fitting subspace is **SLOW!**
 2. A good, low-dimensional subspace might **not exist!**
 3. Even if the fit is decent, the data's **local properties are not preserved!**
(e.g., the **projections** of some pair of data points might be much closer than the original points)

MODERN APPROACH: **Randomized dimension reduction**

Ignore any structure — just apply a random linear function to the data

Example: Compute the projection of $e_1 = (1, 0, 0, \dots)$ onto a random 50-dimensional subspace of $\mathbb{R}^{100}$.

```
n = 100;  
d = 50;
```

```
v = zeros(n,1);  
v(1) = 1;
```

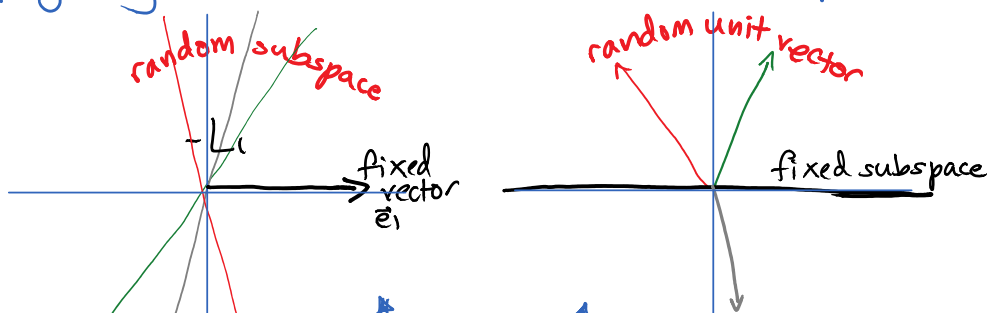
```
A = randn(n, d); % choose d random vectors in  $\mathbb{R}^n$   
[Q,R] = qr(A,0); % Gram-Schmidt  $\Rightarrow$  Q's columns are an  
                 % orthonormal basis for  $\mathcal{R}(A)$ 
```

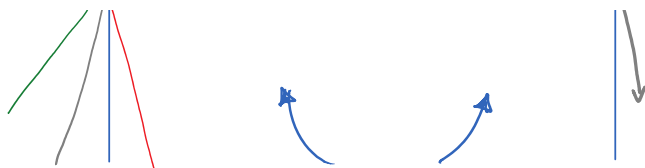
```
projectedv = Q * (Q' * v);
```

What is $\mathbb{E}[\| \text{projected } v \|^2]$?

Answer:

Projecting the fixed vector e_1 onto a random subspace
is equivalent to
projecting a random vector onto a fixed subspace.





the length of the projection
has the same distribution

$$\text{Squared length of random unit vector } \vec{v} = \sum_{i=1}^n |v_i|^2 = 1$$

$$\Rightarrow 1 = \mathbb{E}[\|\vec{v}\|^2] = \sum_{i=1}^n \mathbb{E}[|v_i|^2]$$

$$\Rightarrow \text{for all } i, \mathbb{E}[|v_i|^2] = \frac{1}{n} \quad (\text{by symmetry})$$

↑ expectation is linear

$$\Rightarrow \mathbb{E}[\|P_{\text{first } d \text{ coords}}(\vec{v})\|^2] = \mathbb{E}\left[\sum_{i=1}^d |v_i|^2\right] = \frac{d}{n}$$

$\Rightarrow$ After scaling projection up by $\sqrt{\frac{n}{d}}$, length "should be" almost unchanged.

Lemma: [Johnson-Lindenstrauss 1984]

Start with N data points, in $\mathbb{R}^n$.

Let V be a random d -dim. subspace, where

$$d = \frac{8}{\epsilon^2} \log N.$$

Then, with high probability, for any pair of points p, q ,

$$(1-\epsilon) \|p-q\| \leq \left\| \sqrt{\frac{n}{d}} P_V(p) - \sqrt{\frac{n}{d}} P_V(q) \right\| \leq (1+\epsilon) \|p-q\|.$$

* Distances between all data points are nearly preserved.

* The dimension need only be $O(\log(\# \text{ of data points}))$
— independent of ambient dimension n .

Logarithmic = huge reduction!!

Proof idea: "Concentration of measure"

$$\text{We know } \mathbb{E}[\|p'-q'\|^2] = \|p-q\|^2.$$

Show that the distribution is narrowly concentrated about its mean, and use a union bound over the $\binom{N}{2}$ pairs of data points...

about its mean, and use a union bound over the $\binom{2}{2}$ pairs of data points...

(Law of Large Numbers: Average of many independent, identically distributed random variables is tightly concentrated about the mean. But the $\|v_i\|^2$ are not independent...)

This is important for many, many applications....
→ very fast algorithms on large datasets.

Problem: Gram-Schmidt to find orthonormal basis for dimension- d subspace of $\mathbb{R}^n$ takes $\mathcal{O}(d^2 n)$ time.

Faster, easier way: Don't bother projecting
 $A = \frac{1}{\sqrt{d}} \text{randn}(d, n)$ scaling by $\frac{1}{\sqrt{d}}$ means each entry is $N(\text{mean } 0, \text{variance } \frac{1}{d})$
 $\Rightarrow \mathbb{E}[\text{squared length of a column}] = \text{total variance} = \frac{d}{d} = 1 \checkmark$

map data point $p \mapsto Ap$

Even faster: Multiply by a random matrix $\{-1, 0, 1\}$ entries

$$A = \frac{1}{\sqrt{d}} \text{randi}([-1, 1], [d, n]) \quad [\text{Achlioptas 2003}]$$

$p \mapsto Ap$ still preserves distances!

Test this in Matlab....

HAAR WAVELET BASIS

Haar orthonormal basis for $\mathbb{R}^8$:

$$\begin{aligned} & \frac{1}{\sqrt{8}}(1, 1, 1, 1, 1, 1, 1, 1) \\ & \frac{1}{\sqrt{8}}(1, 1, 1, 1, -1, -1, -1, -1) \\ & \frac{1}{2}(1, 1, -1, -1, 0, 0, 0, 0) \\ & \frac{1}{2}(1, -1, 0, 0, 0, 0, 0, 0) \\ & \frac{1}{2}(0, 0, 1, -1, 0, 0, 0, 0) \\ & \frac{1}{2}(0, 0, 0, 0, 1, -1, 0, 0) \\ & \frac{1}{2}(0, 0, 0, 0, 0, 0, 1, -1) \end{aligned}$$

Intuition: It comes from recursive downsampling:
 $(a, b, c, d, e, f, g, h) \in \mathbb{R}^8$

$$(a, b, c, d, e, f, g, h) \in \mathbb{R}^8$$

$$\rightarrow (a+b, c+d, e+f, g+h, a-b, c-d, e-f, g-h)$$

first $\frac{8}{2} = 4$ coordinates
come from binning coords

downsample
1st 4 coords.

two at a time

$$\rightarrow (a+b+c+d, e+f+g+h, a+b, e+f, \dots)$$

downsample
again

$$\rightarrow \begin{pmatrix} a+b+c+d & a+b+c+d & \vdots & \vdots \\ e+f+g+h & e+f+g+h & \vdots & \vdots \\ -a-b-c-d & -a-b-c-d & \vdots & \vdots \\ -e-f-g-h & -e-f-g-h & \vdots & \vdots \end{pmatrix}$$

DONE! Up to scaling, this gives the above basis vectors.

Haar wavelets for arrays $\mathbb{R}^{2^k \times 2^k}$

What if your vector gives the pixel values of a 2D image?

→ We should use a 2D version of the Haar basis.

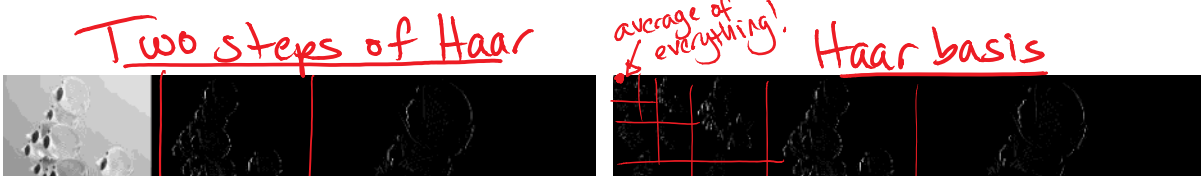
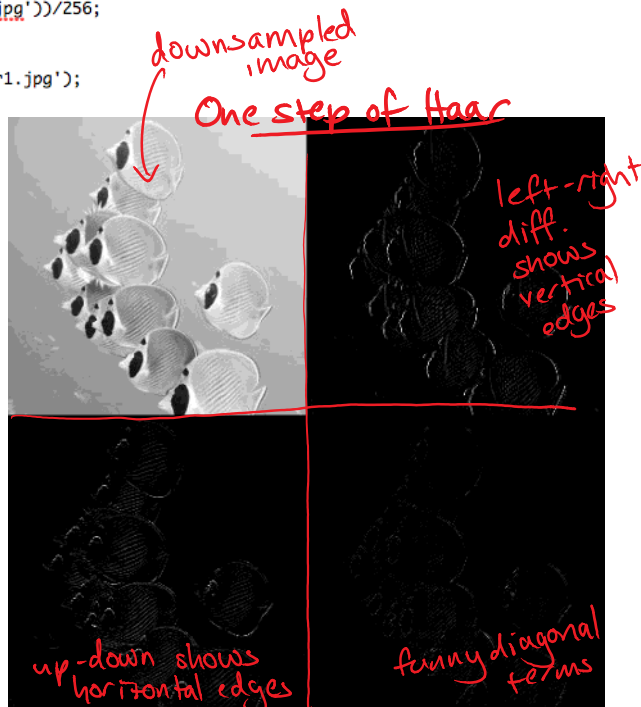
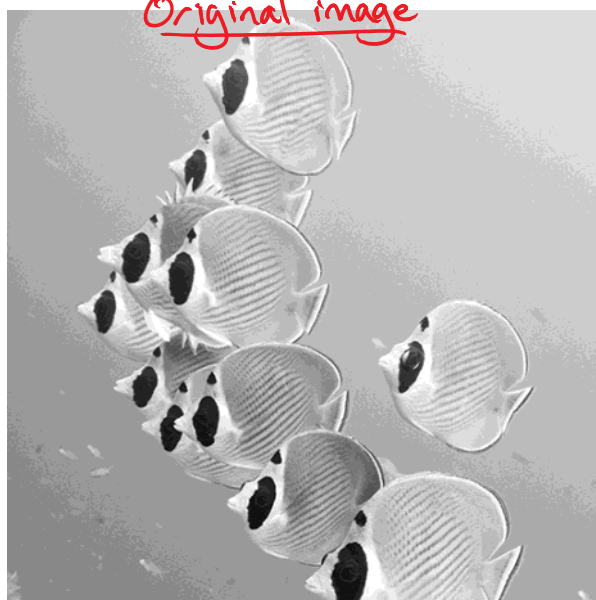
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \rightarrow \begin{pmatrix} \boxed{\text{average}} & \boxed{\text{left-right diff.}} \\ \boxed{\text{up-down difference}} & \boxed{\text{cross difference}} \end{pmatrix}$$

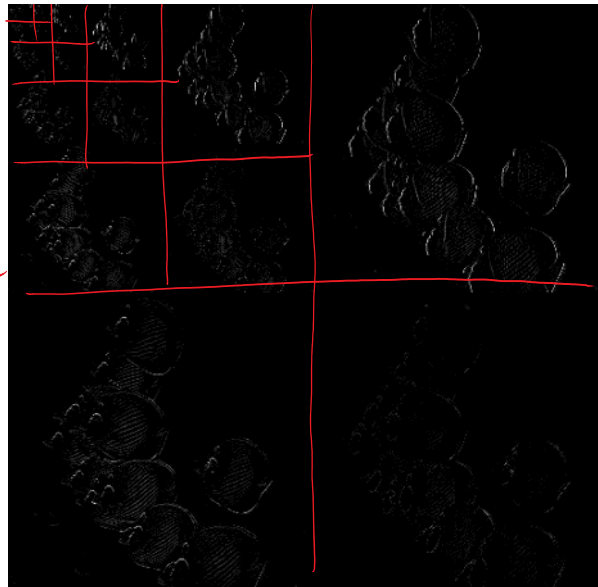
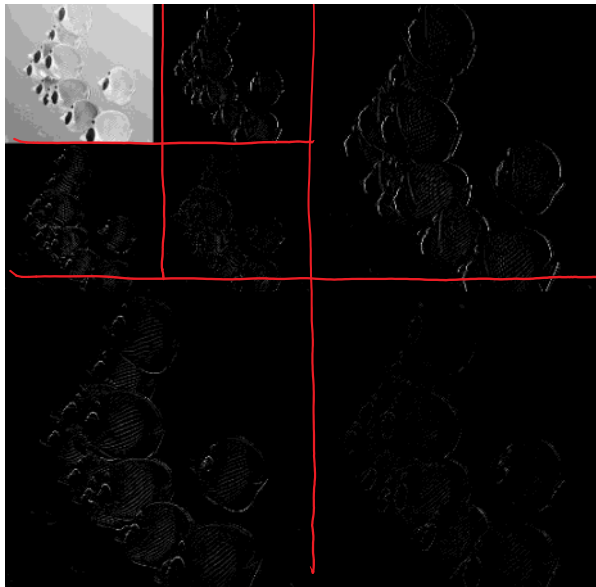
$$\begin{pmatrix} a+b+c+d & a+c-b-d \\ a+b-c-d & a-b-c+d \end{pmatrix}$$

& recurse in upper-left quadrant

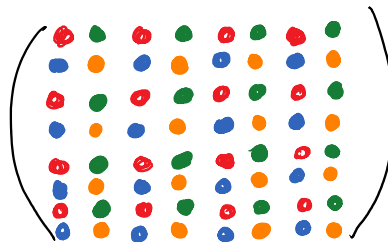
Example:

```
pkg load image;
P = double(imread('IMG_8474_g512.jpg'))/256;
PH1 = Haarstep(P, length(P));
imshow(PH1/2);
imwrite(PH1/2, 'IMG_8474_g512_Haar1.jpg');
```





```
function p = Haarstep(p, n)
    oddodd = p(1:2:n, 1:2:n);
    evenodd = p(2:2:n, 1:2:n);
    oddeven = p(1:2:n, 2:2:n);
    eveneven = p(2:2:n, 2:2:n);
    NW = (odddodd + evenodd + oddeven + eveneven)/2;
    NE = (odddodd + evenodd - oddeven - eveneven)/2;
    SW = (odddodd - evenodd + oddeven - eveneven)/2;
    SE = (odddodd - evenodd - oddeven + eveneven)/2;
    p(1:n/2, 1:n/2) = NW;
    p(1:n/2, n/2+1:n) = NE;
    p(n/2+1:n, 1:n/2) = SW;
    p(n/2+1:n, n/2+1:n) = SE;
end
```



```
function p = Haar(p)
    n = length(p);
    while (n > 1)
        p = Haarstep(p, n);
        n = n/2;
    end
end
```

Nice property: Real-world images tend to have fairly sparse representations in the Haar basis (i.e., most Haar coordinates are close to 0).

Examples:

Original



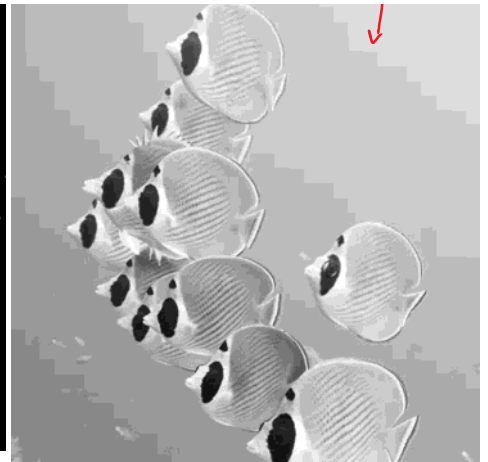
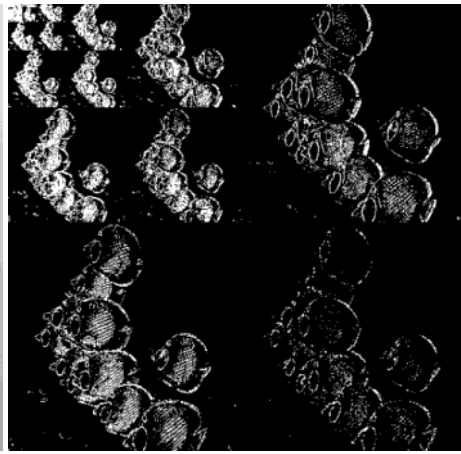
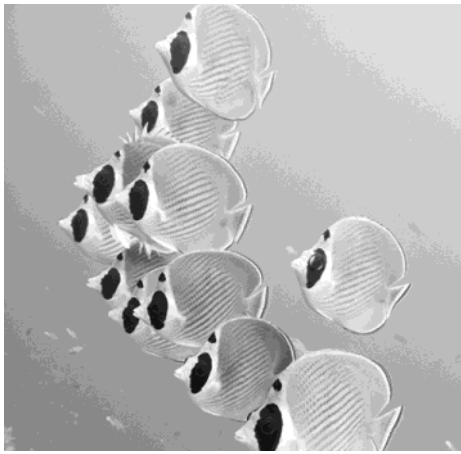
Keep 10% of coeffs



Result



notice the blocky gradients.
why?



Original

Keeping 10%

Keeping 20%





Why are gradients so blocky?

- Other bases perform better...

Code to change back from the Haar basis:

```
function p = undoHaarstep(p, n)
    NW = p(1:n/2, 1:n/2);
    NE = p(1:n/2, n/2+1:n);
    SW = p(n/2+1:n, 1:n/2);
    SE = p(n/2+1:n, n/2+1:n);
    oddodd = (NW + NE + SW + SE)/2;
    evenodd = (NW + NE - SW - SE)/2;
    oddeven = (NW - NE + SW - SE)/2;
    eveneven = (NW - NE - SW + SE)/2;
    p(1:2:n, 1:2:n) = oddodd;
    p(2:2:n, 1:2:n) = evenodd;
    p(1:2:n, 2:2:n) = oddeven;
    p(2:2:n, 2:2:n) = eveneven;
end
```

```
function p = undoHaar(p)
    n = 2;
    while (n <= length(p))
        p = undoHaarstep(p, n);
        n = n*2;
    end
end
```

Moral: Real-world signals are often nearly sparse in some known basis.

COMPRESSED SENSING

Model: True signal $\hat{x}$
Observed signal $b = A\hat{x}$
(for a known A)

Goal: Recover $\hat{x}$ by solving $Ax = b$.

Example: Camera sensor

$\hat{x}$ = image values at each pixel
 $A = I$ identity matrix
 $\Rightarrow$ you observe $\hat{x}$, done.

Different cases for solving $Ax = b$:

1. A $n \times n$ has rank n
 $\Rightarrow$ unique solution $x = A^{-1}b$
2. A $m \times n$ has rank n
 $\Rightarrow N(A) = \{0\}$ (since $\dim N(A) = n - \text{rank}(A)$)
 $\Rightarrow$ unique solution if $b \in R(A)$
3. $b \notin R(A)$
 $\Rightarrow$ no solution
 $\Rightarrow$ New goal: Find x to minimize $\|Ax - b\|$?
4. $b \in R(A)$ but $N(A) \neq \{0\}$
 $\Rightarrow \infty$ many solutions!
eg., if A $m \times n$ with $m < n$,
 $\text{rank}(A) \leq m \Rightarrow \dim N(A) = n - \text{rank}(A) \geq 1$
 $\Rightarrow$ New goal: Find "best" solution?

Compressed sensing:

A $m \times n$ with $m < n$ "
#observations < #degrees of freedom

Goal: Find sparsest solution

$$\begin{cases} \text{minimize } (\# \text{ of nonzero coords of } x) \\ \text{subject to } Ax = b \end{cases}$$

Unfortunately, this is NP-hard.

Even if we know there is a solution x with $\leq d$ nonzero coordinates, best approach is to try all $\binom{n}{d} = O(n^d)$ possible subsets.

$$\begin{matrix} n & & n \\ \left(\begin{matrix} A \end{matrix} \right) & \begin{pmatrix} x \end{pmatrix} & = & \begin{pmatrix} b \end{pmatrix} \\ m & & m \end{matrix}$$

$\downarrow$ if we guess x uses only first d coordinates

$$\begin{matrix} d & n & & & \\ \left(\begin{matrix} A \end{matrix} \right) & \begin{pmatrix} x \end{pmatrix} & = & \begin{pmatrix} b \end{pmatrix} \end{matrix}$$

▽ only first d coordinates

$$\begin{matrix} m & d & n & d & 1 \\ \left(\begin{array}{c|c} B & A \end{array} \right) & \left(\begin{array}{c} x \end{array} \right) & = & \left(\begin{array}{c} b \end{array} \right) \\ \text{keep these} & \text{loss these} & & \end{matrix}$$

solve $By = b$ (if $m \geq d$)

Key trick: l_1 norm minimization

Definition:

- The l_2 , or Euclidean, norm is given by

$$\|x\|_2 = \|x\| = \sqrt{\sum_i |x_i|^2}$$

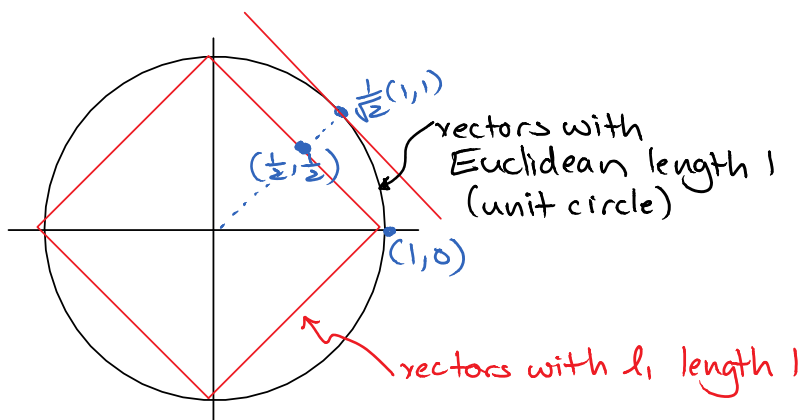
- The l_1 norm is given by

$$\|x\|_1 = \sum_i |x_i|$$

Example:

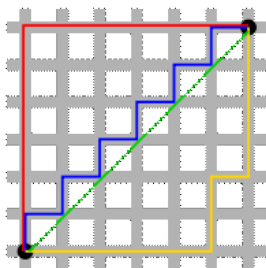
$$\|(1,0)\|_1 = 1 = \|(1,0)\|_2$$

$$\|(\frac{1}{2}, \frac{1}{2})\|_1 = 1 > \|(\frac{1}{2}, \frac{1}{2})\|_2 = \frac{1}{\sqrt{2}}$$



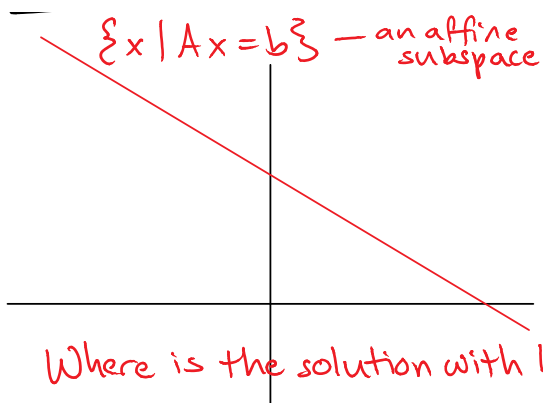
$$\|\frac{1}{\sqrt{2}}(1,1)\|_1 = \frac{1}{\sqrt{2}}\|(1,1)\|_1 = \sqrt{2}$$

"Taxicab geometry/Manhattan distance"

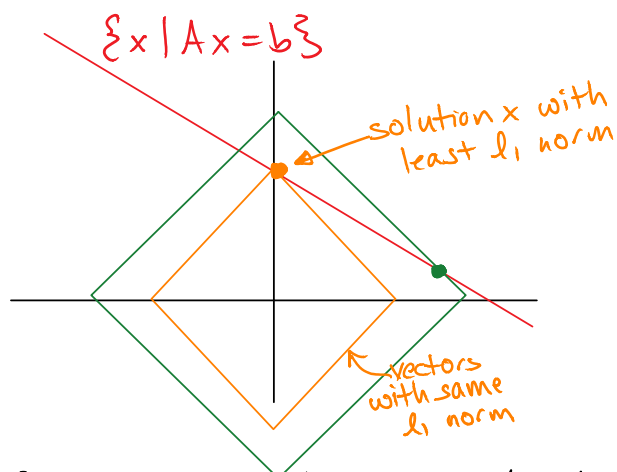


Question:

... ..



Answer:



Observe: The solution with least ℓ_1 norm is also the sparsest solution !!

(This is because the line of solutions doesn't have slope ± 1 .)

Intuition: The Euclidean norm treats

$$\frac{1}{\sqrt{n}}(1, 1, 1, \dots, 1), \quad \frac{1}{\sqrt{2}}(1, 1, 0, 0, \dots, 0)$$

$$\text{and } (0, 1, 0, 0, \dots, 0)$$

all the same; they all have length one.

But the ℓ_1 norm favors sparsity

$$\frac{1}{\sqrt{n}} \cdot n = \sqrt{n}$$

$$\frac{1}{\sqrt{2}} \cdot 2 = \sqrt{2}$$

1 in ℓ_1 norm

And there are efficient algorithms for solving

$$\begin{cases} \text{minimize } \|x\|_1, \\ \text{subject to } Ax = b \end{cases}$$

Examples: (all these are Matlab-only, not Octave)



CVX: Matlab Software for Disciplined Convex Programming
Version 2.0 (beta), September 2013, Build 1010

CVX is a Matlab-based modeling system for convex optimization. CVX turns Matlab into a modeling language, allowing constraints and objectives to be specified using standard Matlab expression syntax. For example, consider the following convex optimization model:

$$\begin{aligned} & \text{minimize} && \|Ax - b\|_2 \\ & \text{subject to} && Cx = d \\ & && \|x\|_\infty \leq \epsilon \end{aligned}$$

The following code segment generates and solves a random instance of this model:

```
m = 20; n = 10; p = 4;
A = randn(m,n); b = randn(m,1);
C = randn(p,n); d = randn(p,1); e = rand;
cvx_begin
    variable x(n)
    minimize( norm( A * x - b, 2 ) )
    subject to
        C * x == d
        norm( x, Inf ) <= e
cvx_end
```

General-purpose convex optimization

ℓ_1 -MAGIC

[code](#) / [papers](#) / [links](#)

One of the central tenets of signal processing is the Shannon/Nyquist sampling theory: the number of samples needed to capture a signal is dictated by its bandwidth. Very recently, an alternative theory of "compressive sampling" has emerged. By using nonlinear recovery algorithms (based on convex optimization), super-resolved signals and images can be reconstructed from what appears to be highly incomplete data. Compressive sampling shows us how data compression can be implicitly incorporated into the data acquisition process, a gives us a new vantage point for a diverse set of applications including accelerated tomographic imaging, analog-to-digital conversion, and digital photography.

See examples of compressive sampling in action.

Code

L1-MAGIC is a collection of MATLAB routines for solving the convex optimization programs central to compressive sampling. The algorithms are based on standard interior-point methods, and are suitable for large-scale problems.

Download the code (including User's Guide)

Download the User's Guide (pdf)

top




SparseLab
SEEKING SPARSE SOLUTIONS TO LINEAR SYSTEMS OF EQUATIONS

Inside SparseLab

- Home
- Download
- Documentation
- Papers, Demos
- Examples
- For Contributors
- Acknowledgements
- License

Other Packages

- WaveLab
- BeamLab
- SymLab

SparseLab is a Matlab software package designed to find sparse solutions to systems of linear equations, particularly underdetermined systems.

See the tabs on the left for an Introduction to SparseLab, and for download instructions. Please note a new version, SparseLab 2.0 has been released May 26, 2007. It is available for download on this website.

SparseLab is a library of Matlab routines for finding sparse solutions to underdetermined systems. It not only aims to provide tools for sparse representation in a cohesive package to the research community, it also allows researchers in this area to publicly release the code accompanying their published papers. The library is available free of charge over the Internet.

SparseLab has over 200 matlab files which are documented, indexed and cross-referenced in various ways. SparseLab makes available, in one package, all the code to reproduce all the figures in the included published articles. The interested reader can inspect the source code to see exactly what algorithms were used, and how parameters were set in producing our figures, and can then modify the source to produce variations on our results. SparseLab has been



for Analysis and Algebra



Group Optimization

SPEAR – Sparse Exact and Approximate Recovery

This research project deals with the problem to recover a sparse solution of an underdetermined linear (equality) system. This topic has many applications and is a very active research area. It is located at the border between analysis and combinatorial optimization. The main goal of our project is to obtain a better understanding of the conditions under which (efficiently) finding such a sparse solution, i.e., recovery, is possible. Our project is characterized by both theoretical and computational aspects as well as the interplay of continuous and discrete methods.

The SPEAR project is a collaboration of the Research Group Optimization (RGO) at the TU Darmstadt and the Institute for Analysis and Algebra (IAA) at the TU Braunschweig. The project is funded by a DFG research grant. Project period: 2011 – 2014.

Project members:

- Dirk A. Lorenz (IAA)
- Marc E. Pfetsch (RGO)
- Andreas M. Tillmann (RGO)
- Christian Kruschel (IAA)



YALL1: Your ALgorithms for L1

Welcome

YALL1 package now includes:

- YALL1 Basic, a solver for sparse reconstruction. Version 1.4, July 15, 2011.
- YALL1 Group, a solver for group/patch sparse reconstruction. Version 1.2, June 28, 2011. WU

YALL1 Basic

solves the following L1-minimization problems:

```
(BP) min ||w||_1, s.t. Aw = b
(L1/L2) min ||w||_1 + (1/2) ||Ax - b||_2^2
(L1/L2) min ||w||_1 + (1/2) ||Ax - b||_2^2
(L1/L2/beam) min ||w||_1, s.t. ||Ax - b||_2 <= B
(BP+) min ||x||_1, s.t. Ax = b, x >= 0
(L1/L2+) min ||x||_1 + (1/2) ||Ax - b||_2^2, s.t. x >= 0
(L1/L2+) min ||x||_1 + (1/2) ||Ax - b||_2^2, s.t. x >= 0
(L1/L2/beam+) min ||x||_1, s.t. ||Ax - b||_2 <= B, x >= 0
```

Pages

- Welcome
- News
- Acknowledgment
- Manuals and Modules
- Demo Set
- Random Tiling / Circulant / Convolution

Links

- Download
- Feedback / Discussions
- Some L1-Related Solvers

SPGL1: A solver for large-scale sparse reconstruction

Version 1.6, May 2013

home
download and install
examples (basic)
examples (advanced)
supplement
release notes
citing spgl1

SPGL1 is a Matlab solver for large-scale one-norm regularized least squares. It is designed to solve any of the following three problems:

basis pursuit denoise

minimize $\|x\|_1$ subject to $\|Ax - b\|_2 \leq \epsilon$

basis pursuit

minimize $\|x\|_1$ subject to $Ax = b$

Lasso

minimize $\|Ax - b\|_2$ subject to $\|x\|_1 \leq \tau$

SPGL1 relies only on matrix-vector operations Ax and $A^T y$ and accepts both explicit matrices and functions that evaluate these products. SPGL1 is suitable for problems that are in the complex domain. The theory underlying SPGL1 is described in these papers:

- Probing the Pareto frontier for basis pursuit solutions E. van den Berg and M. P. Friedlander, SIAM J. on Scientific Computing, 30(2):890-912, November 2008 (pdf)
- Sparse optimization with least-squares constraints E. van den Berg and M. P. Friedlander, SIAM J. on Optimization, 21(4):1201-1229, 2011 (pdf)

Group sparsity

In version 1.6, the core SPGL1 routine was generalized to solve the above three problems with $\|x\|_1$ replaced by any norm on x . In order to do so, it requires that routines are provided for computing the

1. norm $\|x\|_1$
2. corresponding dual norm $\|x\|_*$
3. Euclidean projection

and many more...

Other applications of the minimizing $\ell_1 \rightarrow$ sparsity trick include deep neural networks in machine learning...

EXAMPLE:

Original image:
 "Shepp-Logan phantom"
 (supposed to be a brain)
 64 × 64 pixels



or $P = \text{phantom}(\text{"Modified-Shepp-Logan", 64})$
 ✓

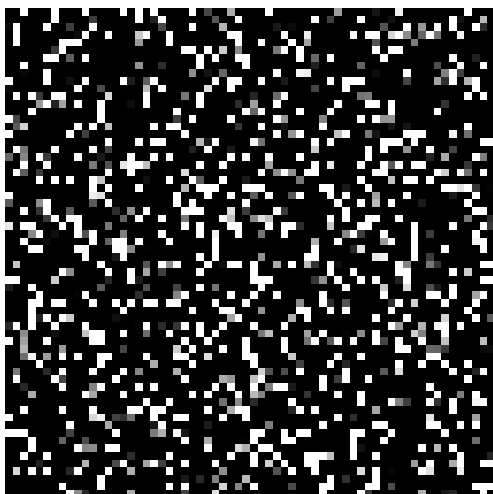
```
P = double(imread('Shepp-Logan64.png'))/256;  
n = length(P); = 64  
N = n * n; = 4096
```

Measurement model:

```
d = n * log2(n) * 5; = 1920 = 0.47 * N  
A = 1/sqrt(d) * randn(d, N);  
b = A * P(:);
```

inner product with d random vectors
 (this is not realistic)

① Matlab's solve routine



```
x = A\b;  
Pout = reshape(x, n, n);  
imshow(Pout);  
imwrite(Pout, 'Pout.png');
```

System is completely undetermined:
1920 equations in 4096 unknowns

② Solution with least ℓ_2 norm

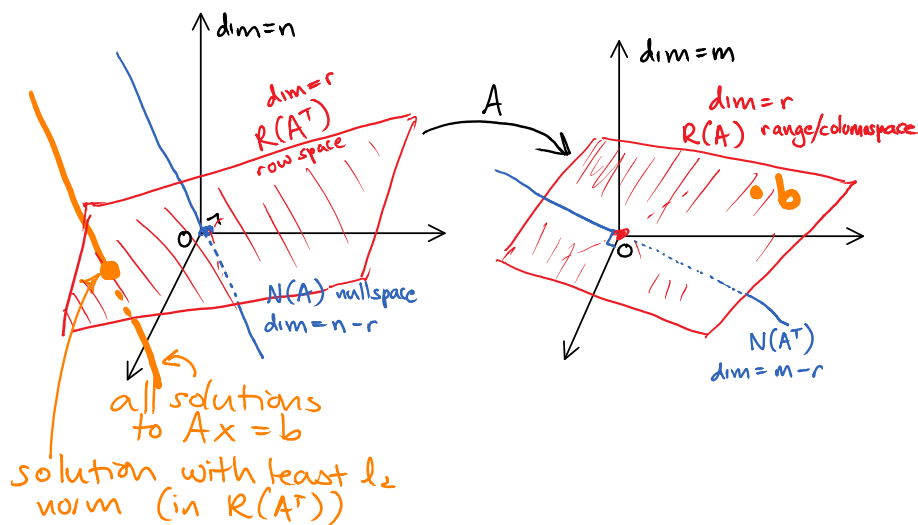


brain?

```
cvx_begin
    variable x(N);
    minimize( norm(x, 2) )
    subject to
        A*x == b
cvx_end
```

takes ~2 minutes
using CVX
there are better ways!!

```
Plsq = reshape(x, n, n);
imshow(Plsq);
```



③ Look for a sparse solution

Move to the Haar basis:

```

% put rows of A into the Haar basis
AHaar = A;
for i = 1:d
    reshapedrow = reshape(AHaar(i,:), n, n);
    reshapedrow = Haar(reshapedrow);
    AHaar(i,:) = reshapedrow(:);
end

sparsity = numnonzeroentries(Haar(P)); 725 (this is cheating...)
subsets = combnk(1:N, sparsity); % don't run this code!
for i = 1:size(subsets,1)
    % try to find a solution using those columns only
    % if successful, then stop
    AHaarrestricted = AHaar(:,subsets(i));
    x = AHaarrestricted\b;
    if (sum(sum(abs(AHaarrestricted*x-b))) < 10^-5)
        break;
    end;
end

```

Note: This takes $\binom{64}{725} = 2^{2750}$ time steps.

This is more than

$$\begin{aligned}
 & \underbrace{(\# \text{ atoms in observable universe})}_{2^{270}} \cdot \underbrace{(\# \text{ of Planck time steps since Big Bang})}_{2^{207}} \\
 & \cdot \underbrace{(\# \text{ books in Library of Congress})}_{2^{24}}^{94}.
 \end{aligned}$$

④ Solution with least ℓ_1 norm



```

cvx_begin
    variable x(N);
    minimize( norm(x, 1) )
    subject to
        AHaar*x == b
cvx_end

```

} takes ~20 minutes

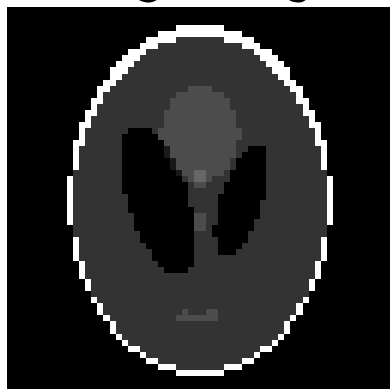
```

Pl1 = reshape(x, n, n);
Pl1 = undoHaar(Pl1);
imshow(Pl1);
imwrite(Pl1, 'Pl1.png');

```

Summary of results:

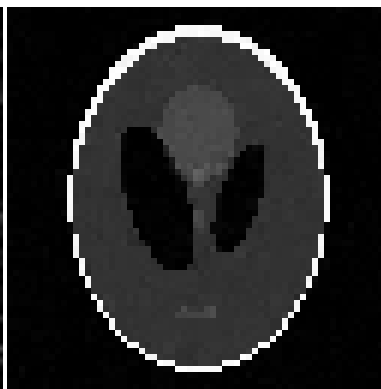
Original image



Least ℓ_2 norm



Least ℓ_1 norm



ℓ_1 error = $0.1 * N$

ℓ_1 error $0.007 * N$

Haar: $725^{=.17N}$ nonzero
278 $\ell_1 = .06N$

Haar: 3943 nonzero
342 ℓ_1

Haar: 2199 nonzero
275 ℓ_1 !

Running ℓ_1 magic

```

tic
xp = l1eq_pd(x0, AHaar, [], b, 1e-3);
toc

```

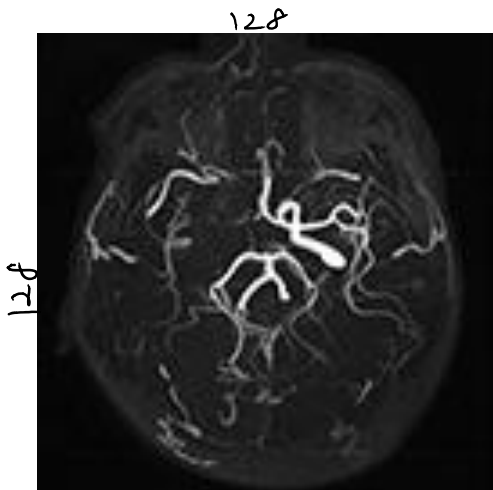
} only 50 seconds



ℓ_1 error = $0.8 = 0.0002 N$

Haar: 734 nonzero
277 ℓ_1

EXAMPLE 2:



of variables
 $N = 128^2 = 16384$

of observations
 $d = 4480 = 0.27N$

Haar: $\sim 0.87N$ nonzero
.05N l_1

~ 15 minutes

↓
15 minutes
in l_1 magic



l_1 error = .046N
Haar: 0.4N nonzero
.03N l_1

How to do better:

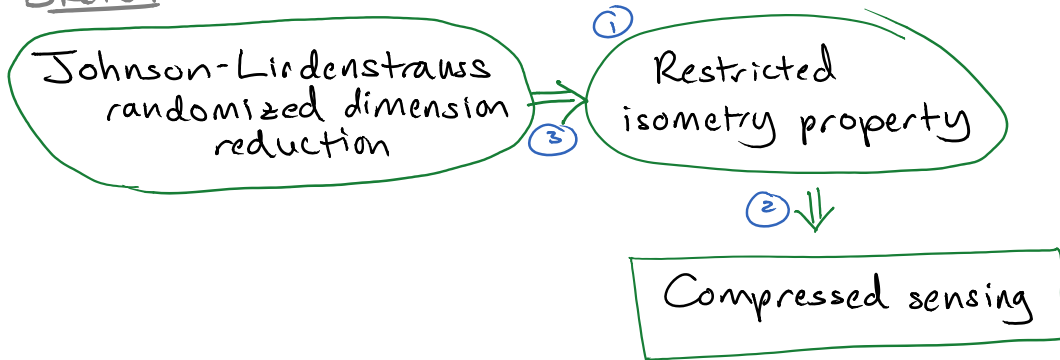
- ① More equations
- ② Better wavelet basis
- ③ Smarter optimization algorithms
(see references)

Why does compressed sensing work?

1. Signals are sparse, or "nearly sparse," in a known basis.
2. Dimension reduction

DIMENSION - REDUCTION & COMPRESSED SENSING

Sketch:



①

Definition: "Restricted Isometry Property" (RIP)

Roughly: A matrix is RIP if multiplying by that matrix preserves the lengths of all sparse vectors.

Precisely: The $d \times n$ matrix A is (s, ϵ) -RIP if

for all $\bar{x} \neq 0$ with $\leq s$ nonzero coordinates,

$$\left| \frac{\|A\bar{x}\|}{\|\bar{x}\|} - 1 \right| \leq \epsilon.$$

Example: Obviously, the $n \times n$ identity I is RIP; it preserves all lengths exactly. But there are also many nontrivial examples, including most random projections, most samples of Fourier coefficients.

Example: If A , $d \times n$ with $d < n$, is very sparse — fewer than n nonzero entries — then it cannot be RIP.

because then one column of A would be 0: $\begin{pmatrix} x & & & x \\ & x & & \\ & & \boxed{\begin{matrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{matrix}} & \\ & x & & x \end{pmatrix}$

$\Rightarrow$ Intuition: Rows of RIP matrices should be "diffuse," not sparse.

Note: Of course, RIP depends on the basis for sparsity. Sampling Fourier coefficients will not be RIP if the

source has a sparse Fourier representation.

②

Restricted isometry property



Compressed sensing

Theorem: Let A be a $(2s, \epsilon)$ -RIP matrix, with $\epsilon < 1$.

Let x have $\leq s$ non-zero coeffs, and $b = Ax$.

Let

$\tilde{x}$ = the sparsest solution to $Ay = b$.

Then $\tilde{x} = x$ — sparse recovery works.

Proof: Assume $\tilde{x} \neq x$. Then $x - \tilde{x}$ has $\leq 2s$ nonzero coeffs,

$$A(x - \tilde{x}) = 0.$$

$$\Rightarrow \left| \frac{\|A(x - \tilde{x})\|}{\|x - \tilde{x}\|} - 1 \right| = 1 > \epsilon, \text{ a contradiction. } \square$$

③

Johnson-Lindenstrauss randomized dimension reduction



Restricted isometry property

J-L lemma: N^2 vectors $\rightarrow O(\log N)$ dimensions
w/o much changing any vectors' length.

For RIP we want all s -sparse vectors to have lengths preserved. There are

$$\binom{n}{s} = O(n^s)$$

coordinate subsets of size s , but infinitely many s -sparse vectors.

Solution:

① If A preserves the length of x , then it also preserves the length of $2x$, $10x$ — any multiple of x .

$$\|A \cdot 2x\| = \|2Ax\| = |2| \cdot \|Ax\|$$

⇒ It is enough that A preserve lengths for all x with $\|x\| = 1$.

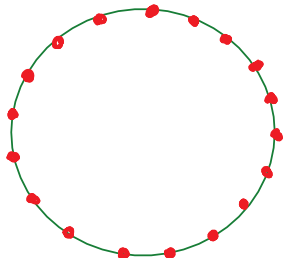
⇒ We only need to check that A works on the

$\binom{n}{3}$ spheres in $\mathbb{R}^d$.

$\sim$ still ∞ many points

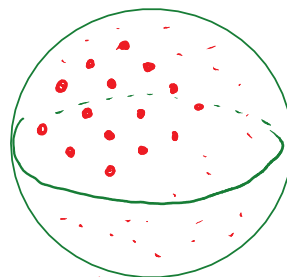
② Approximate the sphere by a finite set

$\mathbb{R}^2$:



all points on the circle are within ε of one of the $\frac{2\pi}{2\varepsilon}$ red points

$\mathbb{R}^3$:



$O(\frac{1}{\varepsilon^2})$ points in net

Fact: $\mathbb{R}^s$: $(\frac{3}{\varepsilon})^{s-1}$ red points are enough to be within ε of any point on the sphere

Claim: If the scaled projection A $(1 \pm \varepsilon)$ -preserves the lengths of all points in the net, then it $(1 \pm 3\varepsilon)$ -preserves lengths for all points in the sphere.

(Intuitively obvious.)

Proof: We are given $\left| \frac{\|Ax\|}{\|x\|} - 1 \right| \leq \varepsilon$ for all red points in the net.

Let y be some point in the sphere; it is within ε of some net point x . So $\|y\| = \|x\| = 1$ and $\|y - x\| \leq \varepsilon$.

$$\begin{aligned} \Rightarrow \|Ay\| &= \|Ax + A(y-x)\| \\ &\leq \|Ax\| + \|A(y-x)\| \\ &\leq \overset{1+\varepsilon}{\|Ax\|} + \overset{\sqrt{\frac{n}{2}}}{\|A(y-x)\|} \end{aligned}$$

since $A = \sqrt{\frac{n}{2}}$ times a projection, and projections can't increase lengths

$$\leq 1 + \left(\sqrt{\frac{n}{2}} + 1\right) \varepsilon.$$

This is no good! We want $d \ll n$, $\varepsilon \sim$ constant.

Better approach: For the worst y , let $\delta = \|Ay\| - 1$.

$$\Rightarrow \|Ay\| = 1 + \delta \leq (1 + \varepsilon) + (1 + \delta) \cdot \varepsilon$$

$$\Rightarrow \delta(1 - \varepsilon) \leq 2\varepsilon$$

$$\Rightarrow f \leq \frac{2\varepsilon}{1-\varepsilon} \leq 3\varepsilon \text{ for } \varepsilon \leq \frac{1}{3}.$$

To lower-bound $\|Ay\|$, use

$$\|Ay\| \geq \|Ax\| - \|A(y-x)\|$$

$$\geq \underbrace{\|Ax\|}_{1-\varepsilon} - \underbrace{\|A(y-x)\|}_{(1+f)\varepsilon}$$

$$\geq 1-\varepsilon - (1+f)\varepsilon$$

$$\geq 1-3\varepsilon \text{ for } \varepsilon \leq \frac{1}{3}. \quad \square$$

$\Rightarrow$ total # of vectors whose lengths we need to preserve

$$\leq \binom{n}{s} \times \left(\frac{3}{\varepsilon}\right)^{s-1} = N$$

$\stackrel{J-L}{\Rightarrow} d = O\left(\frac{\log N}{\varepsilon^2}\right)$ dimensions are enough

$$O\left(\frac{1}{\varepsilon^2} \log\left(\frac{n}{\varepsilon}\right)^s\right)$$

$$d = O\left(\underbrace{s}_{\substack{\text{sparsity} \\ = \# \text{ coeffs we} \\ \text{must recover}}} \cdot \underbrace{\log\left(\frac{n}{\varepsilon}\right)}_{\substack{\text{log-dependence} \\ \text{on dimension}}} \cdot \underbrace{\frac{1}{\varepsilon^2}}_{\substack{\text{fudge} \\ \text{factor}}}\right)$$

Conclusion: If A is $\sqrt{\frac{n}{d}}$ times the projection onto a random d -dimensional subspace, with d as above, then with high probability A is $(s, 3\varepsilon)$ -RIP.

