Intro to Frames, Dictionaries and K-SVD

Jim Van Verth
Software Engineer, Google
Manny Ko
Principal Engineer, Activision R&D

GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21

Overview

- Our goal:
  - represent data in sparsest way possible
  - good for compression

Overview

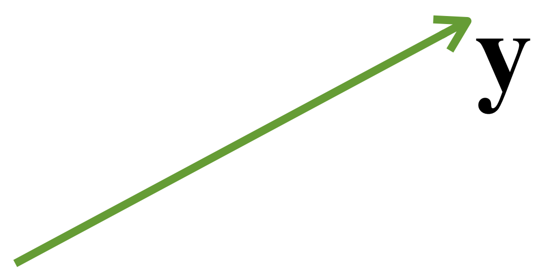
- Part of a series
  - Green & Ko, “Frames, Sparsity and New Math for Games”
  - Green & Ko, “Orthogonal Matching Pursuit and K-SVD for Sparse Encoding”
  - Ko, “Dictionary Learning in Games”

Overview

- Basis vectors
- Frames
- K-SVD

Basis vectors

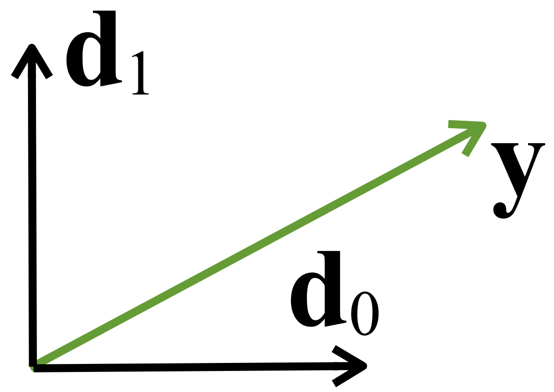
- Have vector



What's a compact way of representing this?

Basis vectors

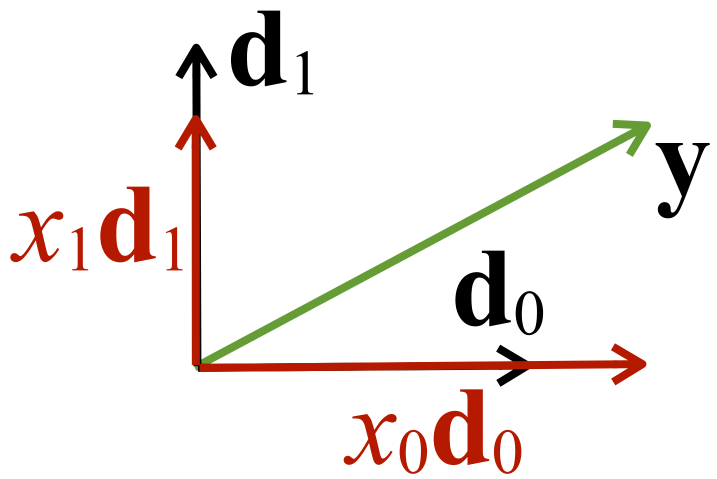
- Can represent as linear combination



Let's fix two vectors in space and define the vector relative to them. I'll talk in a bit how to pick these vectors.

Basis vectors

- Can represent as linear combination

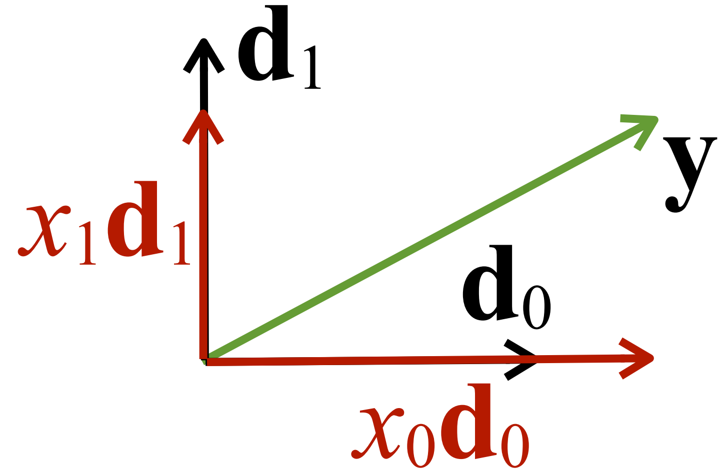


$$\mathbf{y} = x_0\mathbf{d}_0 + x_1\mathbf{d}_1$$

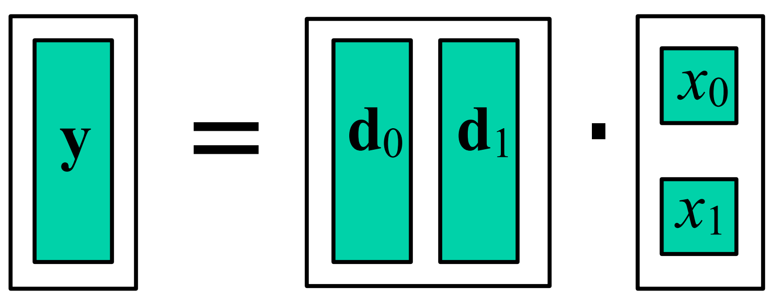
A linear combination is just a weighted sum of vectors.

Basis vectors

- Can represent as linear combination



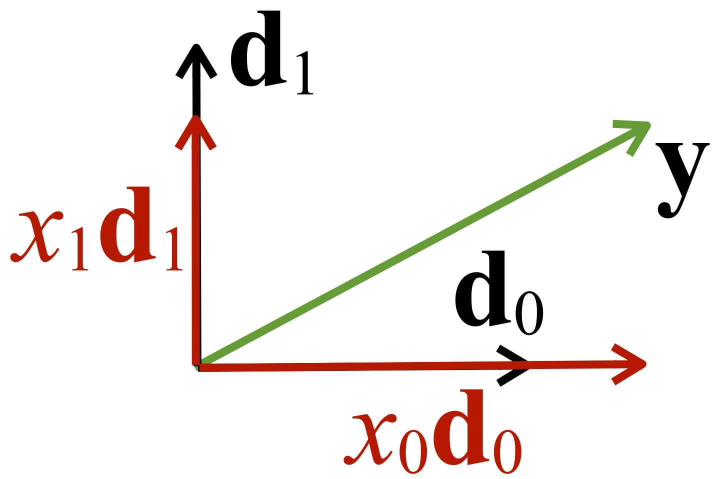
- Alternatively



Alternatively we can represent this as a matrix multiplication, where the matrix has d0 and d1 as column vectors, we multiply by the vector x0, x1, and get y. We'll see this again in another form later.

Basis vectors

- Can represent as linear combination



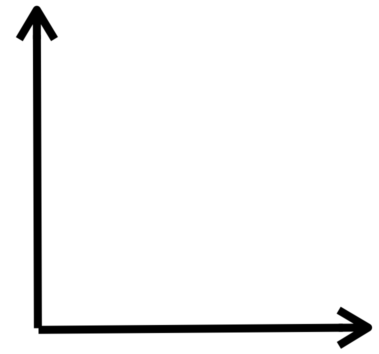
- Alternatively

$$\mathbf{y} = \mathbf{D}\mathbf{x}$$

Or in matrix notation:

Basis vectors

- If:
  - can represent any vector with set (spans)
  - non-redundantly (linearly independent)

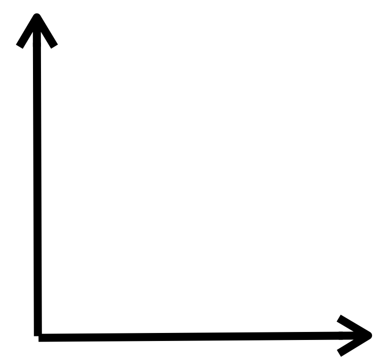


- called a *basis*.

A basis has two main properties: you have to be able to create any vector in your space with a linear combination (called spanning the space), and there can't be any redundant vectors (called linearly independent). The end result is that there is one and only one way to represent a vector using a linear combination of the basis vectors.

Basis vectors

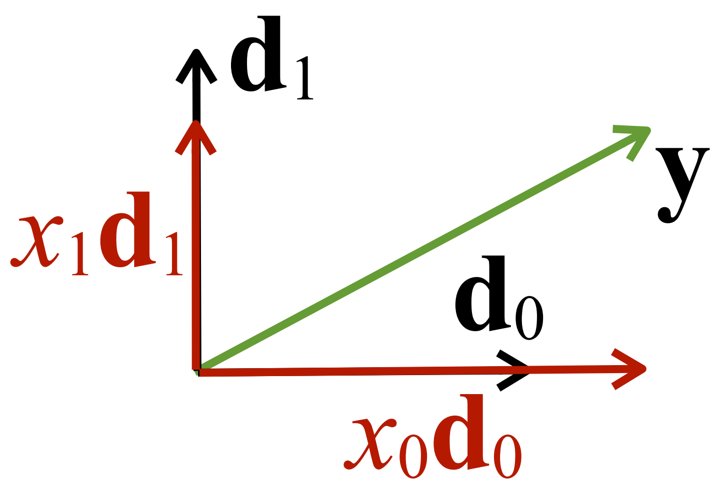
- If:
  - can represent any vector with set (spans)
  - non-redundantly (linearly independent)



- called a *basis*. If unit length & orthogonal, *orthonormal basis (ONB)*

Basis vectors

- Given a basis

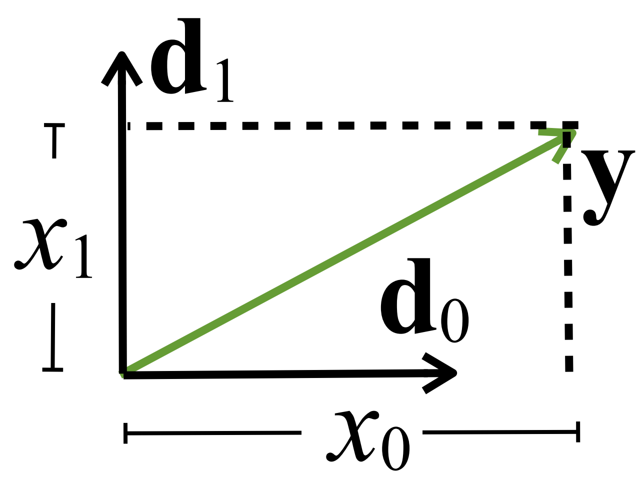


- Represent any vector as (x_0, x_1)
  - (called coefficients)

And since there's one and only one way to represent the vector, we can set the basis vectors as fixed, and store only the coefficients. This gives us our compact form

Basis vectors

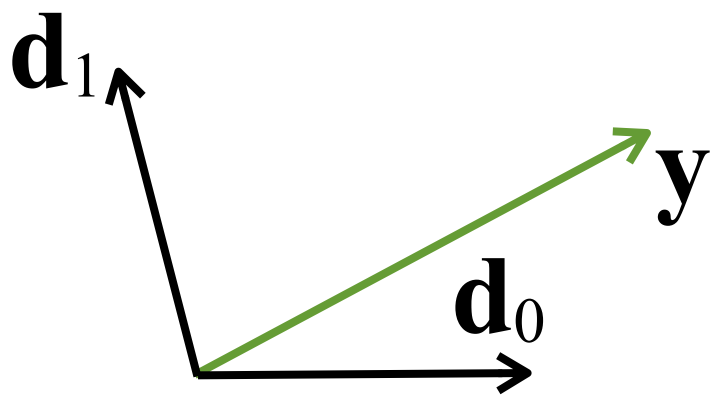
- If ONB, get coefficients via projection



$$x_0 = \frac{< \mathbf{d_0}, \mathbf{y} >}{\|\mathbf{d_0}\|}$$

Basis vectors

- If non-orthogonal

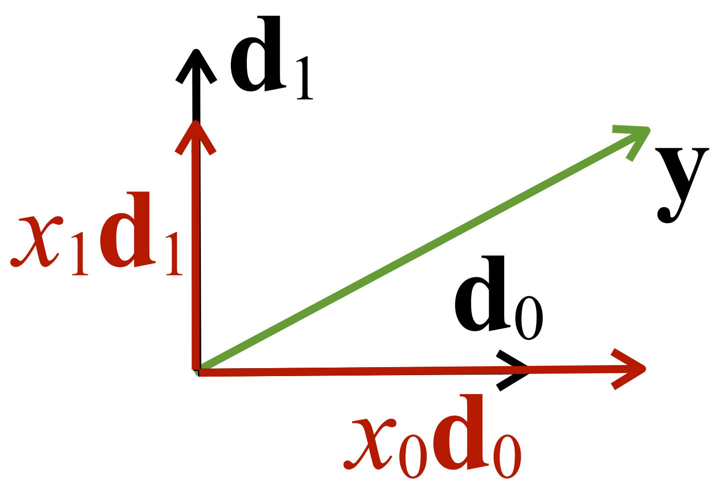


$$\mathbf{x} = \mathbf{D}^{-1}\mathbf{y}$$

In general, on a computer we don't want to invert the matrix due to numerical errors -- rather we'd solve a linear system. But mathematically this is correct.

Basis vectors

- Given a basis

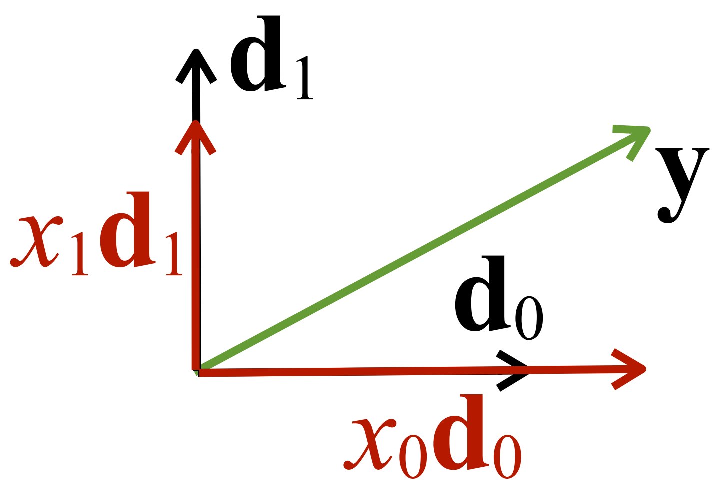


- Represent any vector as coefficients

So to sum this up:

Basis vectors

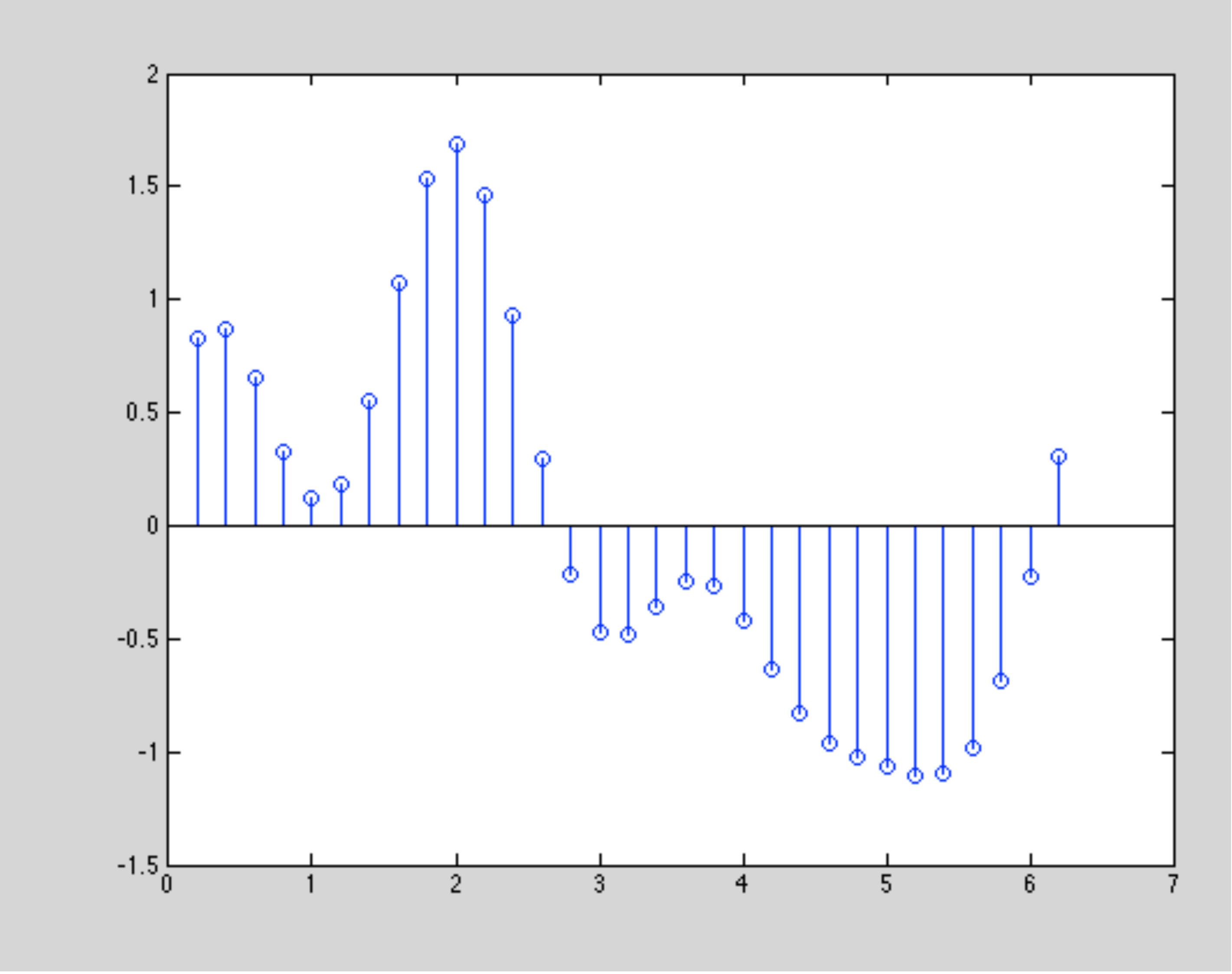
- Given a basis



- Represent any vector as coefficients
- Can do the same for signals

Basis vectors

- Suppose we have a sampled signal



In a game, this could be audio data, or the red component of a scanline in an image, or it could be rotation around y for an animated joint. This is clearly not a sparse representation -- there are a lot of different values here, and no zeroes.

Basis vectors

- Could represent as weighted sum of set of signals
  - Dictionary: set of signals used
  - Atom: element of the dictionary

We'll start off by assuming our dictionary is a basis, though as we'll see that doesn't have to be the case.

Basis vectors

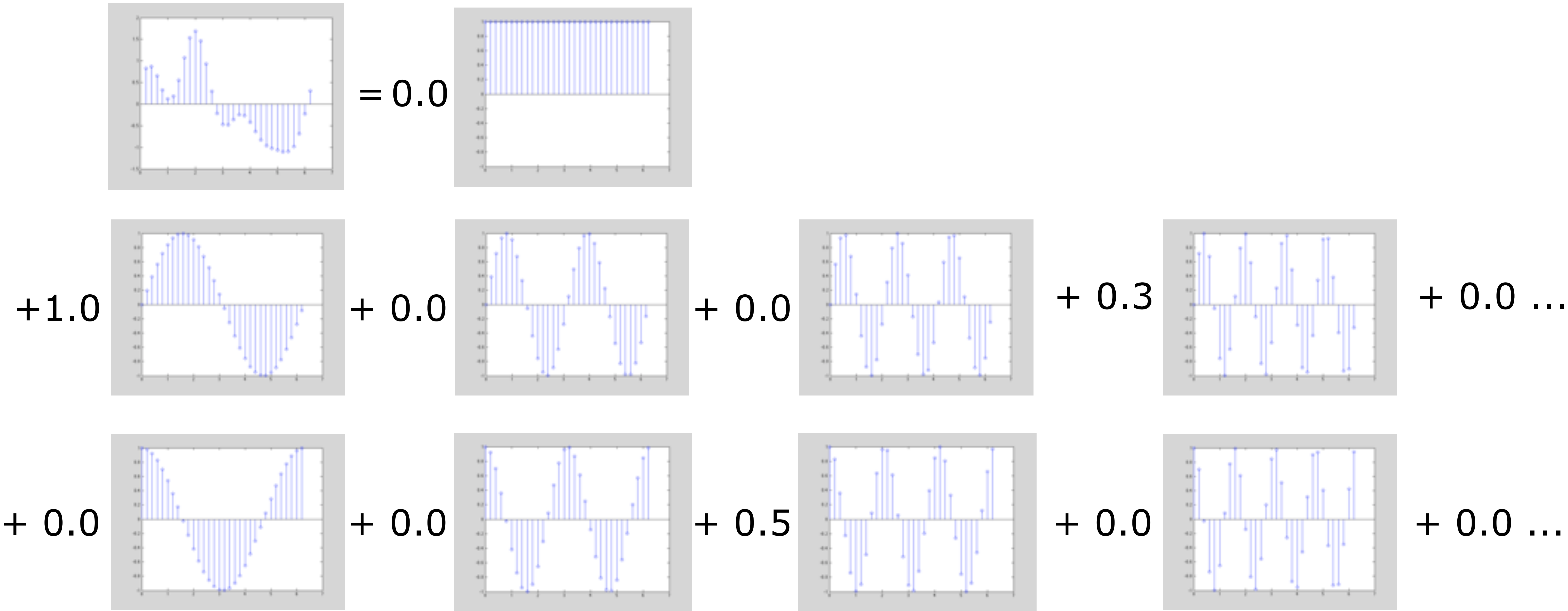
- Real Fourier series

$$\frac{1}{2}a_0 + \sum_{n=1}^{N} a_n \cos(nx) + \sum_{n=1}^{N} b_n \sin(nx)$$

- This is our dictionary

One possibility is to use the Fourier basis, which in the discrete case and using real coefficients, looks like this.

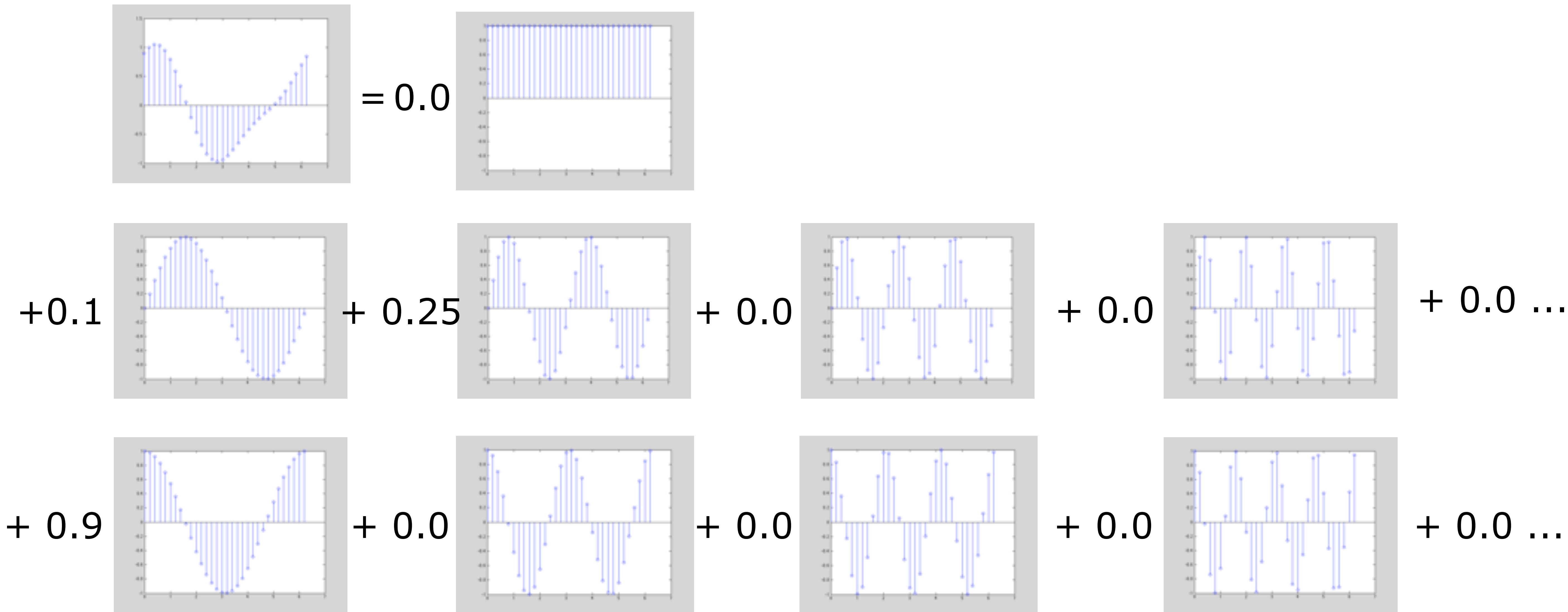
Basis vectors



Saturday, March 22, 14

Here is a portion of discrete Fourier basis, with its constant term, and various sines and cosines. The terms to the right are all scaled by 0, so I haven’t shown them. As you can see, a large number of the terms are multiplied by zero, so our data is sparse, and should compress quite well.

Basis vectors

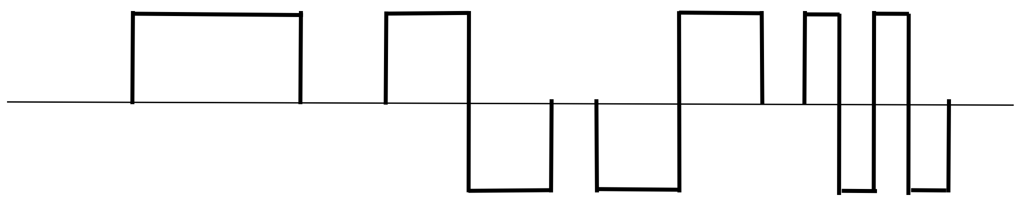


Saturday, March 22, 14

And we can represent a wide variety of signals simply by changing the weights.

Basis vectors

- Other bases:
  - Discrete cosine basis
  - Wavelets
    - Good for sampled/spiky data



- All orthonormal bases:
  - Easy to project

Problems with Fourier: we repeat the signal to make it periodic, which most of the time introduces a discontinuity. And Fourier is not good at representing discontinuities. Also, the general Fourier series uses complex coefficients. For this reason, most people use discrete cosine transform, which mirrors the signal to remove the edge discontinuity, and has real coefficients. But the problem with discontinuities in general is still there. For those types of signals, people use wavelets.

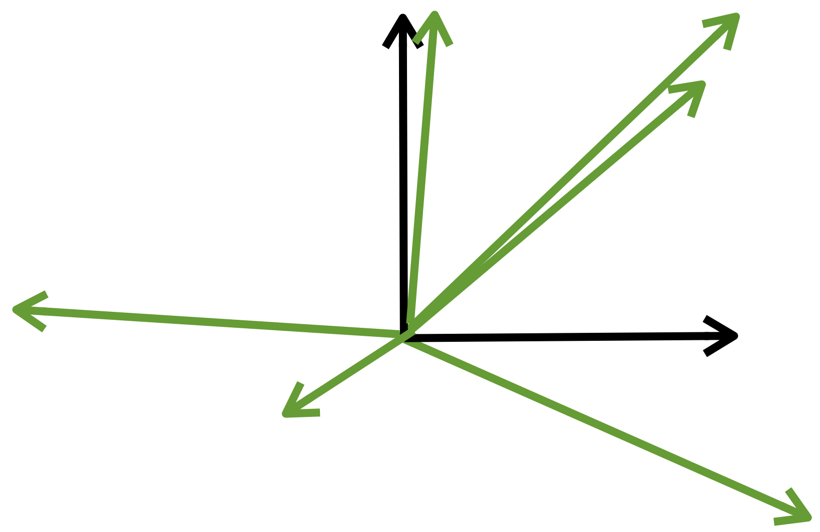
In all these cases, if scaled properly, they're all orthonormal bases.

Basis vectors

- Problem:
  - ONBs not always sparse

Basis vectors

- In general, need all coefficients for ONB

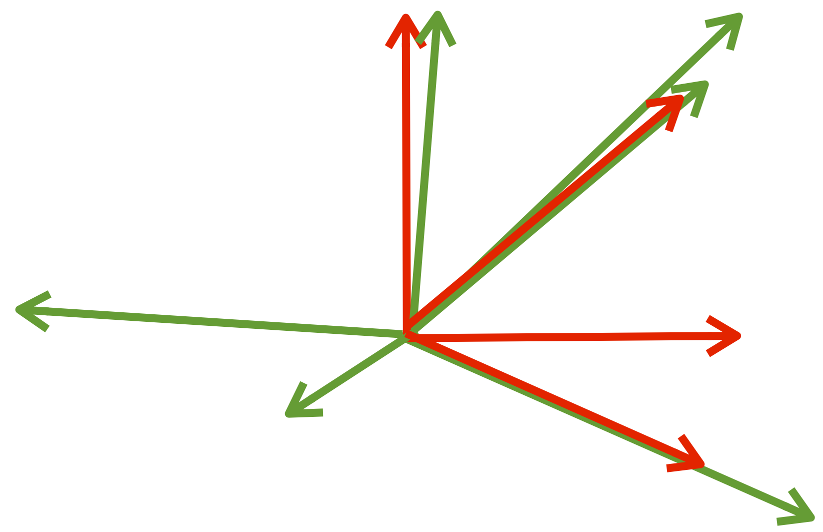


- Bad for compression algorithms

Suppose we'd like to drop coefficients to reduce our signal, i.e. to compress this set of vectors. Using the standard orthonormal basis, we can't just drop one coefficient for each vector without losing a significant amount of information. In the case of the 2D vector, it's not so bad as it's only two values -- but suppose we have a signal with a significant number of samples. If we have to represent it using the same or close to the number of coefficients relative to one of our orthonormal bases, then we're not gaining anything. Instead, it would be great if we could significantly reduce the number of coefficients needed without huge errors.

Frames

- Solution: add vectors to create a *frame*



Instead, we can add more vectors. A frame is an overspecified basis -- we have more vectors than we need to span the space of all vectors. But it has the advantage that we can pick the vectors we need for a given data element to get a decent compression. In this case we can now use only one value per input vector. Note that our goal may not be exact reproduction: lossy compression is okay.

Frames

- Frame vectors $\mathbf{e}_k$ must fulfill *frame condition*

$$\forall \mathbf{v} : A\|\mathbf{v}\|^2 \leq \sum_k | < \mathbf{v}, \mathbf{e}_k > |^2 \leq B\|\mathbf{v}\|^2$$

- where

$$0 < A \leq B < \infty$$

Saturday, March 22, 14

Basically this means that for every v in our space, the projections onto the frame are bounded, and that the frame vectors can be used to represent the entire space we care about. I wouldn't worry too much about it, other than you can't just pick any old vectors and get a frame. We'll talk later about methods for picking frame vectors for a given data set later. That said, it's still useful to keep them unit length.

Frames

- Can do the same for signals
- E.g. use dictionary of DCT and wavelets to cover both smooth and chunky data

Frames

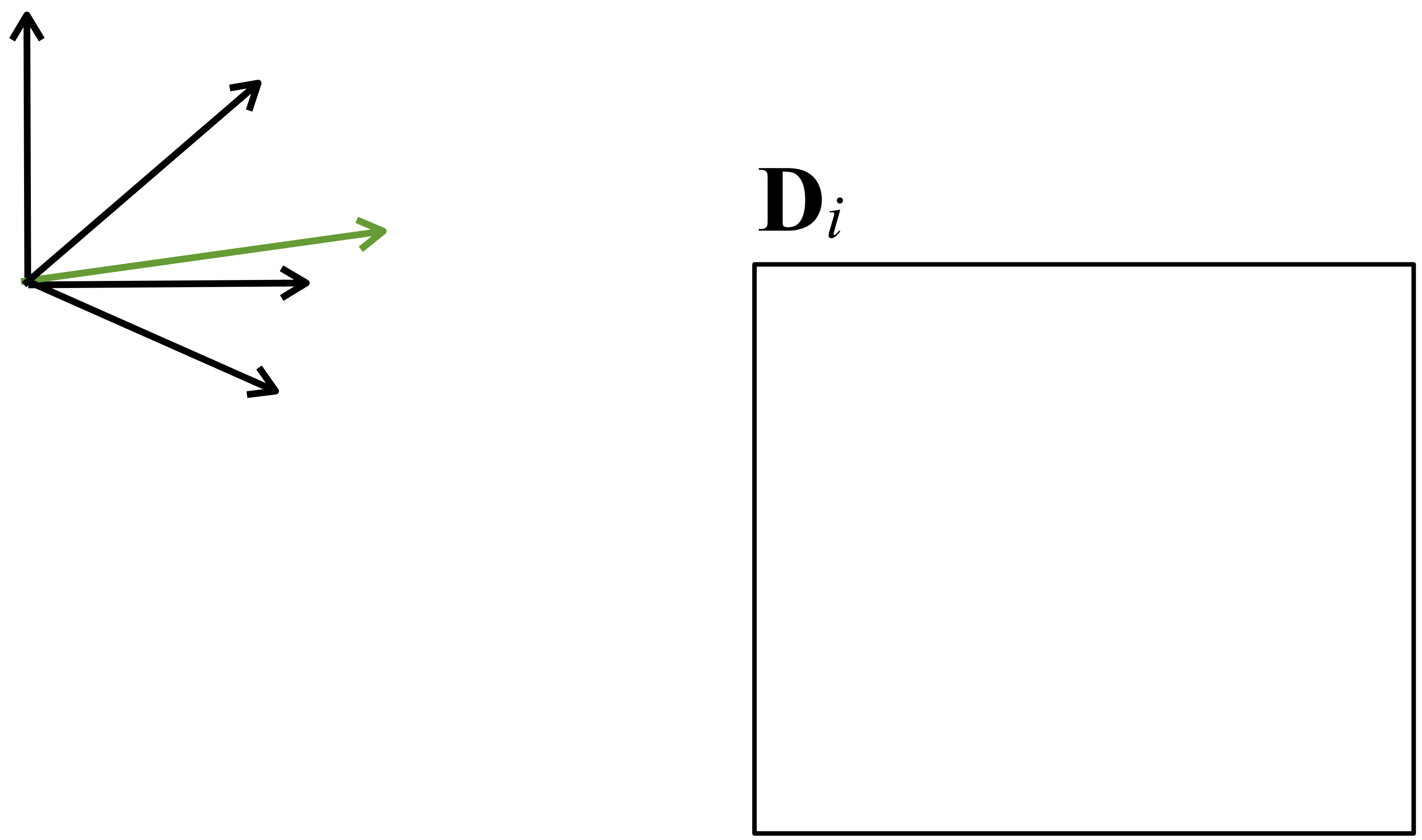
- Given vector and dictionary
- Want minimal set of atoms. How?
  - Least squares (slooooooooowwwwww)
  - Greedy algorithms
    - Matching pursuit

The problem with a frame is that now we have an infinite number of possibilities for our coefficients. How do we pick the ones we want?

Matching Pursuit

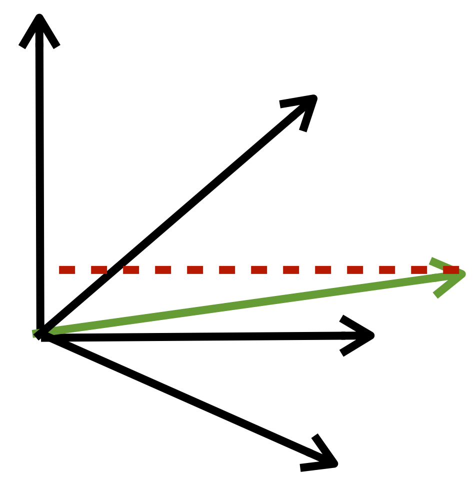
- Method for finding coefficients for $\mathbf{v}$ and given dictionary
  - Project $\mathbf{v}$ onto all atoms in dictionary
  - Take greatest magnitude projection
  - Subtract scaled atom from $\mathbf{v}$
  - Repeat until $\mathbf{v}$ is sufficiently small, or certain # iterations

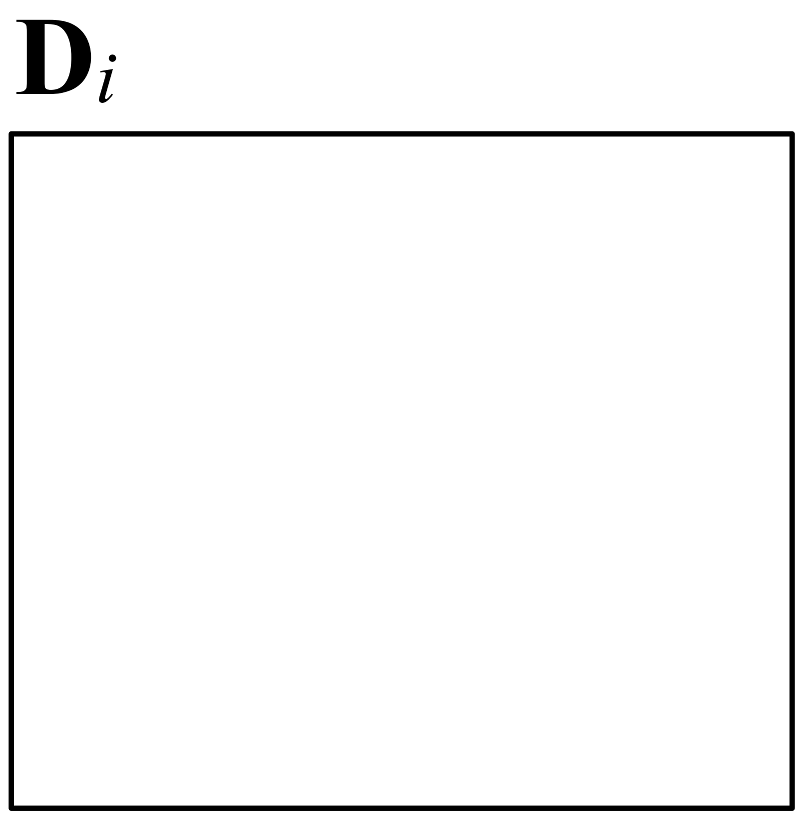
Matching Pursuit



Let’s run through an example using 2D vectors. Here the black vectors are our dictionary, and the green vector is the one we’re compressing. The box is the active set of atoms we’re using to represent our original vector.

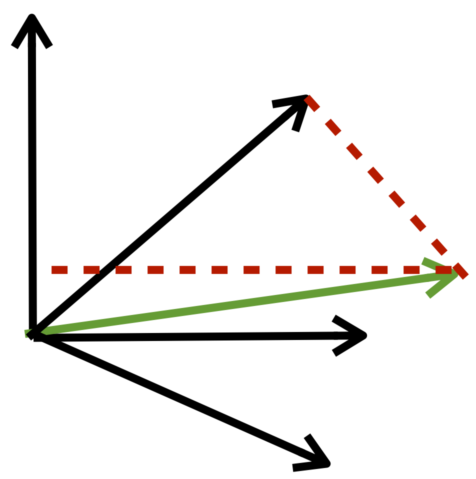
Matching Pursuit

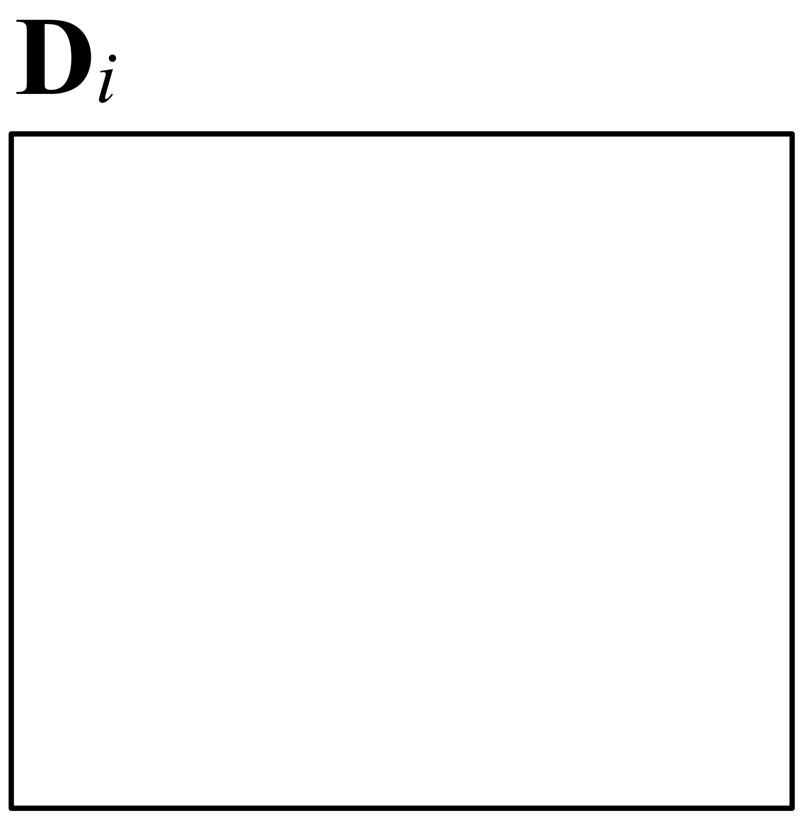




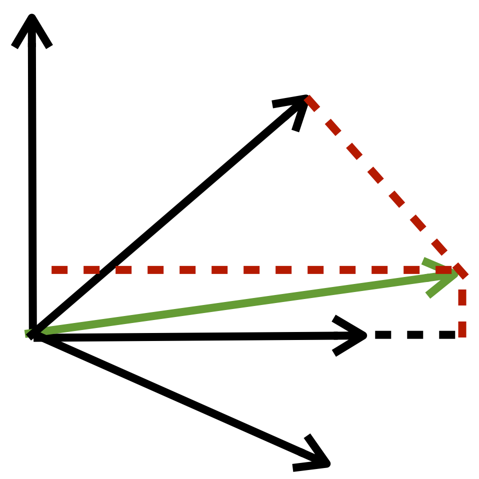
We begin by projecting onto each of the atoms

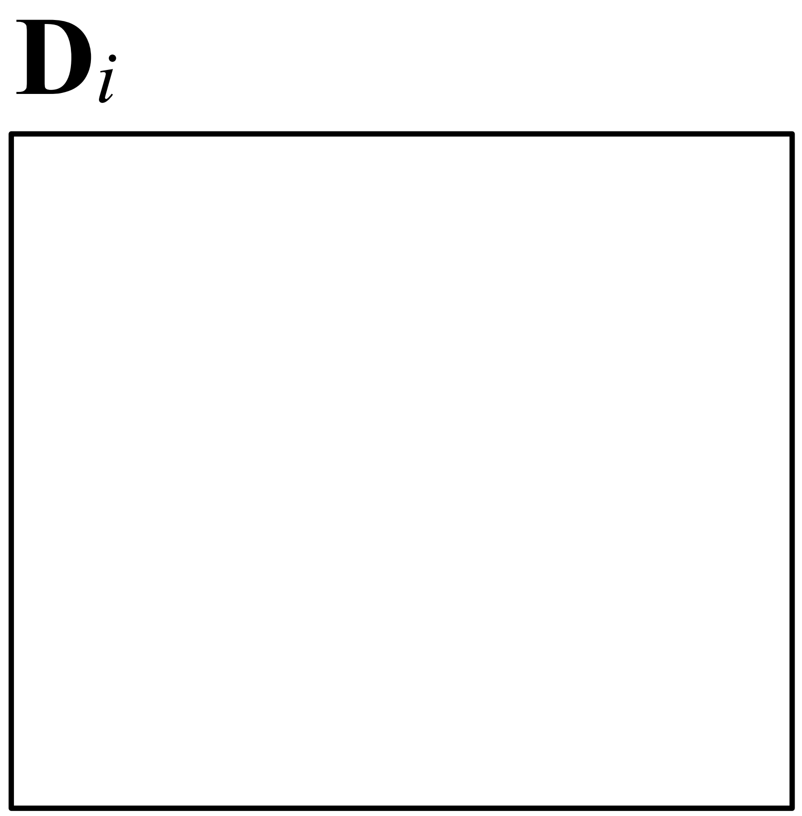
Matching Pursuit



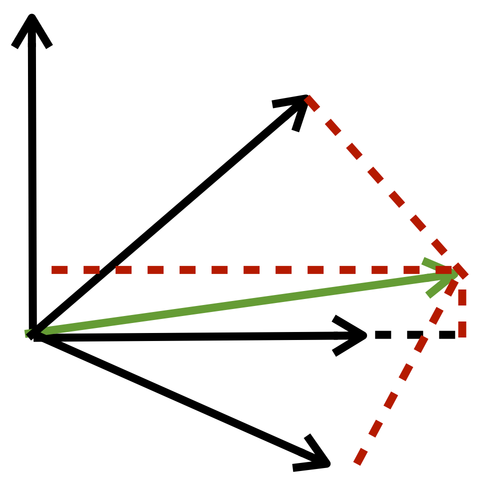


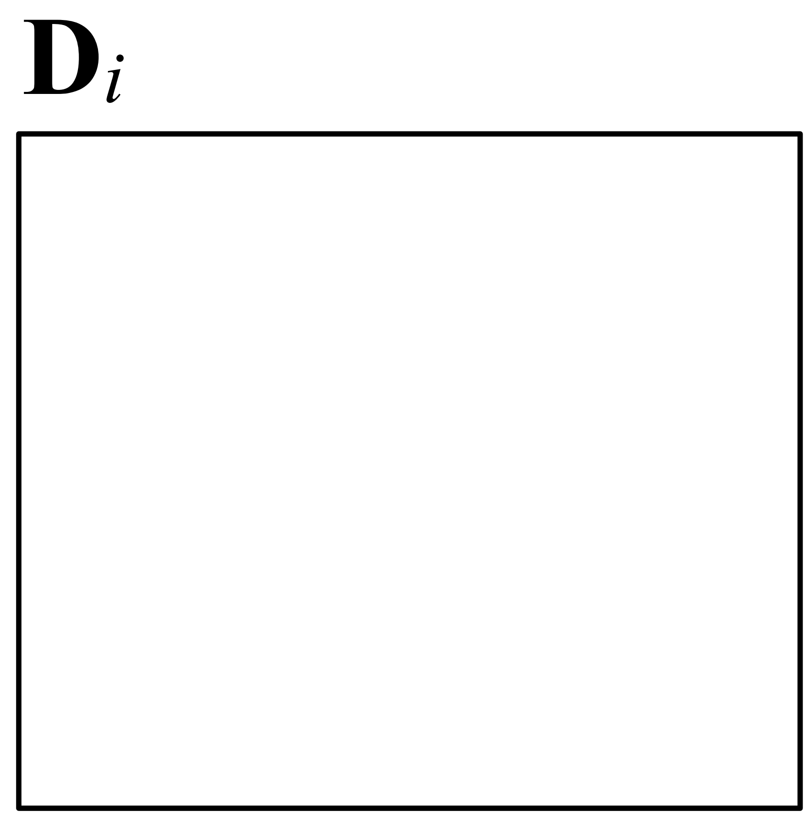
Matching Pursuit



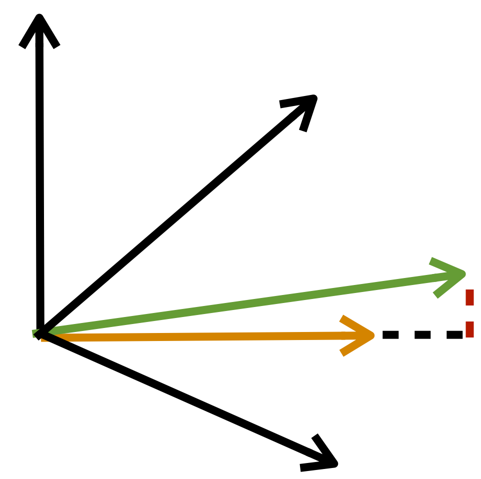


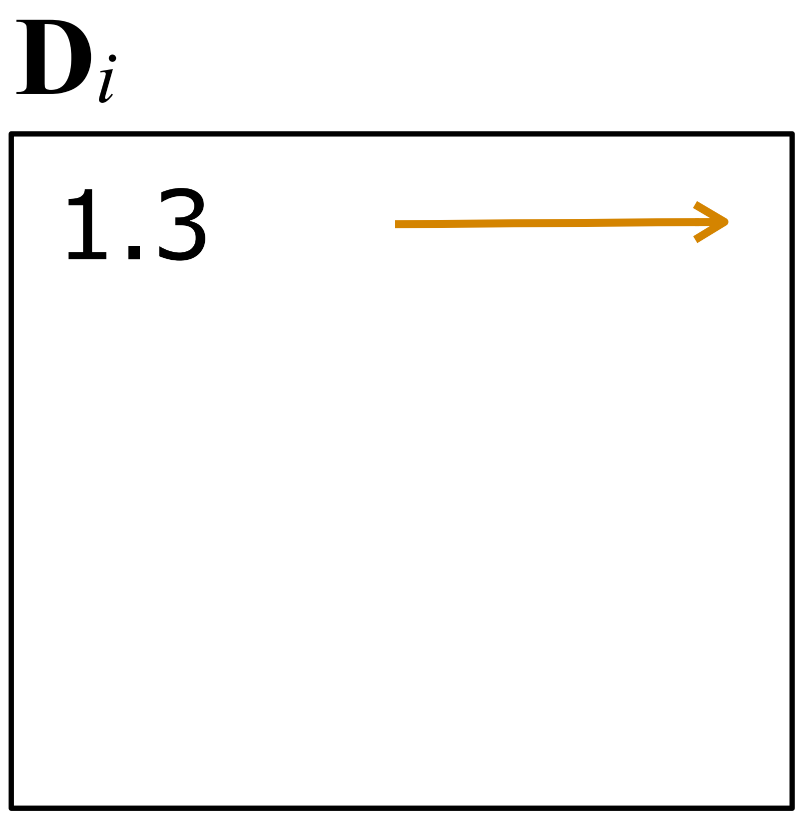
Matching Pursuit





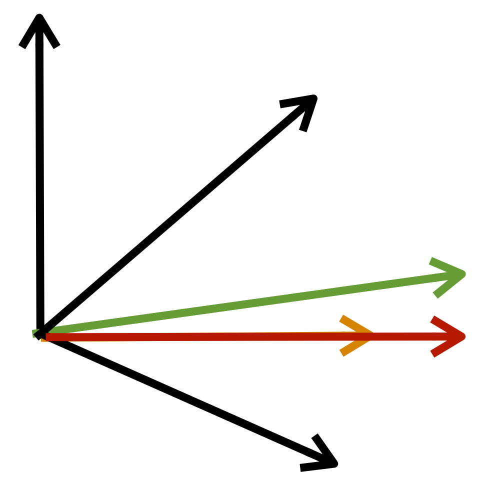
Matching Pursuit

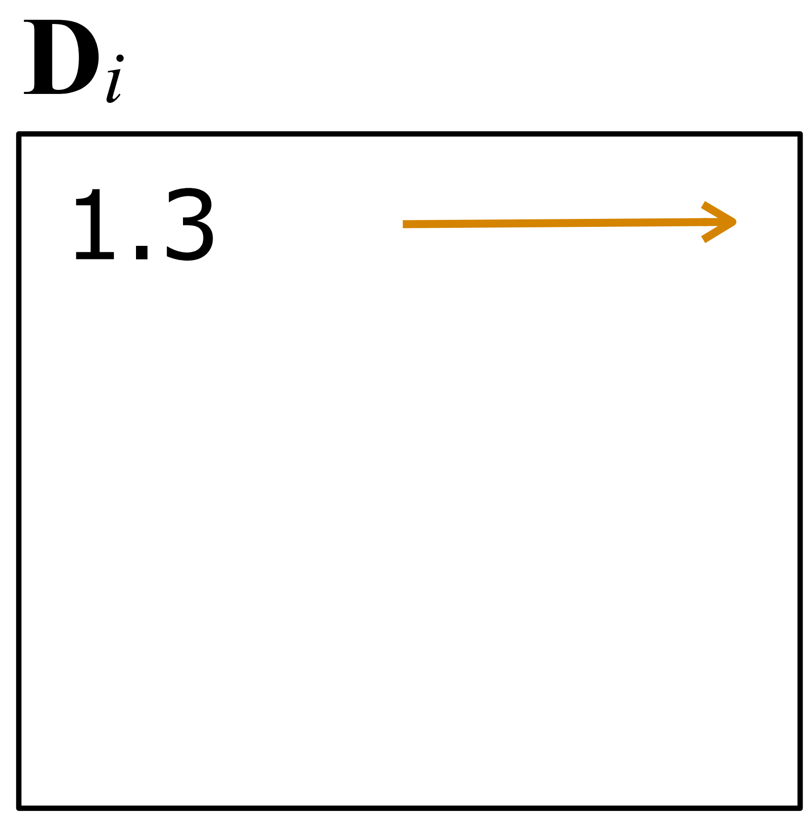




So now we pick the atom with the largest projection and add it to our set, along with the coefficient

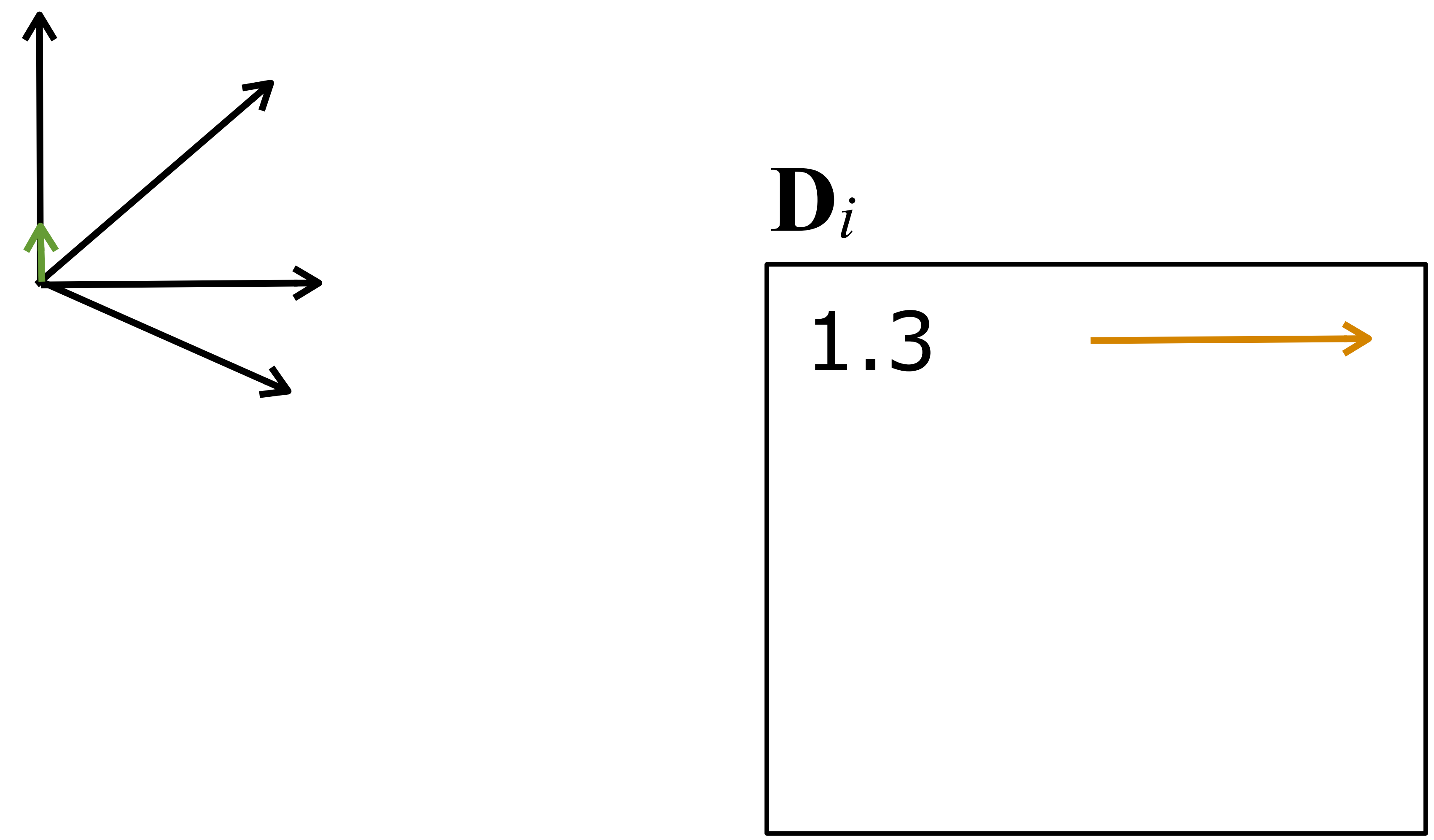
Matching Pursuit





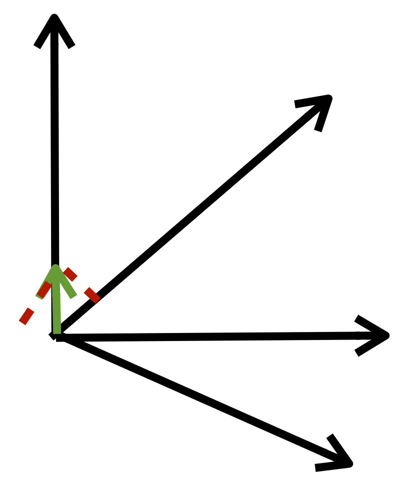
Then we subtract the portion of the residual pointing along the chosen atom...

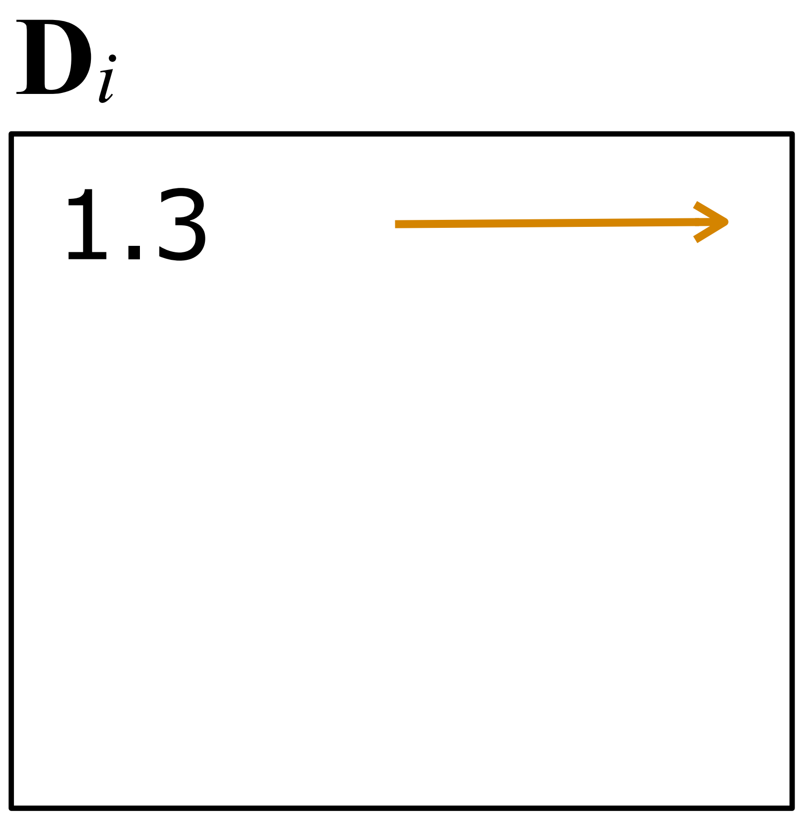
Matching Pursuit



... to get our new residual. At this point we might decide that our error is small enough, or we might continue. Let's continue.

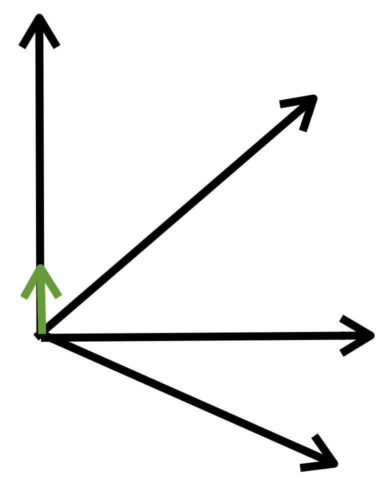
Matching Pursuit



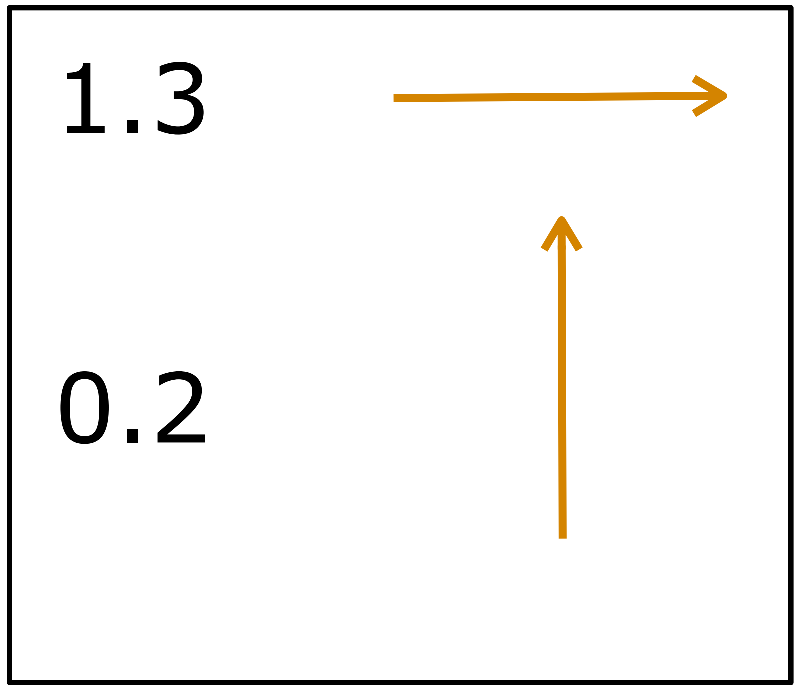


Projecting on all the dictionary again, we see that the longest projection is on the vector pointing up, so we add that to our active set...

Matching Pursuit



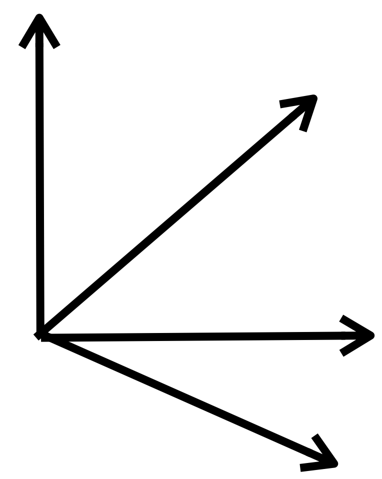
$\mathbf{D}_i$

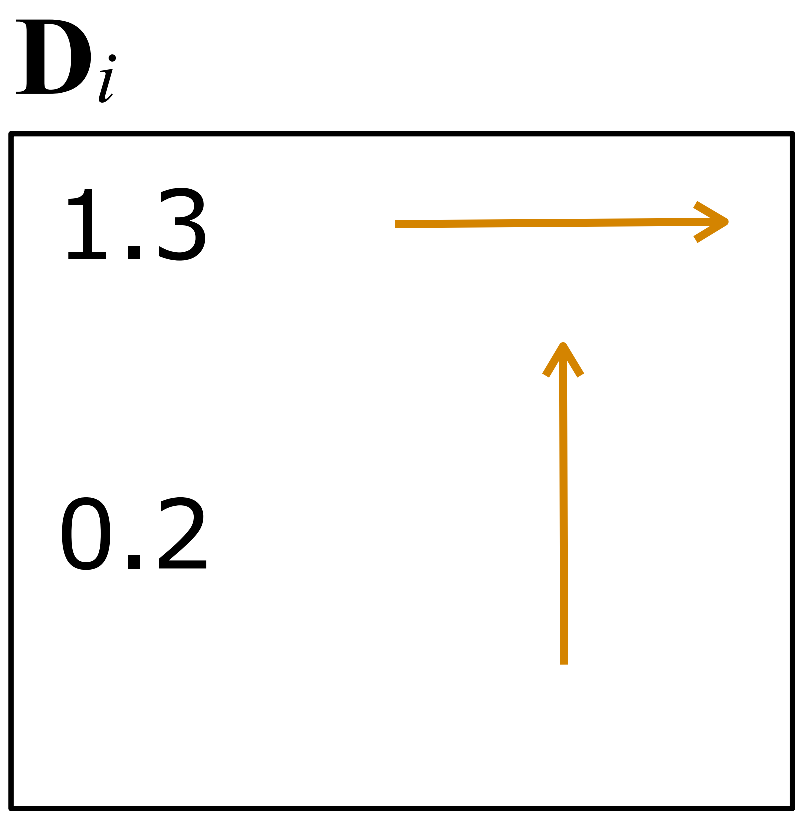


Saturday, March 22, 14

And after subtracting the new atom scaled by the new coefficient,

Matching Pursuit



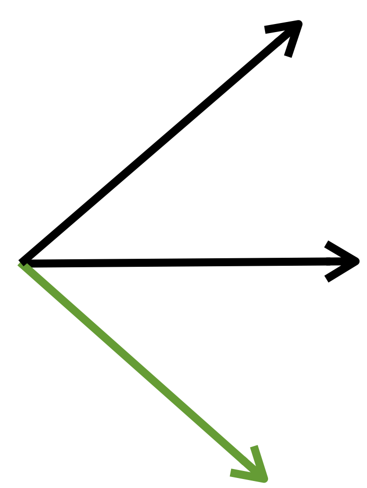


we'll end up with the set of atoms that can represent our original vector. For lossy compression, we could drop the 0.2 term.

Matching Pursuit

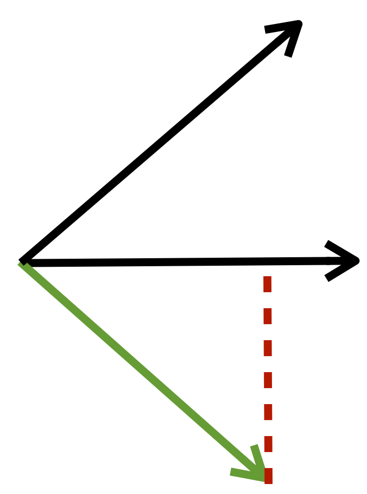
- Will converge to solution, but:
  - Can oscillate between a small set of atoms

Matching Pursuit



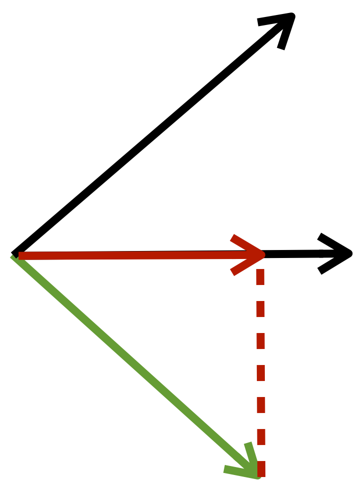
Let’s try another example, but with only two frame vectors. This is ultimately silly because this is a basis, and we could just invert a matrix to solve it, but it does a good job of illustrating the problem.

Matching Pursuit



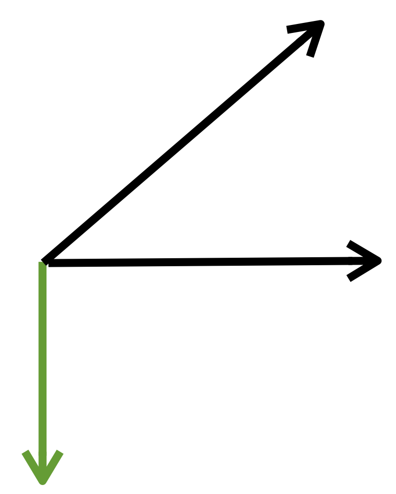
So, project and find the largest projection

Matching Pursuit



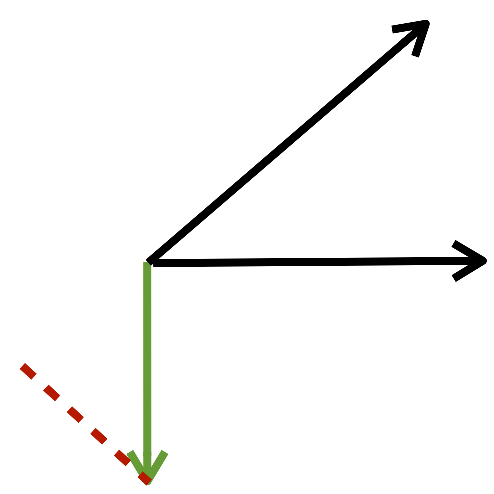
Subtract projected portion...

Matching Pursuit



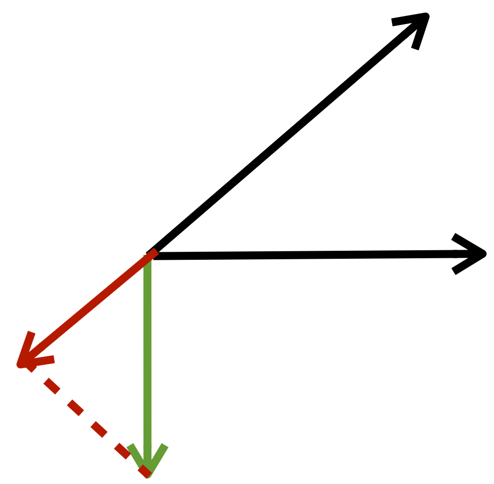
... to get new residual

Matching Pursuit



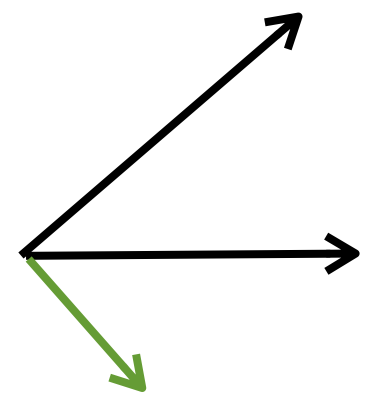
Find max projection again

Matching Pursuit



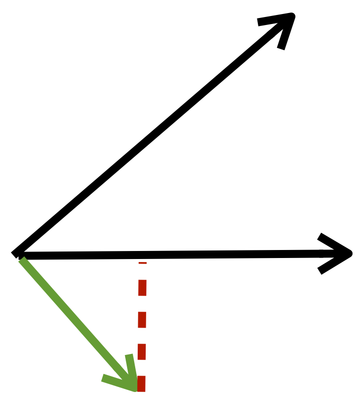
Subtract projected portion

Matching Pursuit



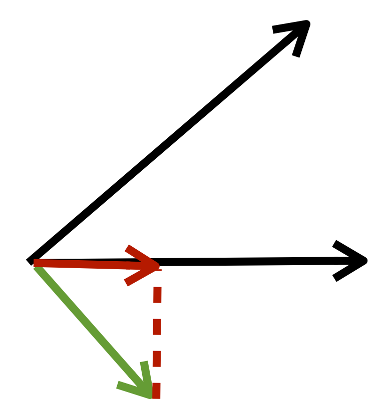
To get new residual. Note that this is pointing the same direction as the original vector, just shorter.

Matching Pursuit



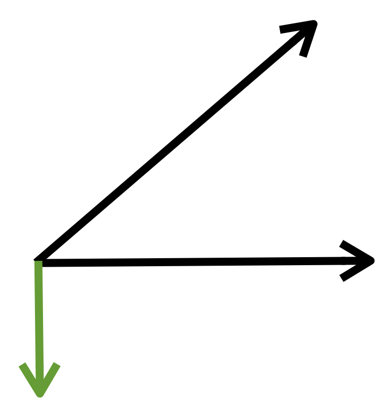
We project again...

Matching Pursuit



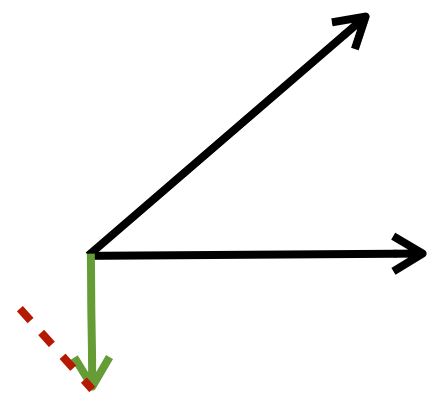
Subtract the scaled atom

Matching Pursuit



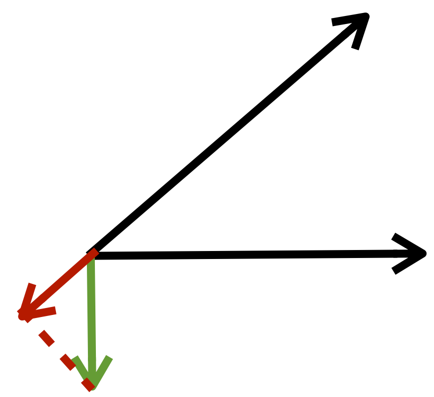
To get the new residual

Matching Pursuit



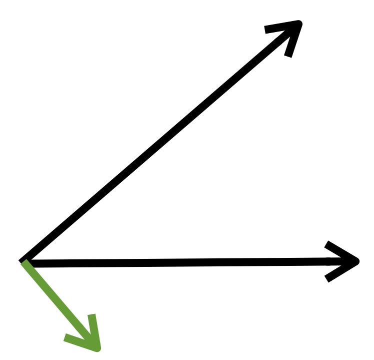
We project again...

Matching Pursuit



And subtract...

Matching Pursuit



Saturday, March 22, 14

And here we are again, just shorter. So we're just going to keep oscillating between these two vectors.

Orthogonal Matching Pursuit

- Refinement of MP
  - Update all coefficients computed so far by reprojecting onto current set of atoms, before subtracting
  - Better results

Orthogonal Matching Pursuit

- Reprojection step
  - Ideally do

$$\mathbf{x} = \mathbf{D}_i^{-1}\mathbf{y}$$

Orthogonal Matching Pursuit

- Reprojection step
  - Ideally do

$$\mathbf{x} = \mathbf{D}_i^{-1}\mathbf{y}$$

Not square

Orthogonal Matching Pursuit

- Reprojection step
  - Ideally do

$$\mathbf{x} = \mathbf{D}_i^{-1}\mathbf{y}$$

Not square

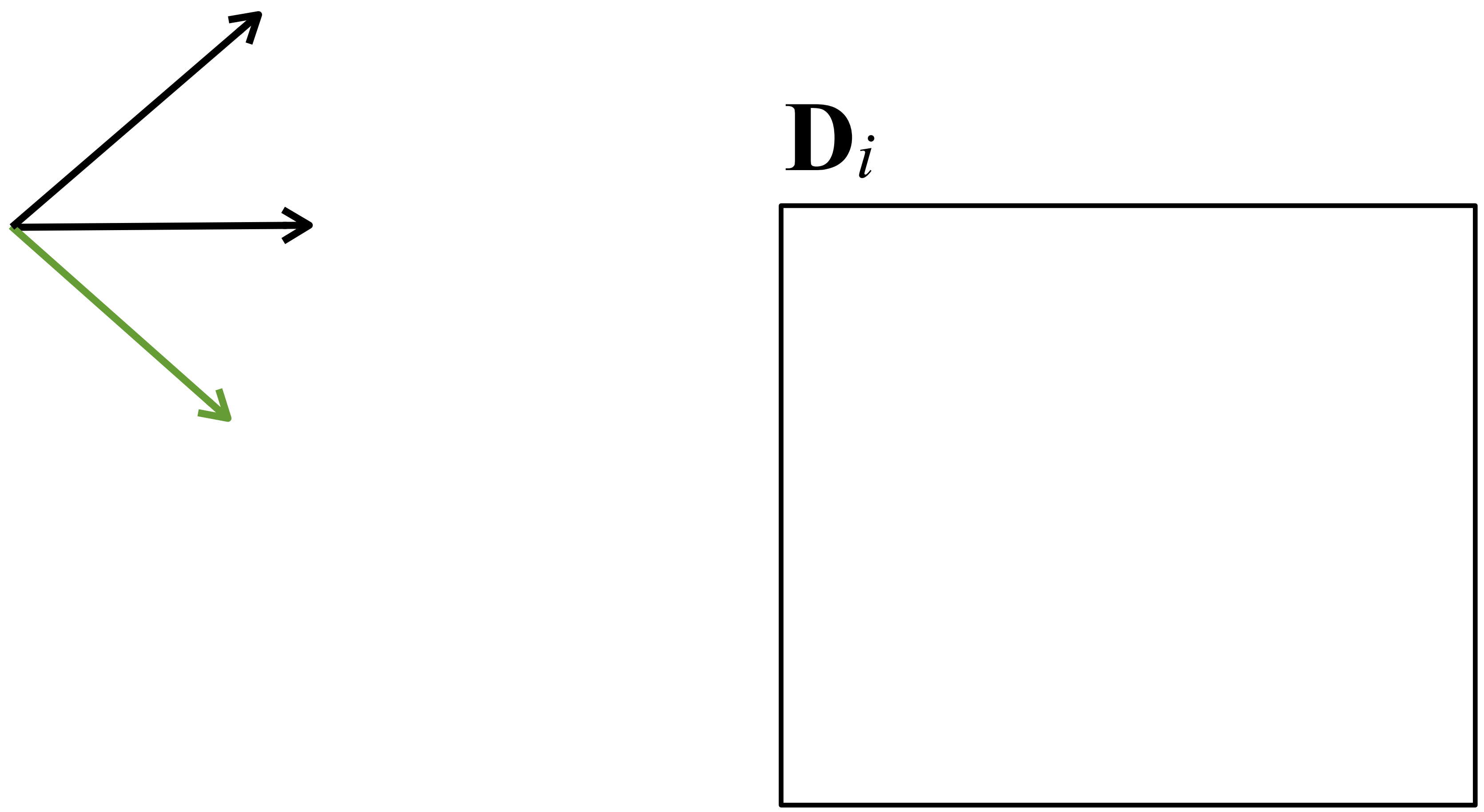
  - Instead:

$$\mathbf{x} = (\mathbf{D}_i^T\mathbf{D}_i)^{-1}\mathbf{D}_i^T\mathbf{y}$$ (pseudo-inverse)

Saturday, March 22, 14

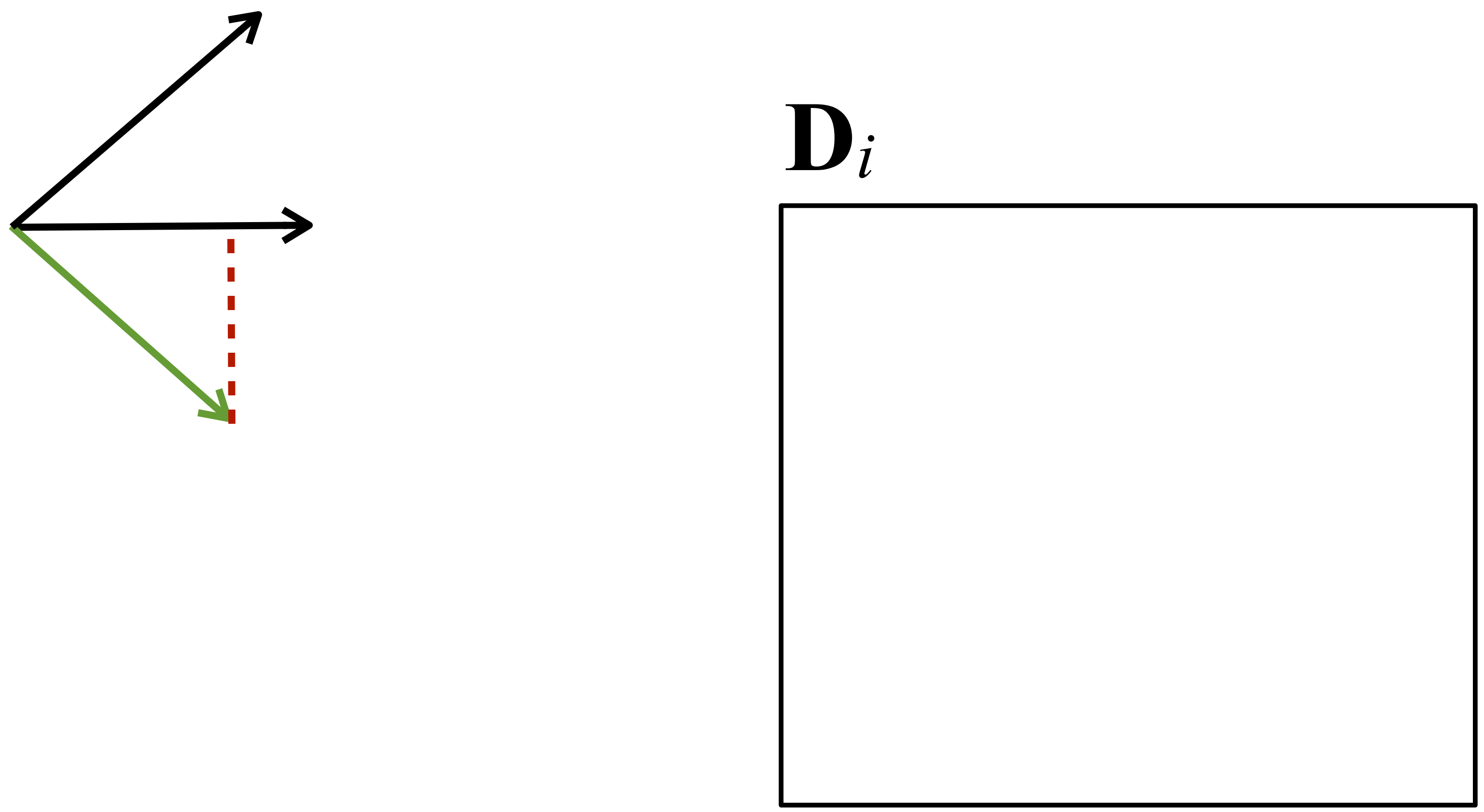
One note on the pseudo-inverse -- doing it directly is not stable (floating point error), so need to use Cholesky decomposition. See Robin and Manny’s talk from last year for more details on this.

Orthogonal Matching Pursuit



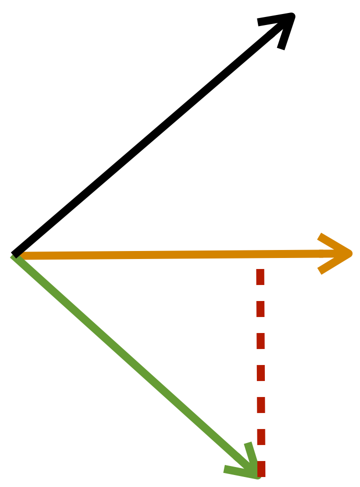
So let's try that again

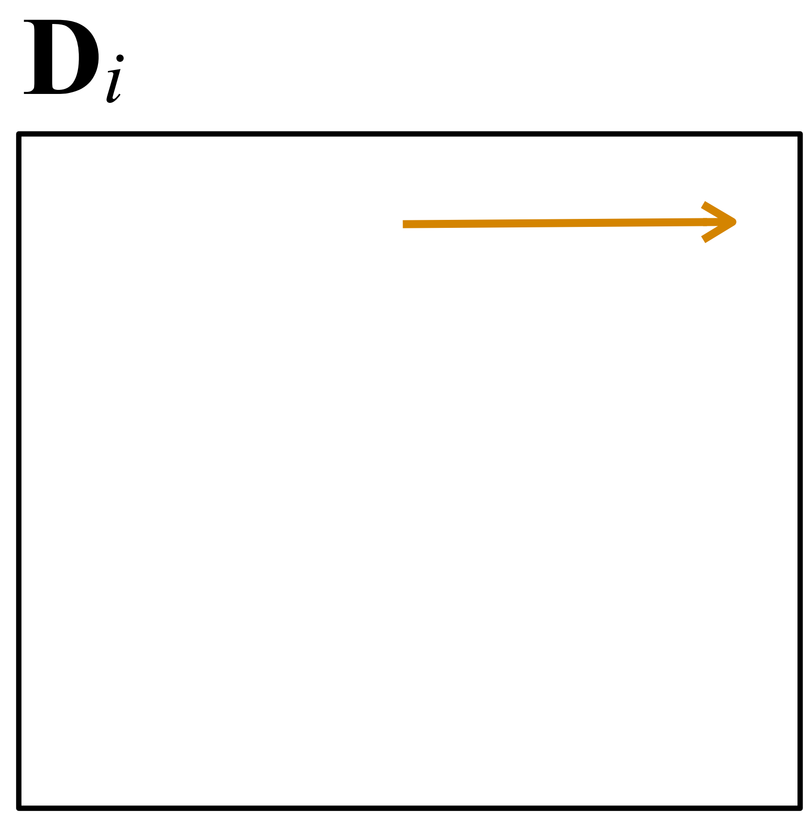
Orthogonal Matching Pursuit



Project and find the largest projection

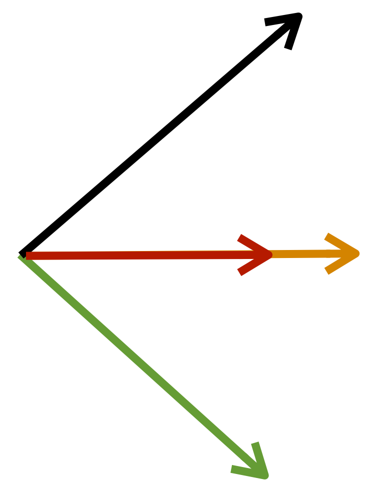
Orthogonal Matching Pursuit

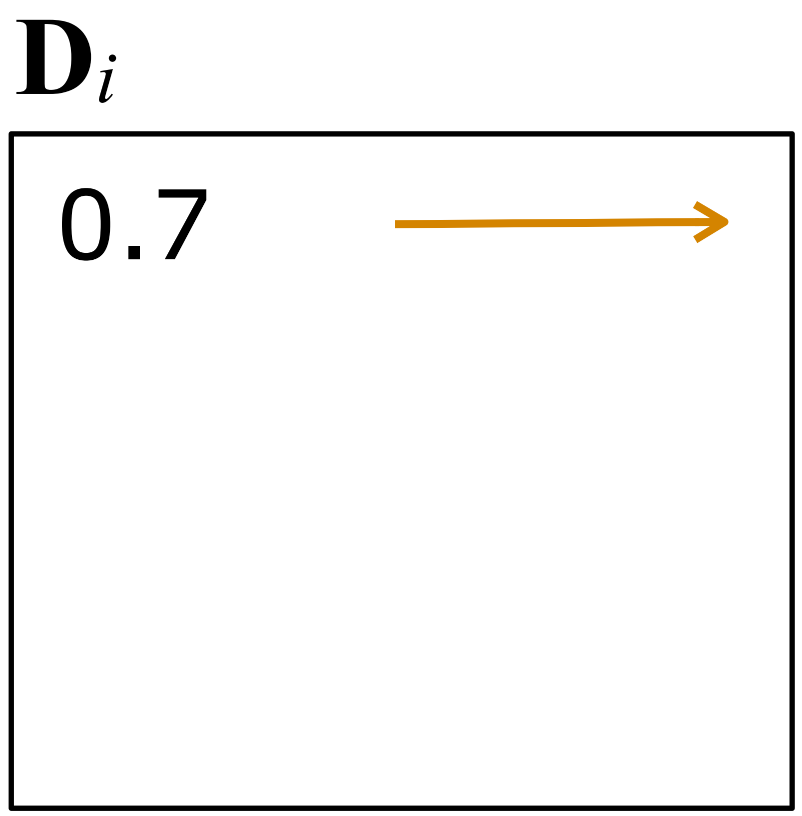




Add that to our current set of atoms

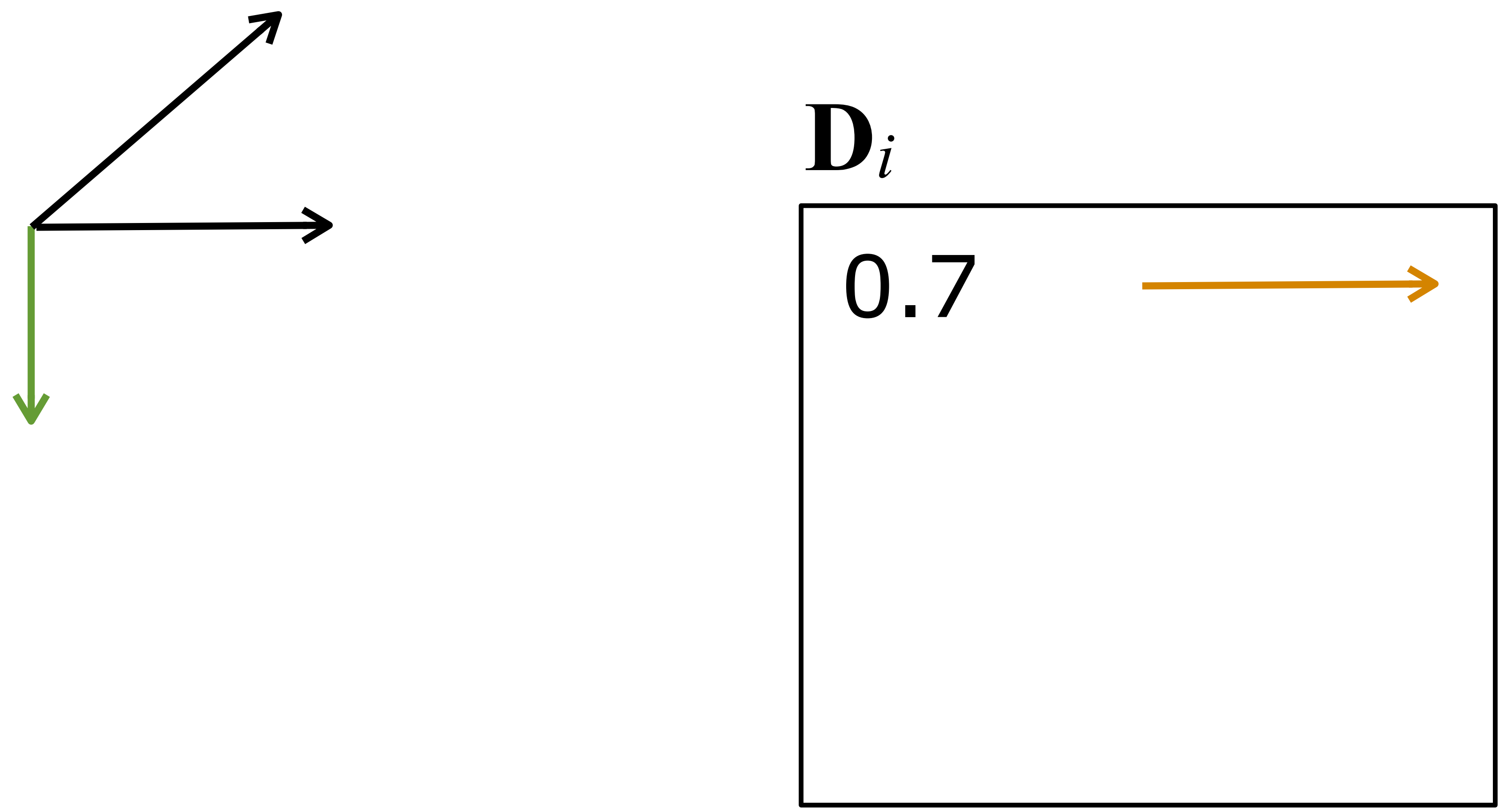
Orthogonal Matching Pursuit





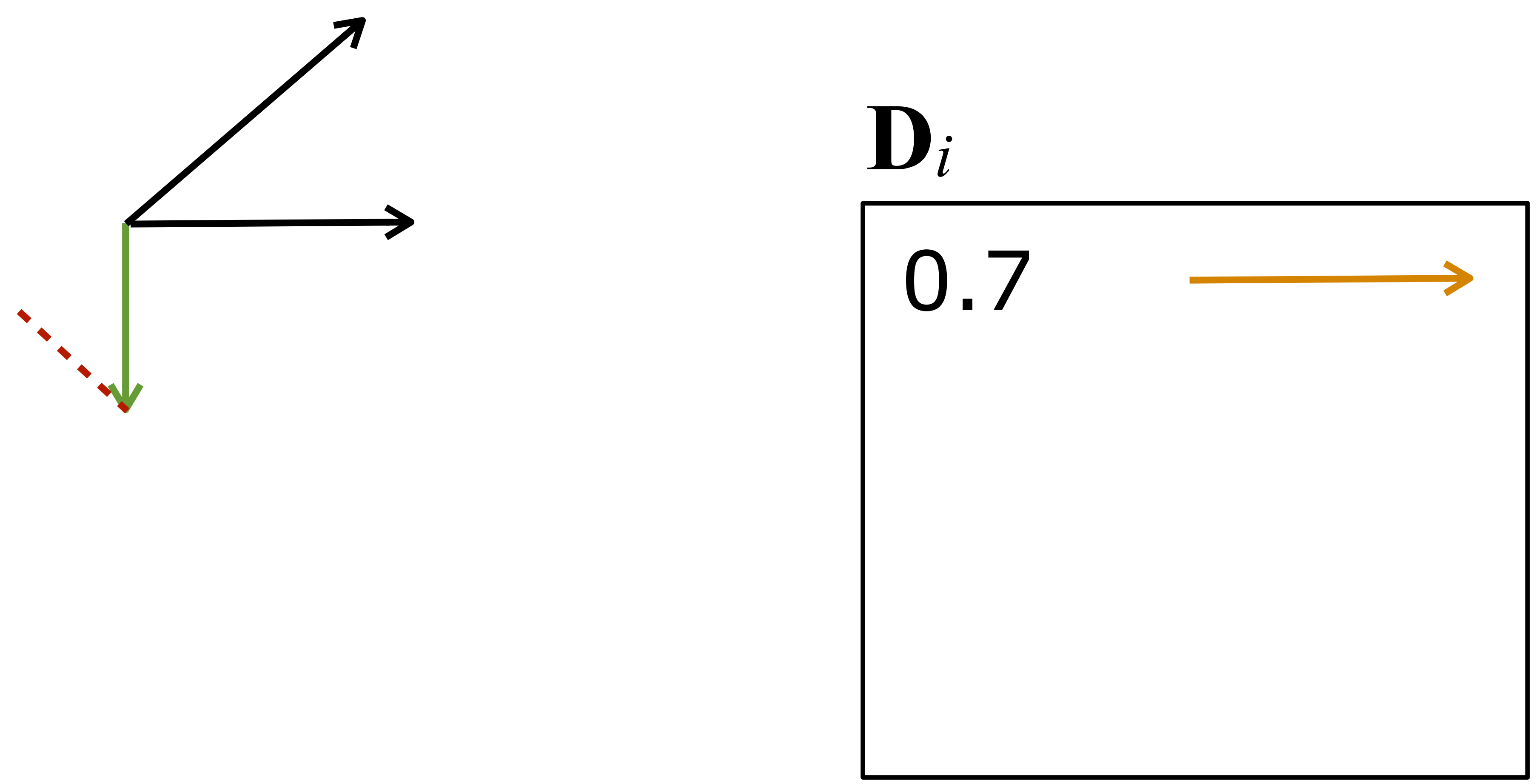
The we reproject the original vector against the single atom in our current set to get our coefficient

Orthogonal Matching Pursuit



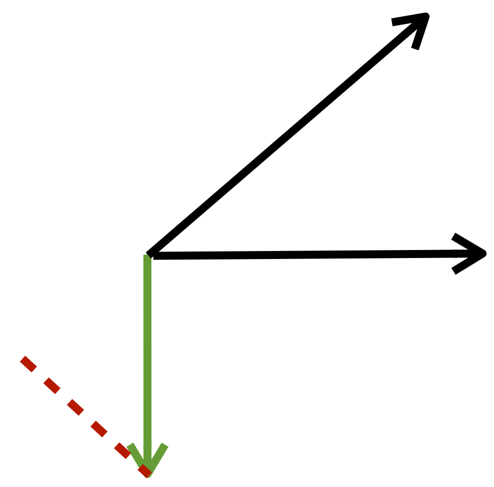
Subtract projected portion to get new residual

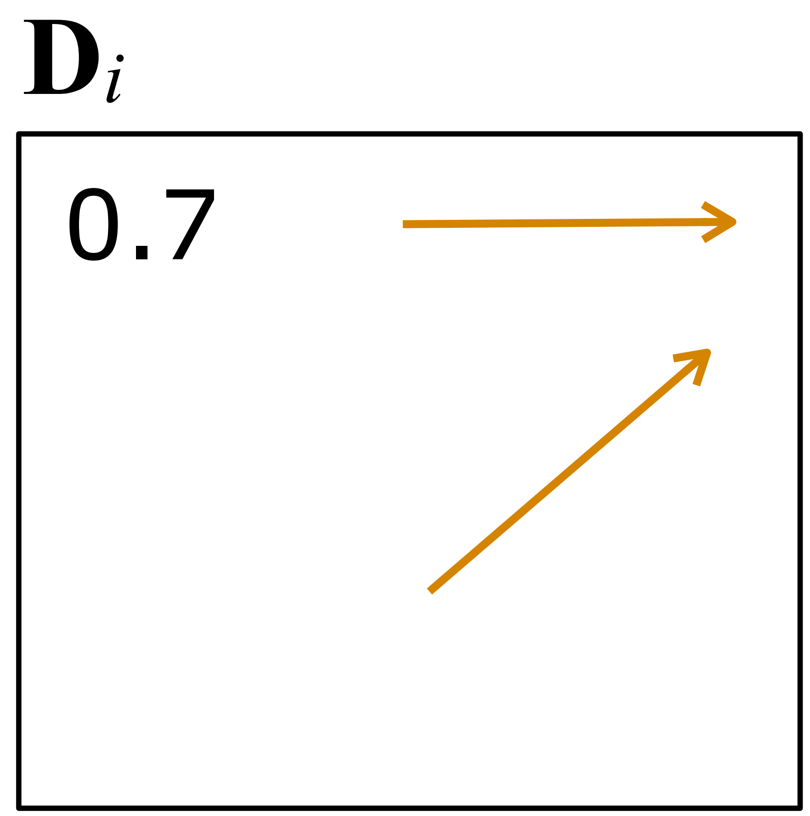
Orthogonal Matching Pursuit



Find max projection again

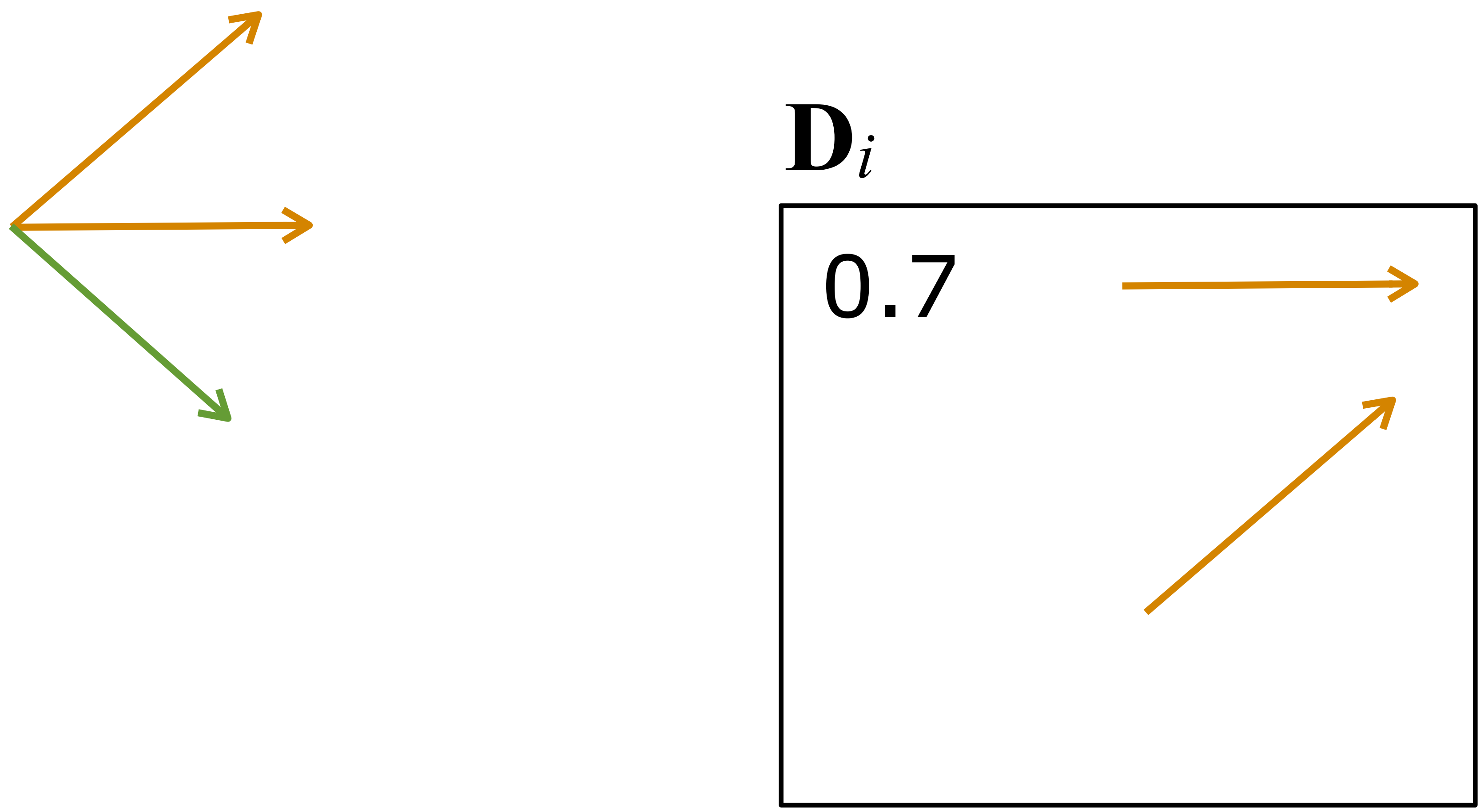
Orthogonal Matching Pursuit





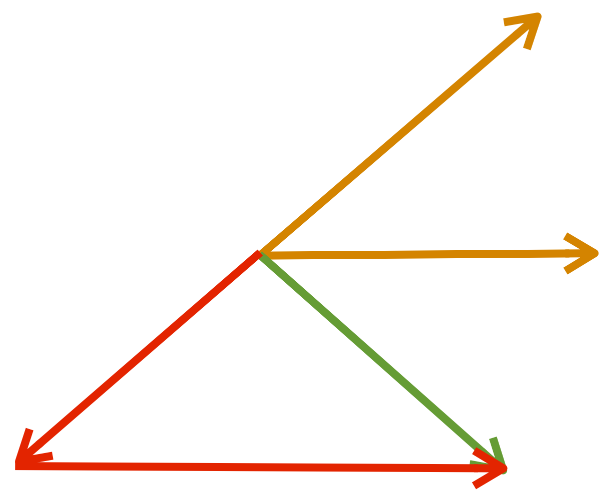
And add that atom to our current set

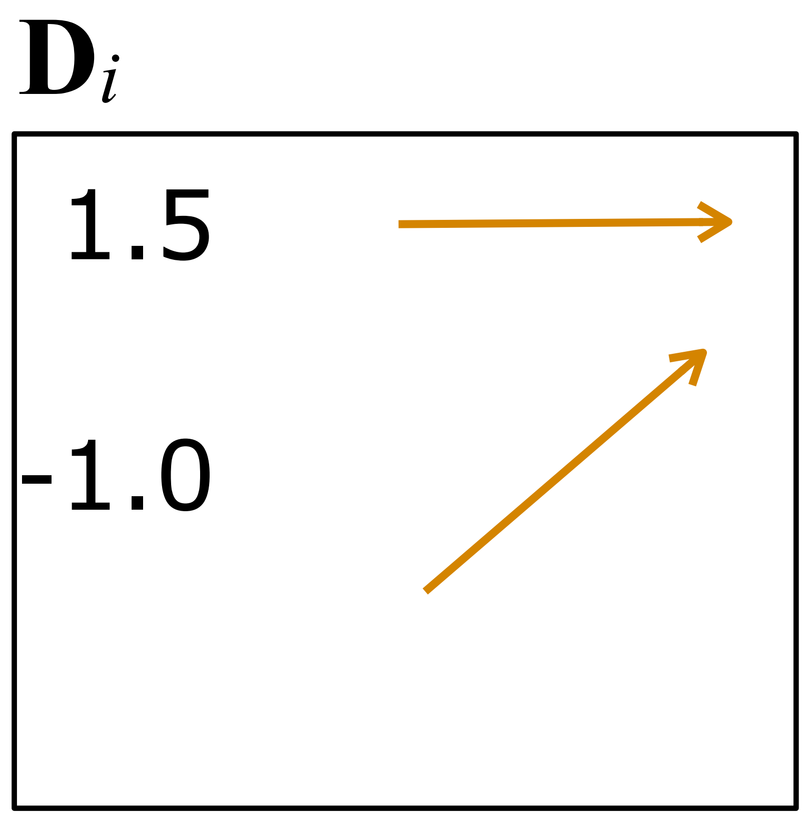
Orthogonal Matching Pursuit



At this point, these are the two atoms in our active set. So we reproject the original vector against these to update the coefficients...

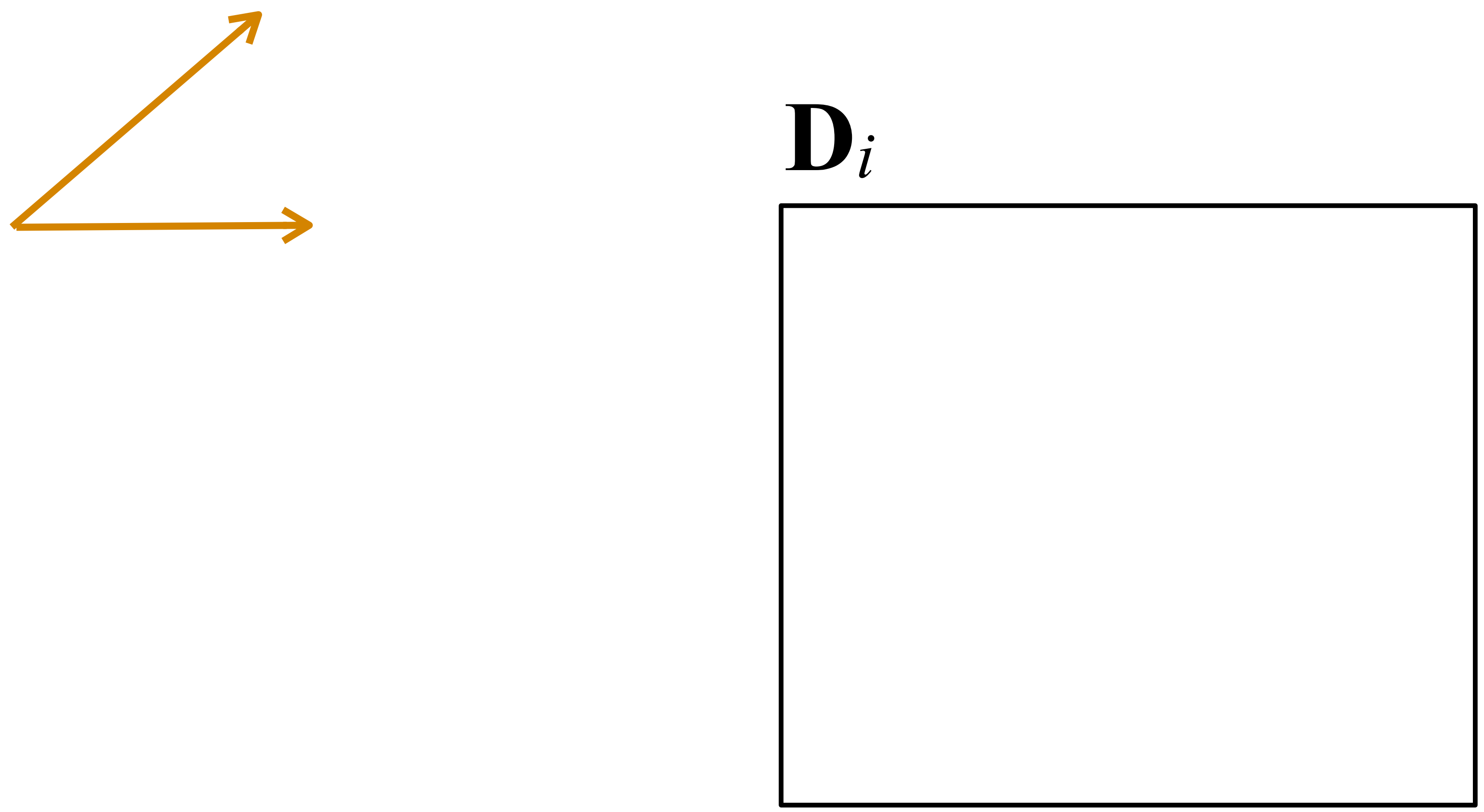
Orthogonal Matching Pursuit





To get something like this. Then we subtract the scaled atoms from the original vector to get the new residual....

Orthogonal Matching Pursuit



Saturday, March 22, 14

Which is negligible, so we're done.

Orthogonal Matching Pursuit

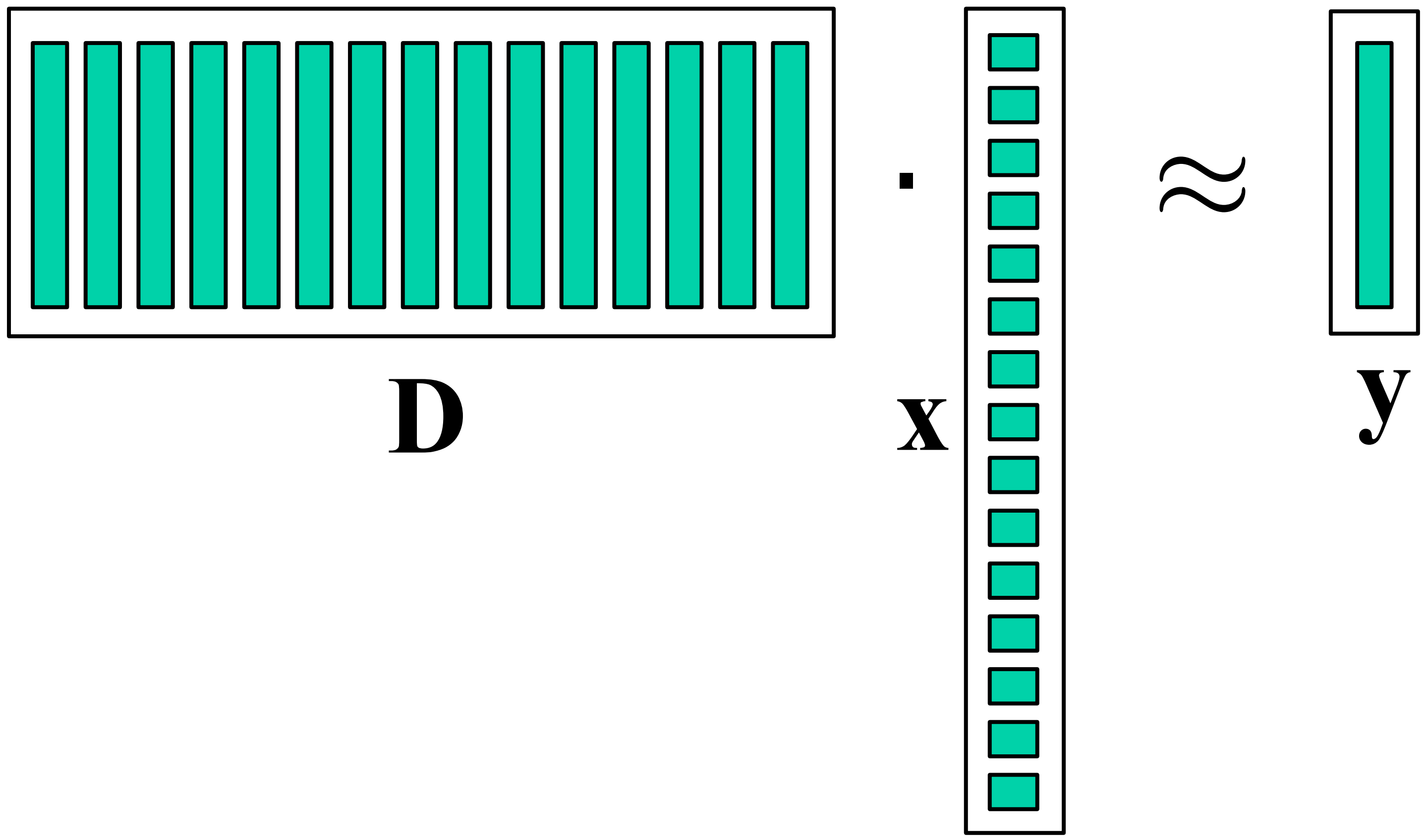
- Reprojection step takes extra time
- But converges much quicker!

Choosing a Dictionary

- Can just pick one
  - E.g. DCT + wavelets
- Refine from training set of data
  - K-SVD

K-SVD

- Can represent signal rep. as matrix mult.

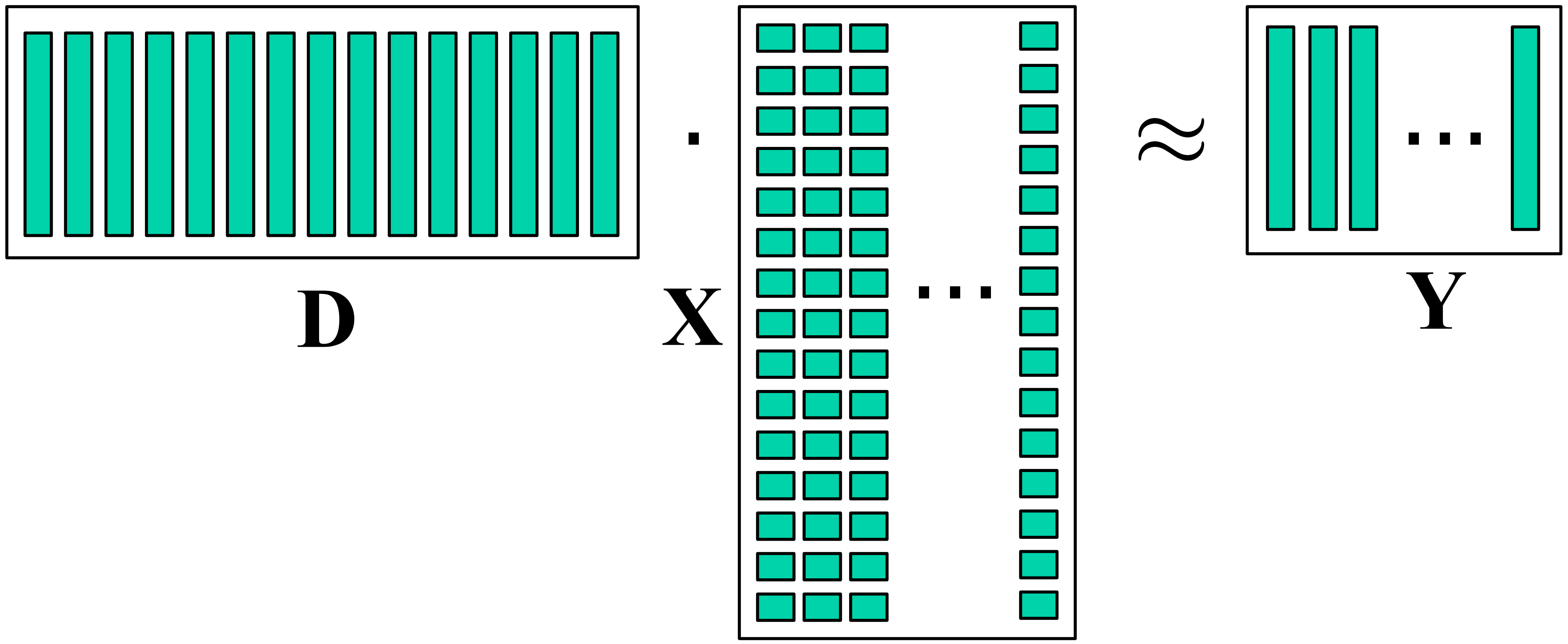


- Error is $\|\mathbf{y} - \mathbf{D}\mathbf{x}\|^2$

Here y is our original signal, D is our dictionary, and x are the coefficients.

K-SVD

- Can extend to many signals

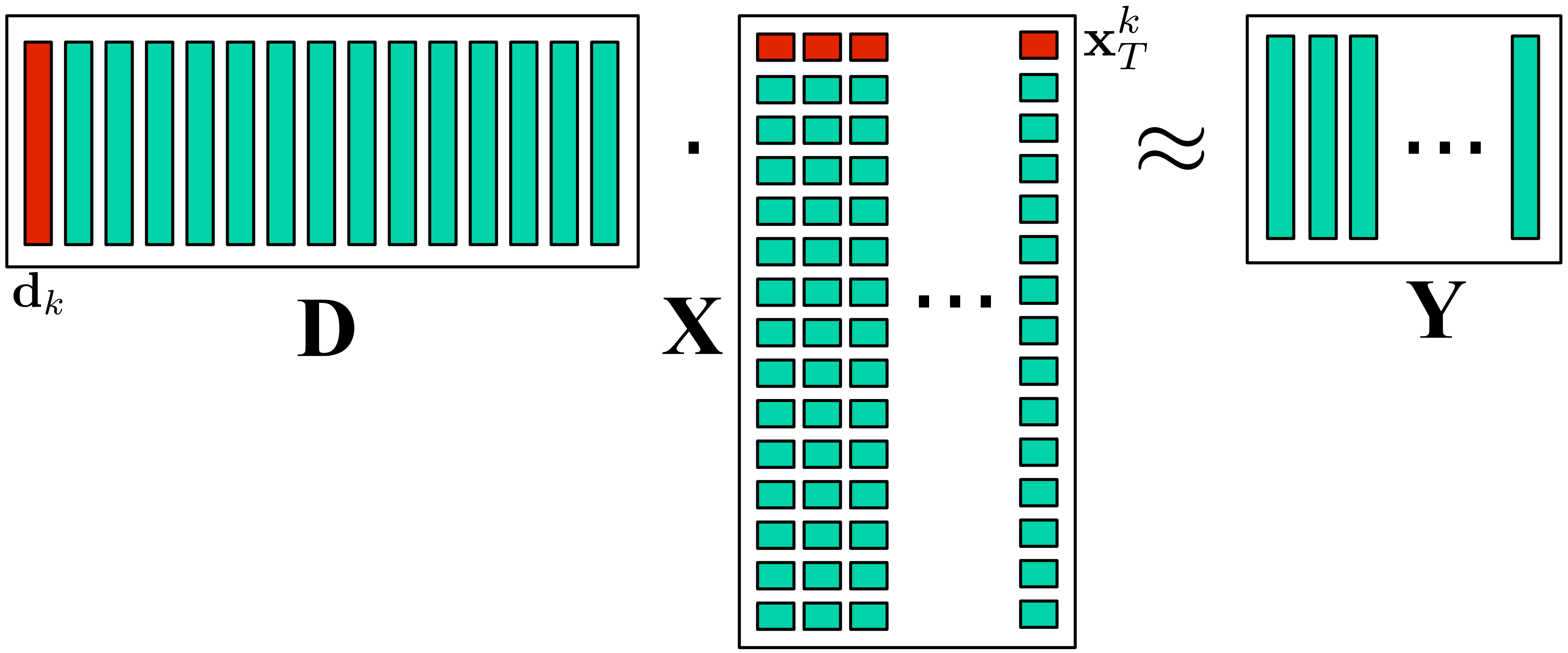


- Error is $\|\mathbf{Y} - \mathbf{DX}\|^2$ (matrix norm)

Here each row in X represents all the coefficients corresponding to a particular atom in D, across all our training signals, and Y is the training set of signals.

K-SVD

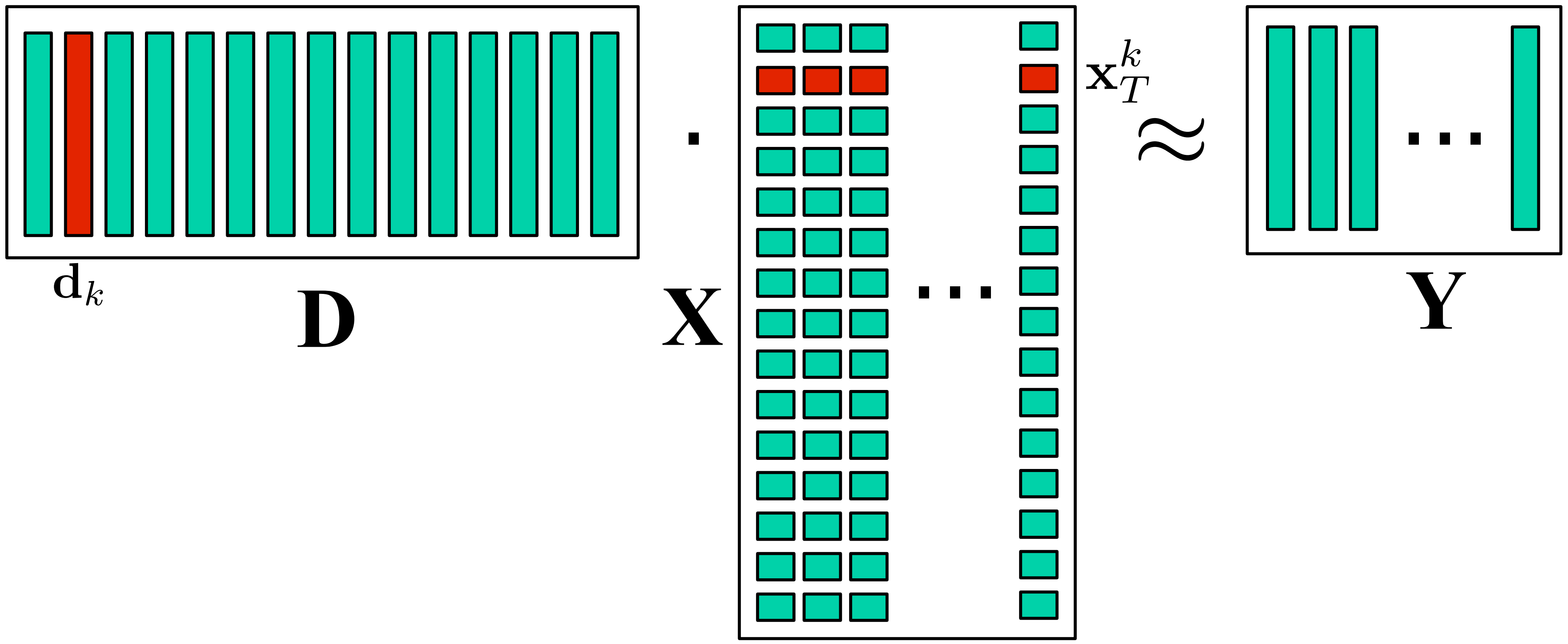
- Idea: iteratively minimize error per atom



- Error is $\|\mathbf{Y} - \mathbf{D}\mathbf{X}\|^2$

K-SVD

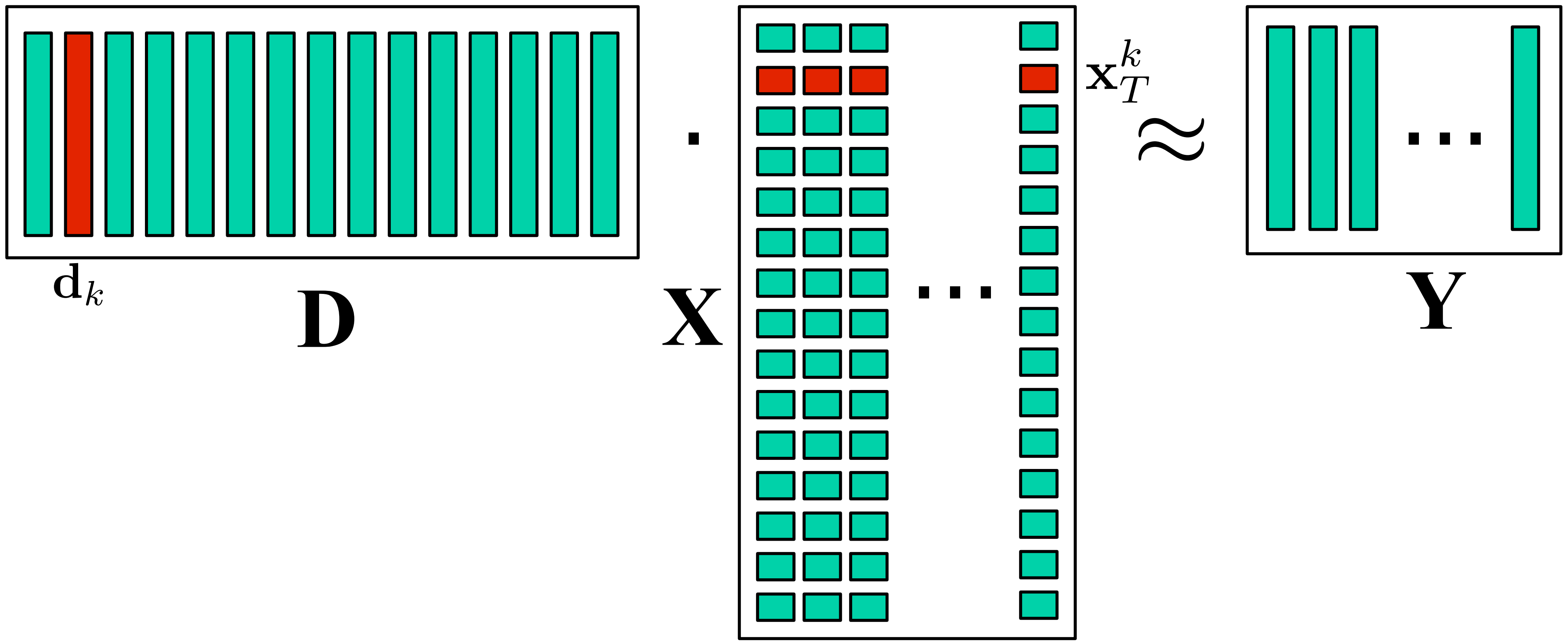
- Idea: iteratively minimize error per atom



- Error is $\|\mathbf{Y} - \mathbf{DX}\|^2$

K-SVD

- Idea: iteratively minimize error per atom

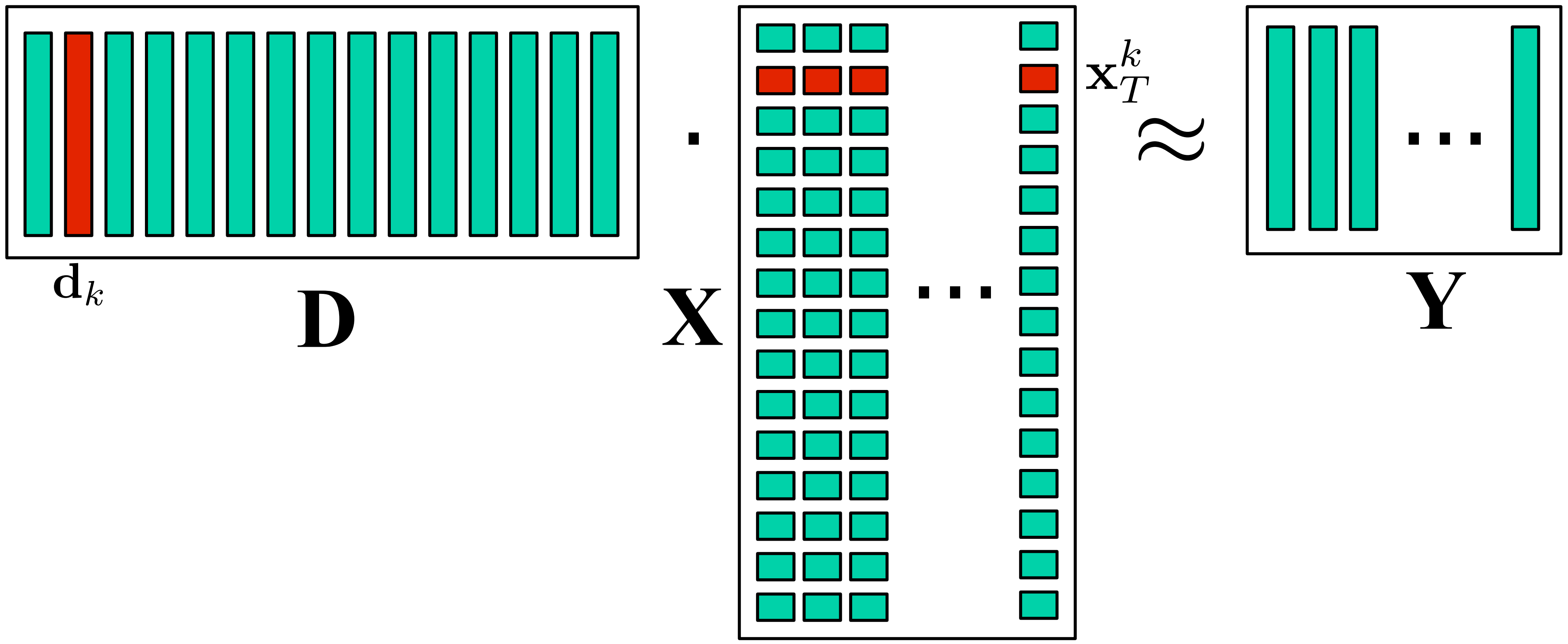


- Error is $\|\mathbf{Y} - \mathbf{DX}\|^2 = \|(\mathbf{Y} - \sum_{j \neq k} d_j x_T^j) - d_k x_T^k\|^2 = \|\mathbf{E}_k - d_k x_T^k\|^2$

We can rewrite our error, pulling out the terms corresponding to the atom we're trying to minimize, and recombining the rest to get an error term E_k.

K-SVD

- Idea: iteratively minimize error per atom



Minimize

- Error is $\|\mathbf{Y} - \mathbf{DX}\|^2 = \|(\mathbf{Y} - \sum_{j \neq k} d_j x_T^j) - d_k x_T^k\|^2 = \|\mathbf{E}_k - d_k x_T^k\|^2$

K-SVD

- How to minimize $\|\mathbf{E}_k - d_k x_T^k\|^2$?
- Idea:
  - decompose $\mathbf{E}_k$
  - use to create new $\mathbf{d}_k$ and $\mathbf{x}_T^k$

K-SVD

- Decompose $\mathbf{E}_k$ using SVD

$$\mathbf{E}_k = \mathbf{UWV}^T$$

  - $\mathbf{U}$, $\mathbf{V}$ orthogonal
  - $\mathbf{W}$ diagonal, large to small magnitude
- Problem: need vectors

It's not clear how this helps us: U, W and V are all matrices.

K-SVD

- Decompose $\mathbf{E}_k$ using SVD

$$\mathbf{E}_k = \mathbf{U}\mathbf{W}\mathbf{V}^T$$

  - $\mathbf{U}$, $\mathbf{V}$ orthogonal
  - $\mathbf{W}$ diagonal, large to small magnitude
- Problem: need vectors

The key is this property of W

K-SVD

- Decompose $\mathbf{E}_k$ using SVD

$$\mathbf{E}_k = \mathbf{UWV}^T$$

  - $\mathbf{U}$, $\mathbf{V}$ orthogonal
  - $\mathbf{W}$ diagonal, large to small magnitude
- Contributes the most to $\mathbf{E}_k$:

$$\mathbf{u}_0 w_{00} \mathbf{v}_T^0$$

So the upper leftmost element of W is the largest in magnitude. And since W is diagonal, we only end up multiplying that element by the first column in U and the first row in V^T. This product will end up being the largest contribution to E_k, so we'll use that to generate our new d and x values. This is called a rank-1 approximation.

K-SVD

- How to minimize $\|\mathbf{E}_k - d_k x_T^k\|^2$?
- Decompose $\mathbf{E}_k$ using SVD

$$\mathbf{E}_k = \mathbf{U}\mathbf{W}\mathbf{V}^T$$

- Use decomposition to minimize

$$\tilde{\mathbf{d}}_k = \mathbf{u}_0$$

$$\tilde{\mathbf{x}}_T^k = w_{00}\mathbf{v}_T^0$$

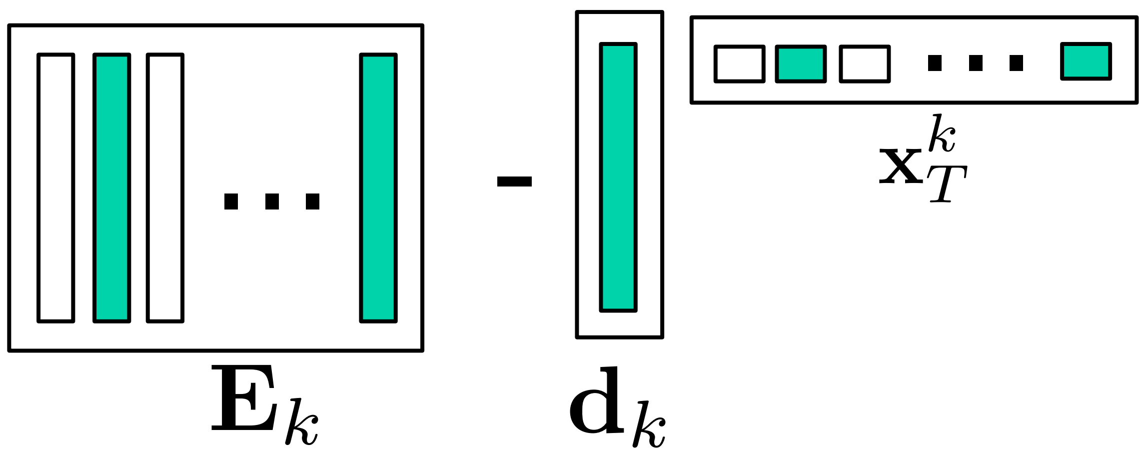
K-SVD

Iterate until desired error level reached:

1. Compute X coefficients for Y via OMP
2. For each column of D/row of X
   a. compute $\mathbf{E}_k$
   b. decompose $\mathbf{E}_k$ using SVD
   c. $\mathbf{d}_k$ becomes $\mathbf{u}_0$
   d. $\mathbf{x}_T^k$ becomes $w_{00}\mathbf{v}_T^0$

K-SVD

- One wrinkle: doesn't converge well
- Solution: collapse $\mathbf{x}_T^k$ to non-zero entries and collapse $\mathbf{E}_k$ to corresponding columns



- Converges much better

K-SVD

Iterate until desired error level reached:

1. Compute X coefficients for Y via OMP
2. For each column of D/row of X
   a. using only non-zero entries of $\mathbf{x}_T^k$, compute $\mathbf{E}_k$
   b. decompose $\mathbf{E}_k$ using SVD
   c. $\mathbf{d}_k$ becomes $\mathbf{u}_0$
   d. $\mathbf{x}_T^k$ becomes $w_{00}\mathbf{v}_T^0$

Summary

- For compression, can use more than ONB
- Use dictionary to get sparse coding
- Use pursuit algorithm to determine coeffs
- Use K-SVD to tailor dictionary to data

References

- Green & Ko, “Frames, Sparsity and New Math for Games”
- Green & Ko, “Orthogonal Matching Pursuit and K-SVD for Sparse Encoding”
- Ko, “Dictionary Learning in Games”
- Rubenstein, Bruckstein, and Elad, “Dictionaries for Sparse Representation Modeling”

Saturday, March 22, 14

The last is a good overall article about using dictionaries for representing signals, and covers a lot more options than presented here. Two of the authors were also the creators of K-SVD.

Contact Info

- jim@essentialmath.com
- Google+: +JimVanVerth
- Twitter: @cthulhim