Least-Square Optimization

Fredo Durand
MIT EECS
6.815/6.865

Project

- Due Nov 19
- small talk, small report

Conjugate gradient

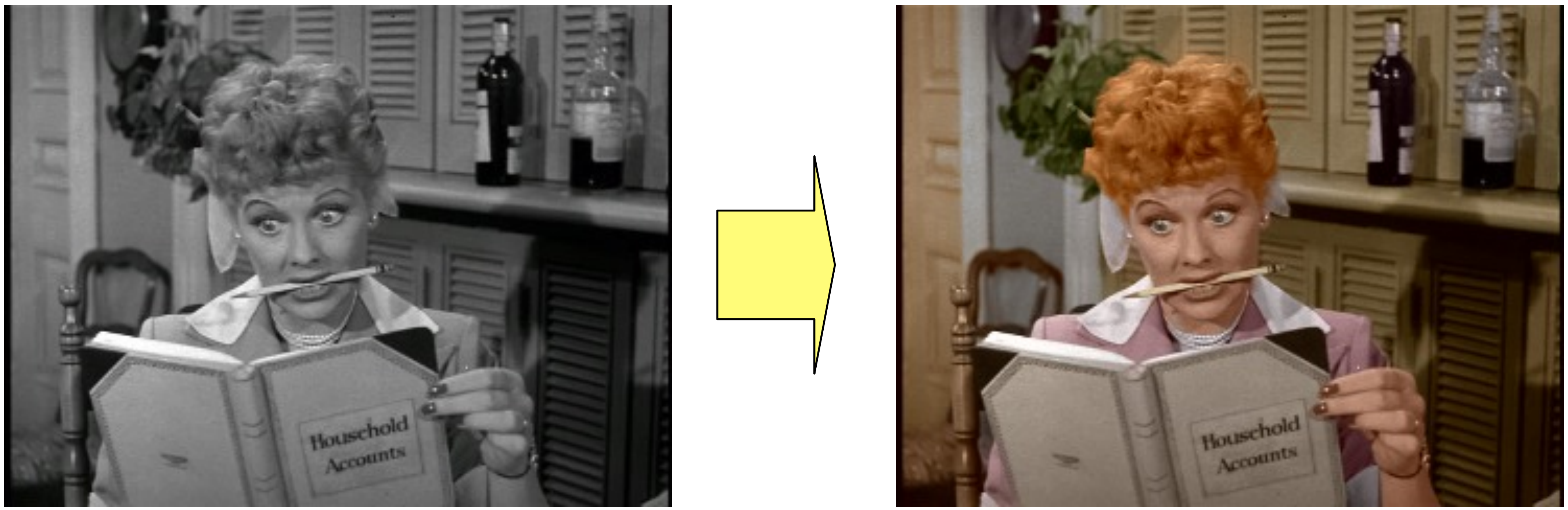
An Introduction to
the Conjugate Gradient Method
Without the Agonizing Pain
Edition $1\frac{1}{4}$

Jonathan Richard Shewchuk
August 4, 1994

- "The Conjugate Gradient Method is the most prominent iterative method for solving sparse systems of linear equations. Unfortunately, many textbook treatments of the topic are written with neither illustrations nor intuition, and their victims can be found to this day babbling senselessly in the corners of dusty libraries. For this reason, a deep, geometric understanding of the method has been reserved for the elite brilliant few who have painstakingly decoded the mumblings of their forebears. Nevertheless, the Conjugate Gradient Method is a composite of simple, elegant ideas that almost anyone can understand. Of course, a reader as intelligent as yourself will learn them almost effortlessly."
  - http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

COLORIZA
TION

Colorization



Colorization: a computer-assisted process of adding color to a monochrome image or movie. (Invented by Wilson Markle, 1970)

Slide courtesy of Anat Levin

Motivation

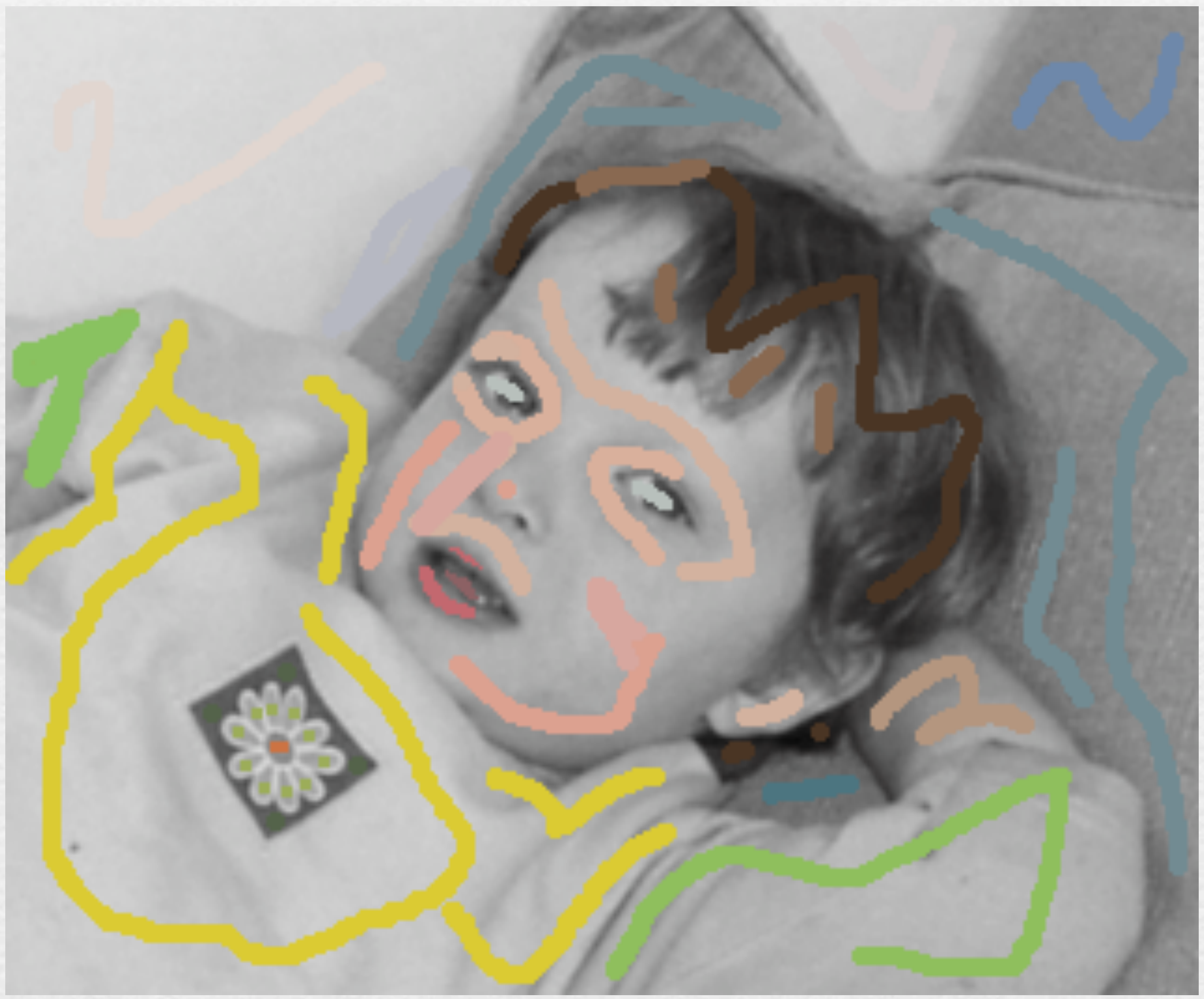
- **Colorizing black and white movies and TV shows**

Earl Glick (Chairman, Hal Roach Studios), 1984:
"You couldn't make Wyatt Earp today for $1 million an episode. But for $50,000 a segment, you can turn it into color and have a brand new series with no residuals to pay"

Slide courtesy of Anat Levin

Colorization using Optimization

- Anat Levin, Dani Lischinski, Yair Weiss
  http://www.cs.huji.ac.il/~yweiss/Colorization/



Input BW image
with user color strokes

Result

Principle

- ✦ Colors vary smoothly
- ✦ Except at strong edges
- ✦ Technical idea:
  - unknowns: pixel color (e.g. UV chrominance in YUV)
  - Energy function that encourages neighboring pixels to have the same value
    - Strength depends on greyscale similarity: the color is more likely the same if the greyscale is the same
  - User stroke = boundary condition
  - It's all a big least square problem

Homogenous smoothness

- Similar to Laplace/Poisson
- Solve for U, minimize

$$J(U) = \sum_r \left[U(r) - \sum_{s \in N(r)} \frac{1}{4} U(s) \right]^2$$

- constrained to boundary conditions at the user's strokes

Non-homogenous weights

- Idea: weight less pixels *s* that are very different from a given center pixel *r*
- The energy definition now varies spatially (non-homogenous)

$$J(U) = \sum_r \left[U(r) - \sum_{s \in N(r)} w_{rs} U(s) \right]^2$$

- where w_{rs} is high when *r* and *s* have similar greyscale values in the input, and low if they are different
- N(r) is the neighborhood of a pixel, e.g. 3x3

Weight/Compatibility functions

- Gaussian on intensity (Y) difference

$$w_{\mathbf{rs}} \propto e^{-(Y(\mathbf{r})-Y(\mathbf{s}))^2/2\sigma_{\mathbf{r}}^2}$$

- Or Normalized correlation

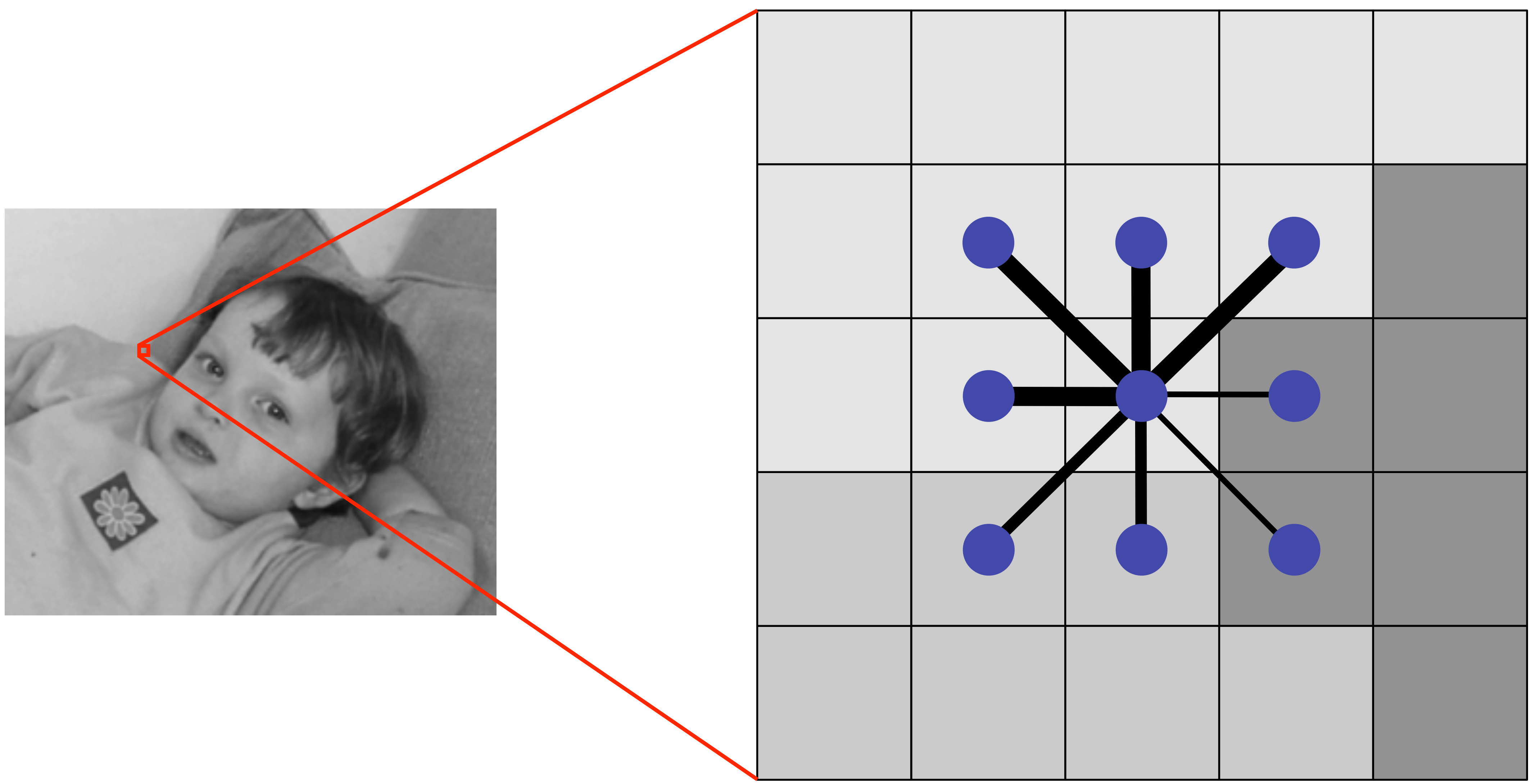
$$w_{\mathbf{rs}} \propto 1 + \frac{1}{\sigma_{\mathbf{r}}^2}(Y(\mathbf{r}) - \mu_{\mathbf{r}})(Y(\mathbf{s}) - \mu_{\mathbf{r}})$$

- where μ is the local mean intensity, σ^2 the variance
- All normalized to sum to 1 in a window

Affinity Functions

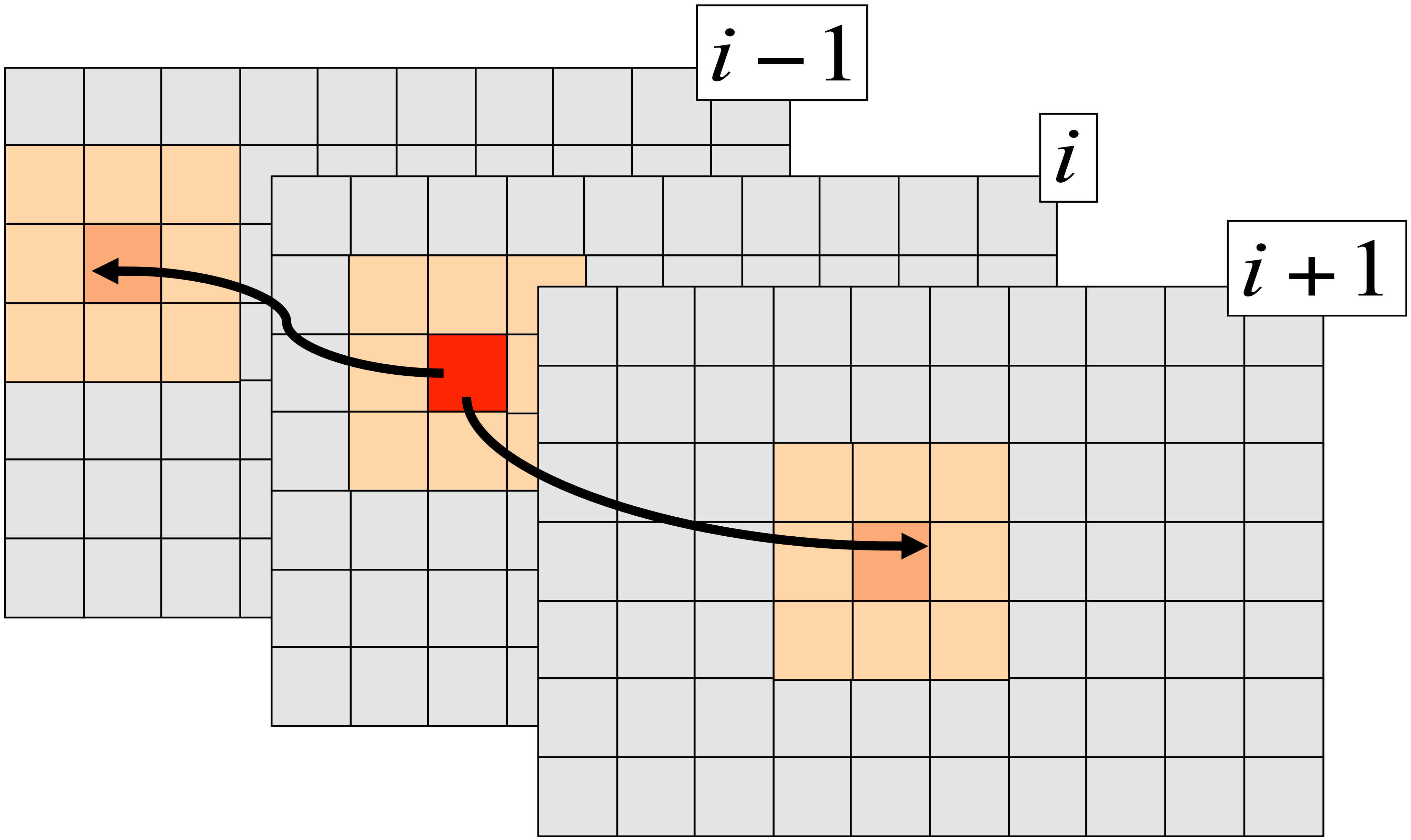
$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/\sigma_r^2}$$

σ_r proportional to local variance



Affinity Functions in Space-Time

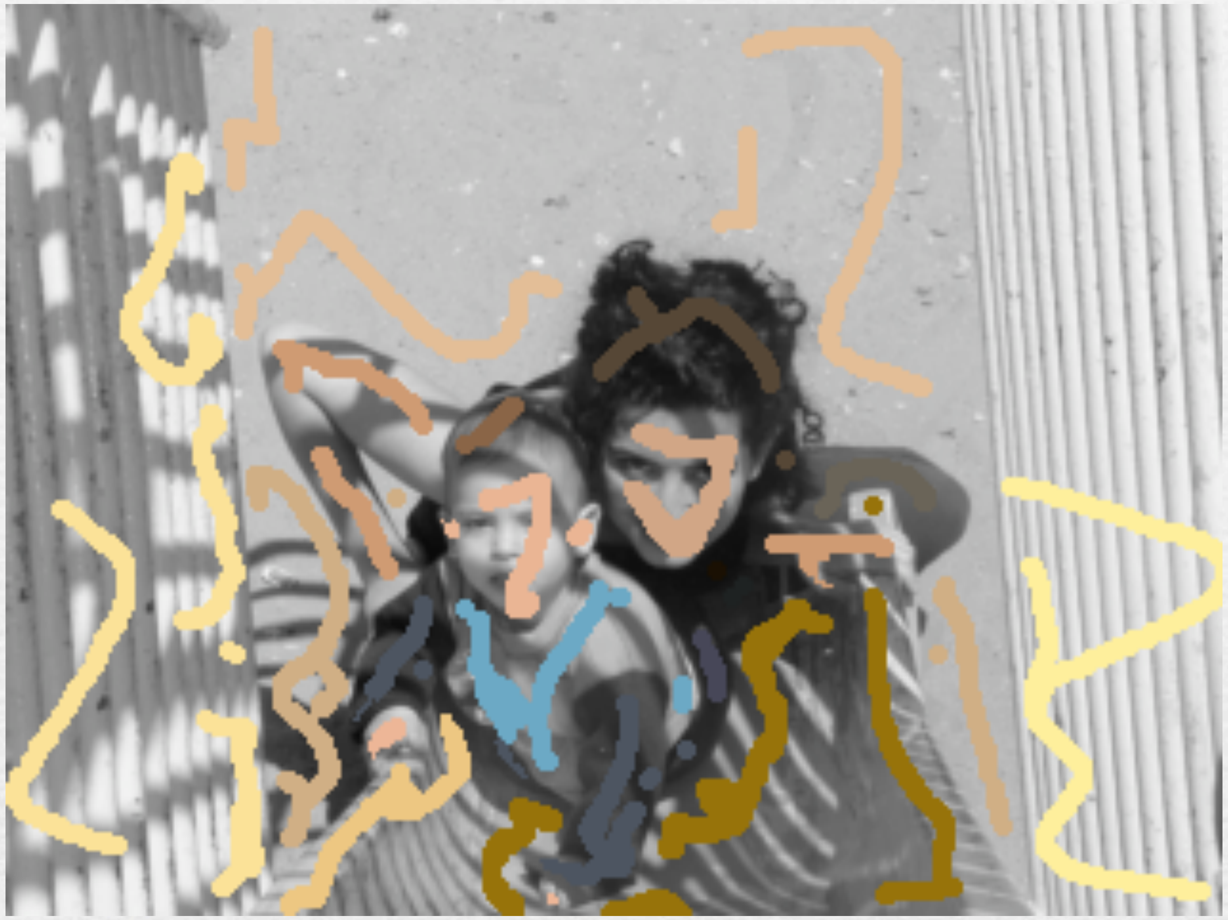
$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/2\sigma_r^2}$$

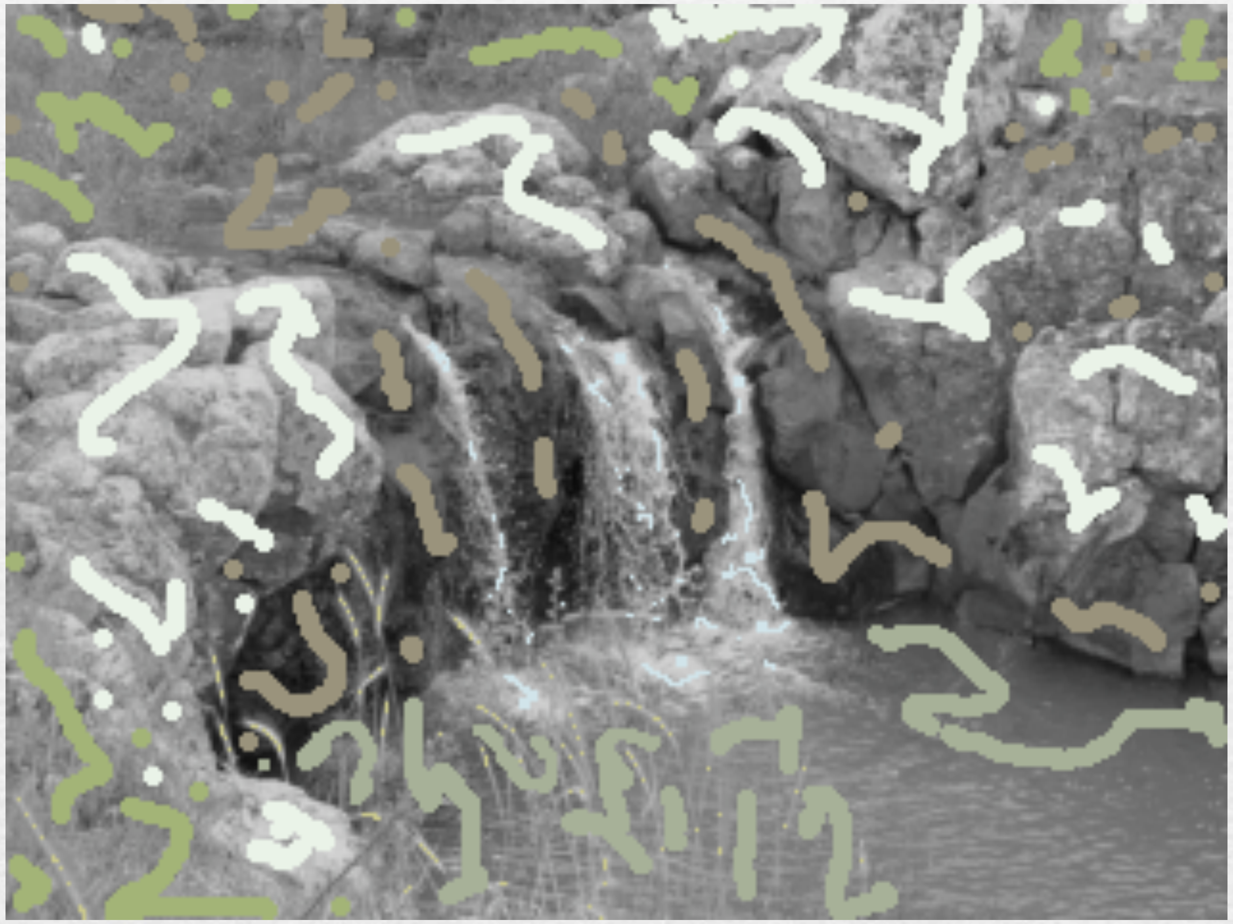


Recap

- Input: black and white image Y
- Non-homogenous least square energy on U, V
  - weight depends on pixel similarity
- User specifies U,V at stroke locations (boundary conditions
- Big linear systen

Results





Progressive refinement

Progressively improving a colorization. The artist begins with the scribbles shown in (a1), which yield the result in (a2). Note that the table cloth gets the same pink color as the girl's dress. Also, some color is bleeding from the cyan pacier onto the wall behind it. By
adding color scribbles on the table cloth and on the wall (b1) these problems are eliminated (b2). Next, the artist decides to change the color
of the beads by sprinkling a few red pixels (c1), yielding the nal result (c2). Note that it was not necessary to mark each and every bead.

Colorization Challenges

Recoloring

Affinity between pixels – based on intensity AND color similarities.

Slide courtesy of Anat Levin

Recoloring

Slide courtesy of Anat Levin

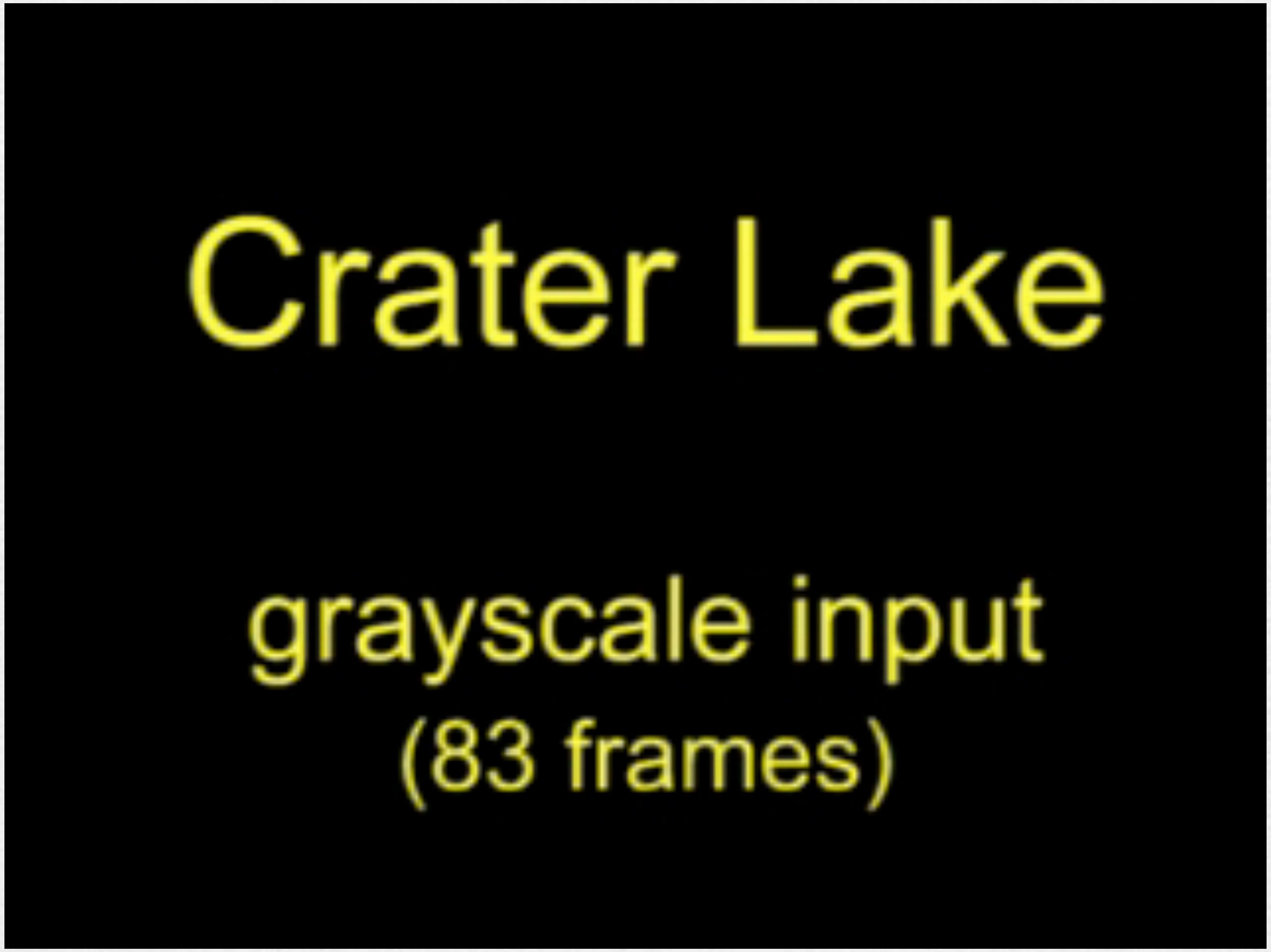
Recoloring

c.f. "Poisson image editing" Perez et al. SIGGRAPH 2003

Slide courtesy of Anat Levin

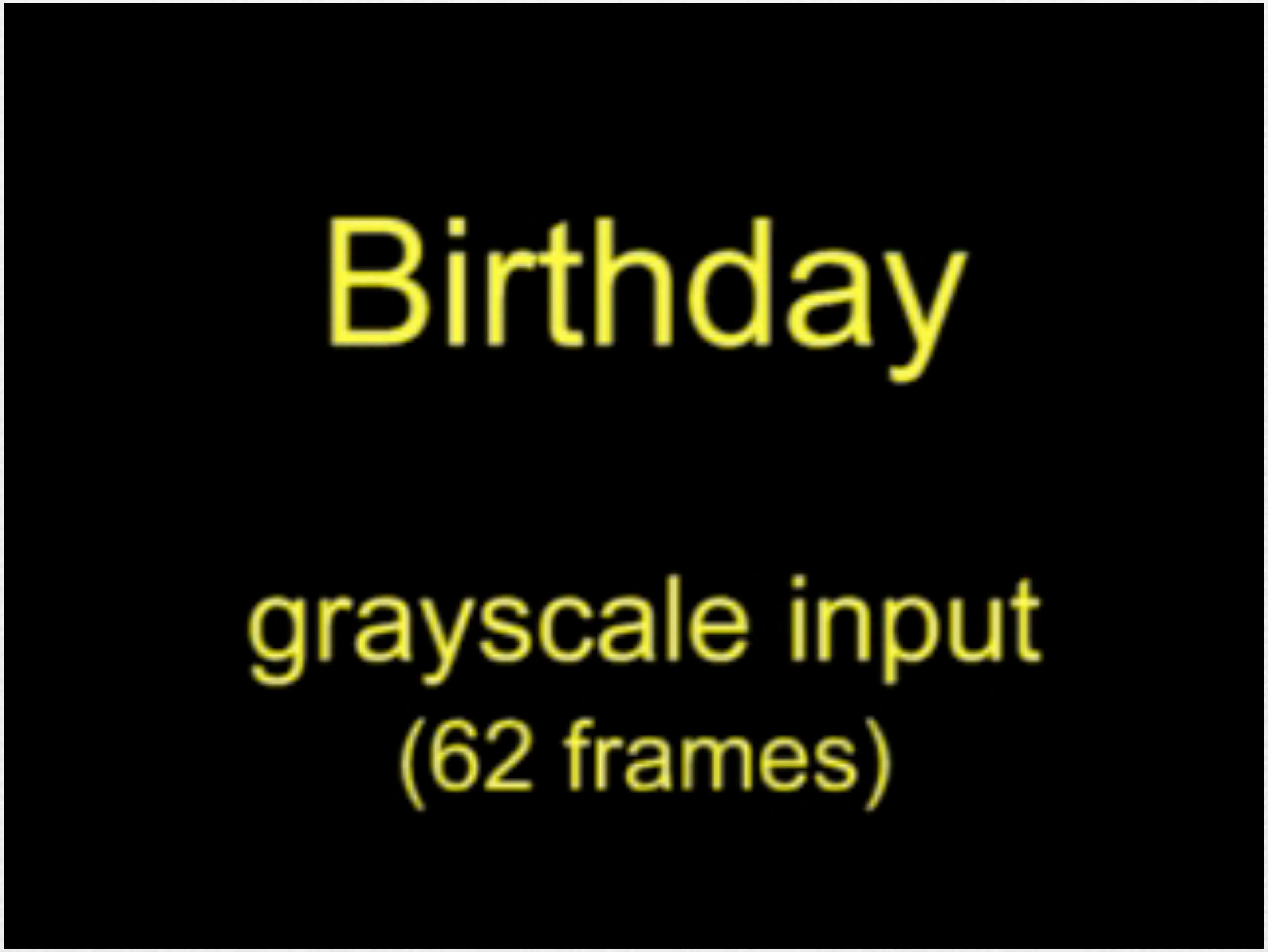
Extension to video

- N(s) now takes motion into account (optical flow)



Extension to video

- N(s) now takes motion into account (optical flow)



Colorizing Video

13 out of 92 frames

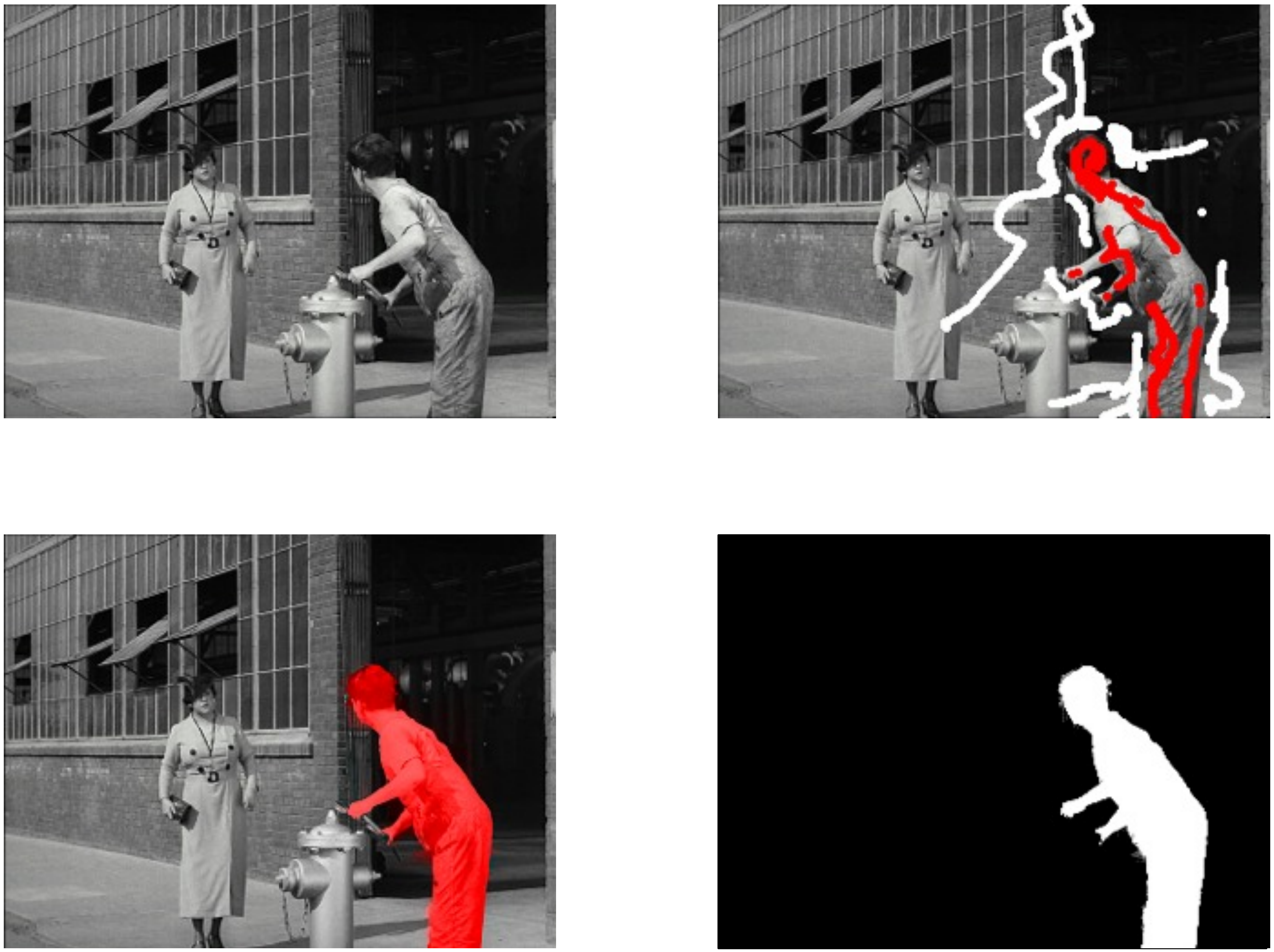
Slide courtesy of Anat Levin

Colorizing Video

16 out of 101 frames

Slide courtesy of Anat Levin

Matting as Colorization



Red channel<->matte

Slide courtesy of Anat Levin

Matting as Colorization

Slide courtesy of Anat Levin

LOCAL EDITS

Interactive Local Adjustment of Tonal Values

- Lischinski, Farbman, Uyttendaele, Szeliski
- http://www.cs.huji.ac.il/~danix/itm/
- 

Interactive Local Adjustment of Tonal Values

- User specifies tone/color manipulations at stroke location
- Interpolated with non-homogenous least squares
  - respects strong edges

LINEAR SYSTEMS

Motivations

- Gaussian elimination is too slow
- Matrix is sparse
  - But usually inverse is dense!
- Applying matrix is relatively cheap

- I will follow Jonathan Shechuck's wonderful exposition of the conjugate gradient method: http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

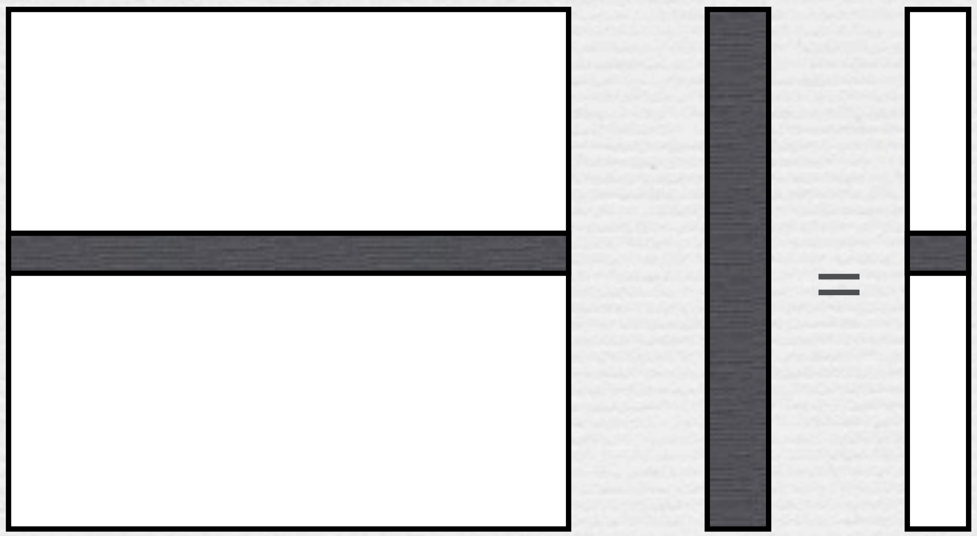
Plan

- Jacobi method
  - Standard but not very effective
- Gradient descent
  - Mostly as a basis for conjugate gradient
- Conjugate gradient
  - Easy and effective
- More advanced stuff
  - preconditioning
  - multigrid
- All at a rather high level
  - Take a course in linear algebra and numerical methods

JACOBI

Jacobi

- The most direct thing you can think of: iteratively solve for each unknown, assuming the other ones are known
- $a_{i0}\, x_0 + a_{i1}\, x_1 + \ldots = bi$
- $xi = -1/a_{ii}(a_{i0}\, x_0 + a_{i1}\, x_1 + \ldots + bi)$
- depending whether you update all at once or one at a time you get Jacobi or Gauss-Seidel



Jacobi - derivation

- $Ax=b$
- Split $A=D+E$ into diagonal D and off-diagonal E
- $(D+E)x=b$
- $Dx=-Ex+b$
- $x=-D^{-1}Ex+D^{-1}b$ (note that D^{-1} is trivial)
- can be written $x=Bx+z$
- leads to iterative procedure: $x_{(i+1)}=Bx_{(i)}+z$

Analysis of Jacobi: eigenvectors

- Iterative methods apply the same matrix over and over
- Logical tool to analyze this: eigenvectors v and eigenvalues λ
  - $Bv=\lambda v$
  - $B^n v=\lambda^n v$
- As a result, the convergence of an iterative technique depends on the largest eigenvector of its update matrix
  - update matrix: $B=D^{-1}E$ for Jacobi
  - geometric series of ratio λ_{max}

Convergence analysis

- Initial vector $x_{(0)}$ expressed in terms of true solution and eigenvalues of update matrix B:
  - $x_{(0)} = x + \Sigma a_j v_j$
- $x_{(1)} = Bx_{(0)} + z$

  $= B\ (\ x + \Sigma a_j v_j\ )\ + z$

  $= (Bx + z)\ + \Sigma a_j B v_j$

  $= \Sigma a_j \lambda_j v_j z$
- Similarly,

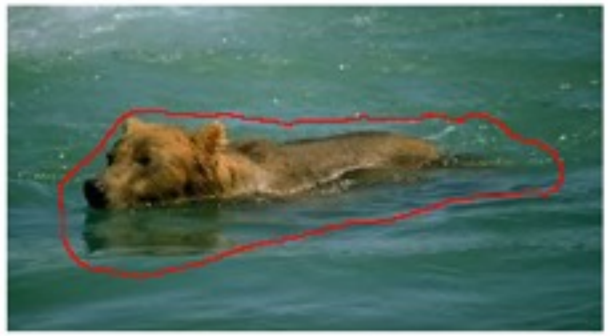
  $x(i) = \Sigma a_j \lambda_j^i v_j z$

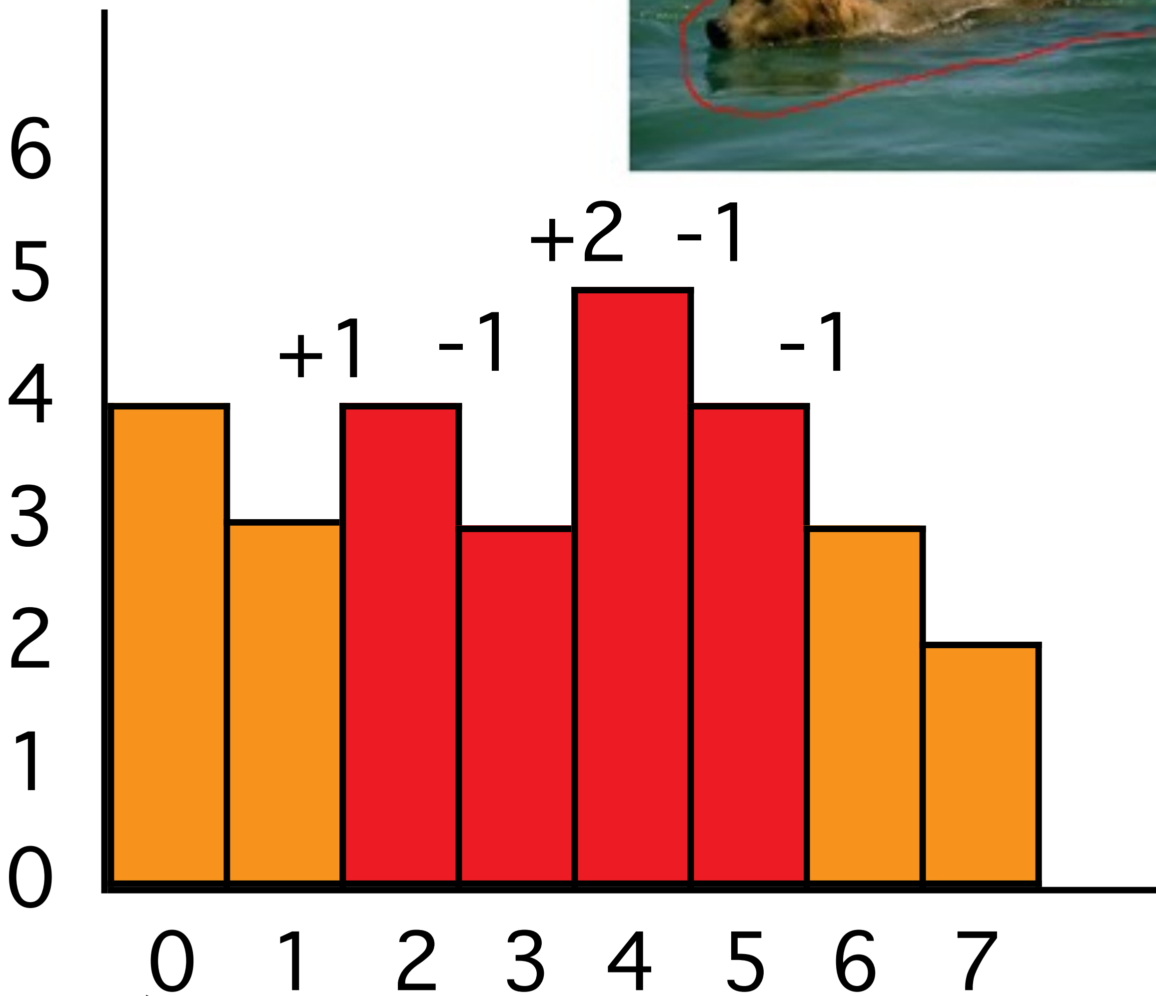
Very common mathematical trick

- Want to understand behavior of some linear update strategy
- Express everything in eigen space
- End up with geometric series of ratio the largest eigenvalue
  - go bad if $|\lambda_{max}| > 1$
  - oscillate if $\lambda_{max} < 0$

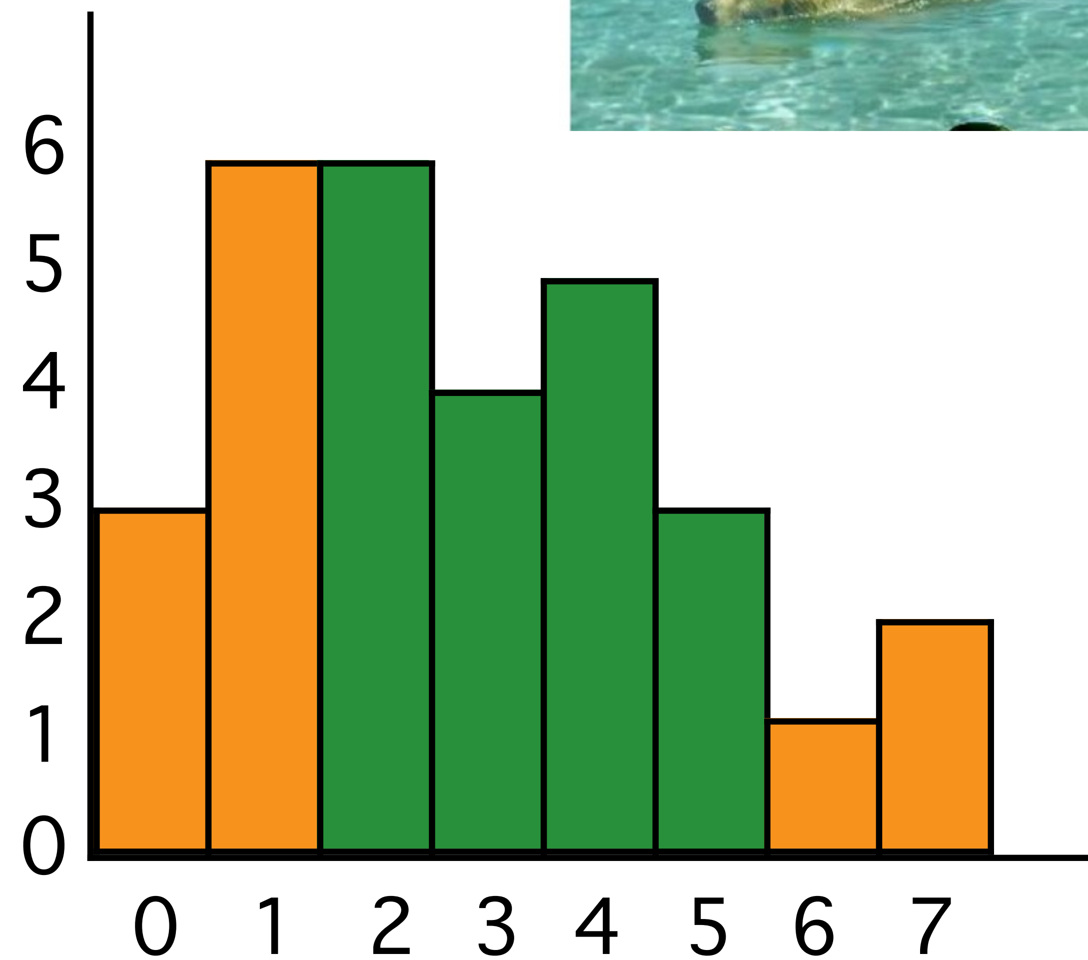
Recall our 1D example

- Copy





to



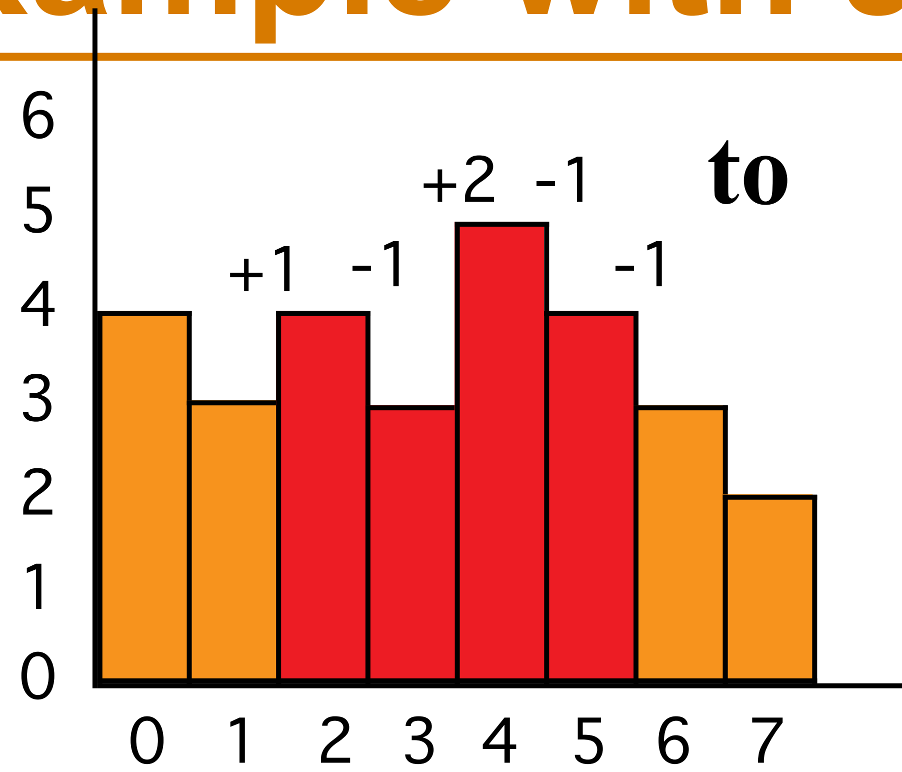
==>

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

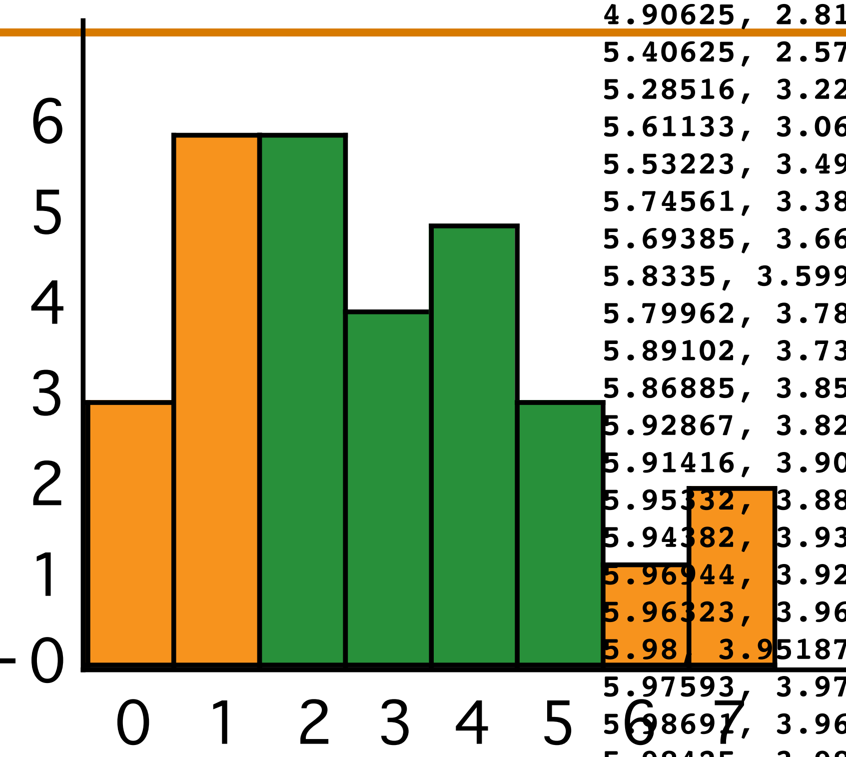
$$\boxed{\begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 5 \\ 3 \end{pmatrix}}$$

1D example with Jacobi

- **Copy**



to




```
4, -1.5, 1.5, 0.5,
3.25, 1.25, 1, 1.25,
4.625, 0.625, 2.75, 1,
4.3125, 2.1875, 2.3125, 1.875,
5.09375, 1.8125, 3.53125, 1.65625,
4.90625, 2.8125, 3.23438, 2.26562,
5.40625, 2.57031, 4.03906, 2.11719,
5.28516, 3.22266, 3.84375, 2.51953,
5.61133, 3.06445, 4.37109, 2.42188,
5.53223, 3.49121, 4.24316, 2.68555,
5.74561, 3.3877, 4.58838, 2.62158,
5.69385, 3.66699, 4.50464, 2.79419,
5.8335, 3.59924, 4.73059, 2.75232,
5.79962, 3.78204, 4.67578, 2.8653,
5.89102, 3.7377, 4.82367, 2.83789,
5.86885, 3.85735, 4.7878, 2.91183,
5.92867, 3.82832, 4.88459, 2.8939,
5.91416, 3.90663, 4.86111, 2.9423,
5.95332, 3.88764, 4.92446, 2.93056,
5.94382, 3.93889, 4.9091, 2.96223,
5.96944, 3.92646, 4.95056, 2.95455,
5.96323, 3.96, 4.9405, 2.97528,
5.98, 3.95187, 4.96764, 2.97025,
5.97593, 3.97382, 4.96106, 2.98382,
5.98691, 3.9685, 4.97882, 2.98053,
5.98425, 3.98287, 4.97451, 2.98941,
5.99143, 3.97938, 4.98614, 2.98726,
5.98969, 3.98879, 4.98332, 2.99307,
5.99439, 3.9865, 4.99093, 2.99166,
5.99325, 3.99266, 4.98908, 2.99546,
5.99633, 3.99117, 4.99406, 2.99454,
5.99558, 3.9952, 4.99285, 2.99703,
5.9976, 3.99422, 4.99611, 2.99643,
5.99711, 3.99686, 4.99532, 2.99806,
5.99843, 3.99622, 4.99746, 2.99766,
5.99811, 3.99794, 4.99694, 2.99873,
5.99897, 3.99752, 4.99834, 2.99847,
5.99876, 3.99865, 4.998, 2.99917,
5.99933, 3.99838, 4.99891, 2.999,
5.99919, 3.99912, 4.99869, 2.99946,
5.99956, 3.99894, 4.99929, 2.99934,
5.99947, 3.99942, 4.99914, 2.99964,
5.99971, 3.99931, 4.99953, 2.99957,
5.99965, 3.99962, 4.99944, 2.99977,
5.99981, 3.99955, 4.99969, 2.99972,
5.99977, 3.99975, 4.99963, 2.99985,
5.99988, 3.9997, 4.9998, 2.99982,
5.99985, 3.99984, 4.99976, 2.9999,
5.99992, 3.99981, 4.99987, 2.99988,
5.9999, 3.99989, 4.99984, 2.99993,
5.99995, 3.99987, 4.99991, 2.99992,
5.99994, 3.99993, 4.9999, 2.99996,
5.99997, 3.99992, 4.99994, 2.99995,
5.99996, 3.99995, 4.99993, 2.99997,
```

$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

System
(I+A')x=b

Iterations:
xn+1=b- A'xn

$$\begin{matrix} 4 \\ -1.5 \\ 1.5 \\ 0.5 \end{matrix} \; +1/2 \; \begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix} \; \text{xn}$$

Plan

- Jacobi method
  - Standard but not very effective
- Gradient descent
  - Mostly as a basis for conjugate gradient
- Conjugate gradient
  - Easy and effective
- More advanced stuff
  - preconditioning
  - multigrid
- All at a rather high level
  - Take a course in linear algebra and numerical methods

GRADIENT DESCENT

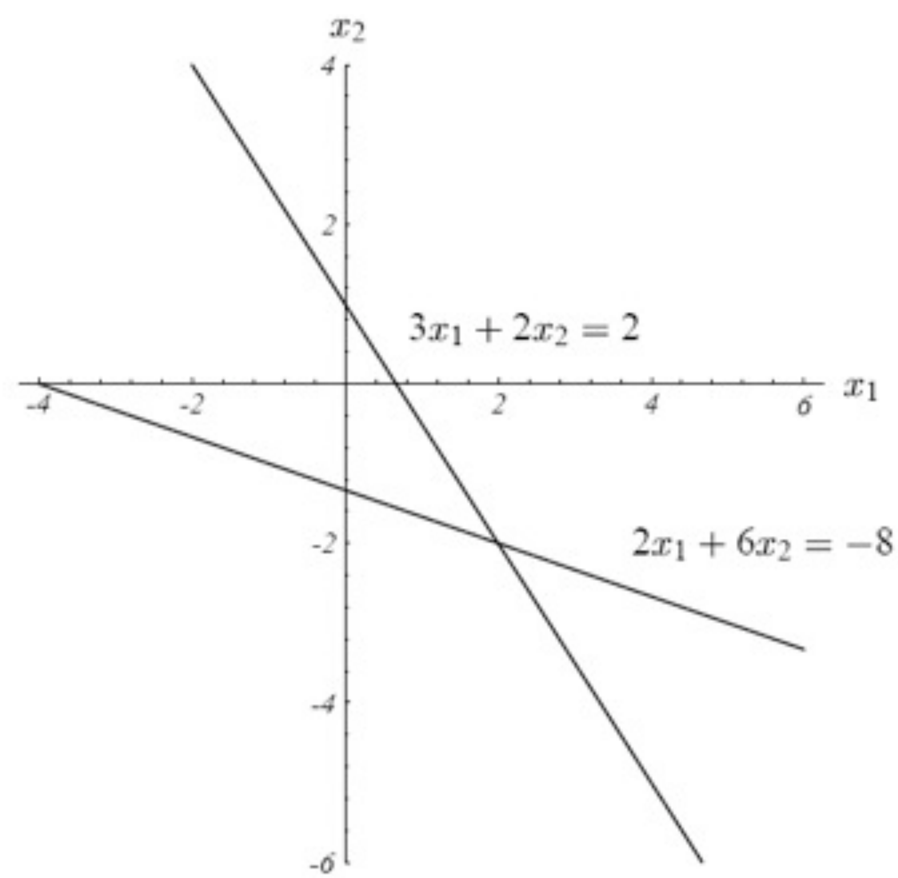
Ax=b

- **A is square, symmetric and positive-definite**
  - When A is dense, you're stuck, use backsubstitution
- **When A is sparse, iterative techniques (such as Conjugate Gradient) are faster and more memory efficient**

- **Simple example:**

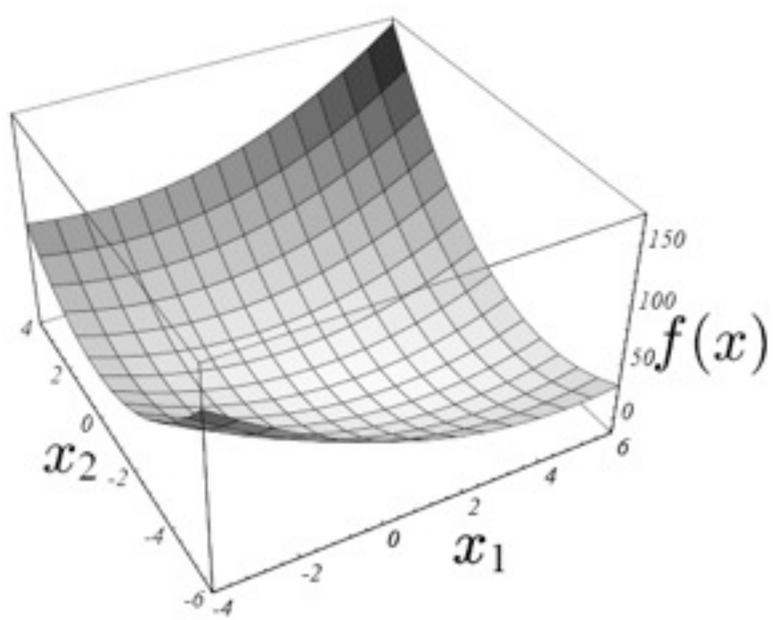
$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

(Yeah yeah, it's not sparse)

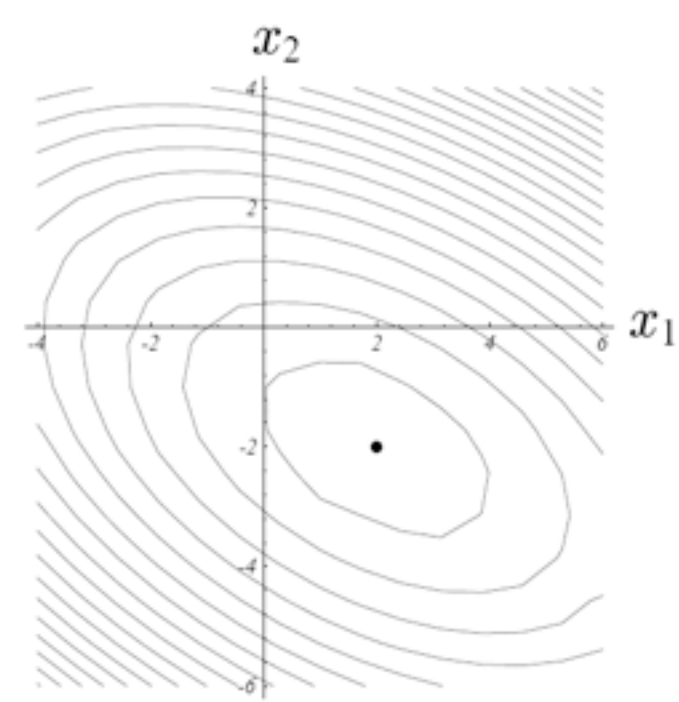


Turn Ax=b into a minimization problem

- **Minimization is more logical to analyze iteration (gradient ascent/descent)**
- **Quadratic form** $f(x) = \frac{1}{2}x^T Ax - b^T x + c$
  - c can be ignored because we want to minimize
- **Intuition:**
  - the solution of a linear system is always the intersection of n hyperplanes
  - Take the square distance to them
  - A needs to be positive-definite so that we have a nice parabola with a minimum, not maximum



Graph of quadratic form $f(x) = \frac{1}{2}x^T Ax - b^T x + c$. The minimum point of this surface is the solution to $Ax = b$.



Contours of the quadratic form. Each ellipsoidal curve has constant $f(x)$.

Gradient of the quadratic form

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}$$

– Not our image gradient!

– Multidimensional gradient (as many dim as rows in matrix)

since

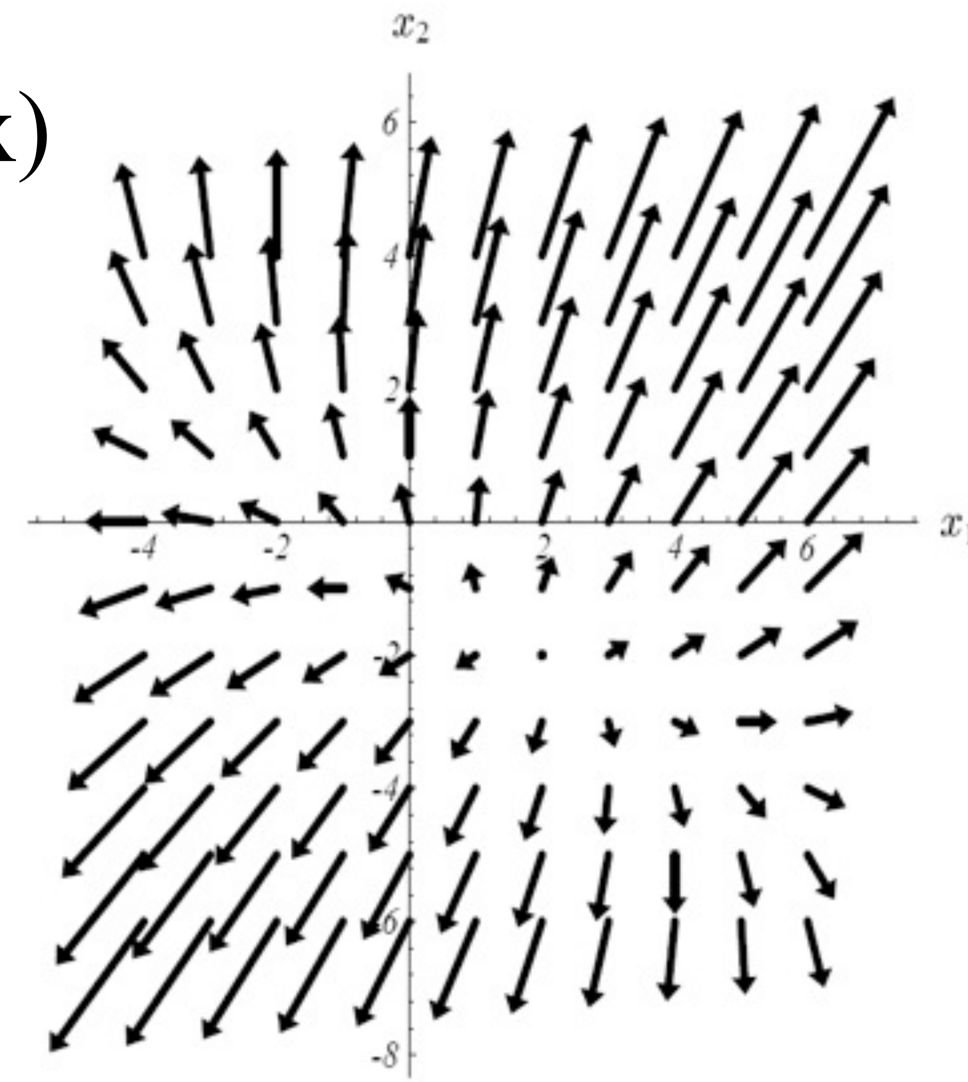
$$f(x) = \frac{1}{2} x^T A x - b^T x + c$$

$$f'(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b.$$

And since A is symmetric

$$f'(x) = Ax - b$$

Not surprising: we turned Ax=b into the quadratic minimization & vice versa



Gradient $f'(x)$ of the quadratic form. For every x, the gradient points in the direction of steepest increase of $f(x)$, and is orthogonal to the contour lines.

(if A is not symmetric, conjugate gradient finds solution for $\frac{1}{2}(A^T + A)x = b$.

New term: Residual

- **How different is the value of an equation from the desired value**
  - Different from error: how far we are from solution
- **At iteration i, we are at a point $x_{(i)}$**
- **Residual $r_{(i)}=b-Ax_{(i)}$**
- **Cool property of quadratic form: residual = - gradient**

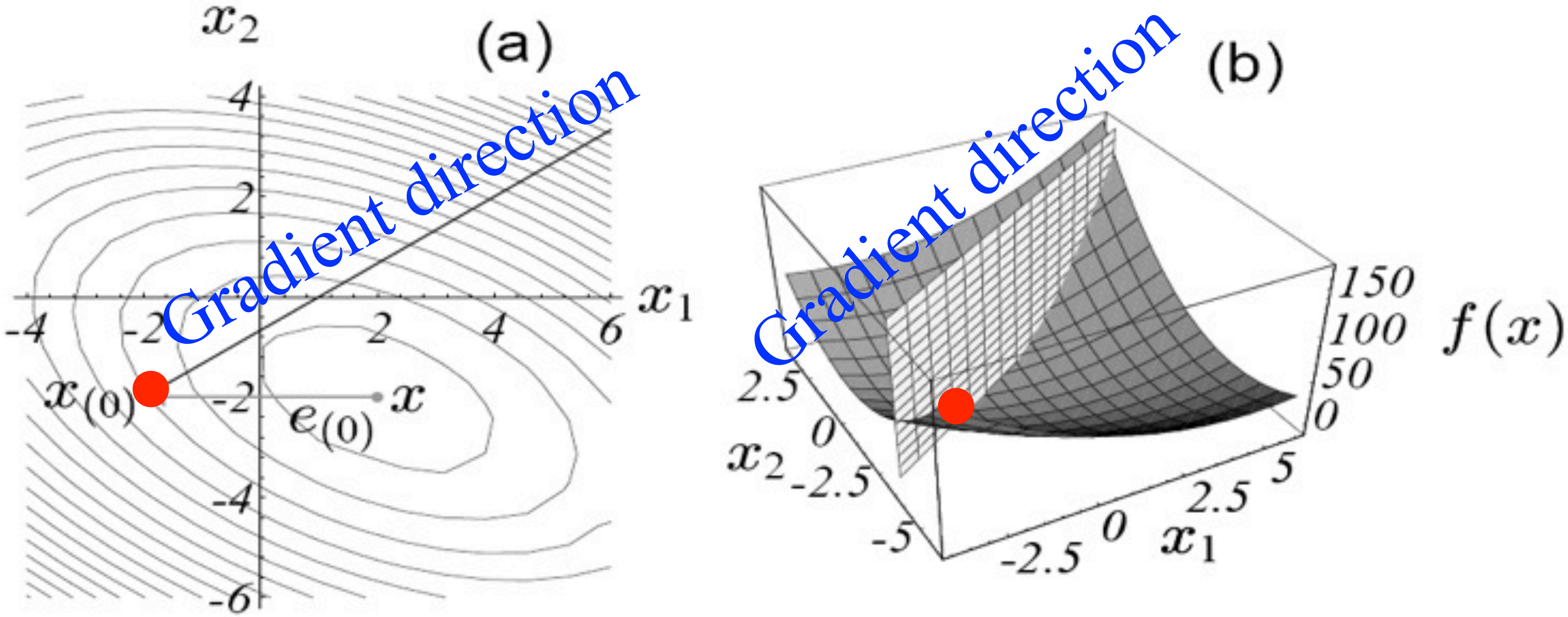
Recap

- **linear least squares <=> linear system**
- **$1/2x^{T}Ax-bx+c$ <=> $Ax=b$**
- **Gradient of quadratic form is $Ax-b$**
  - Residual is negative gradient

Questions?

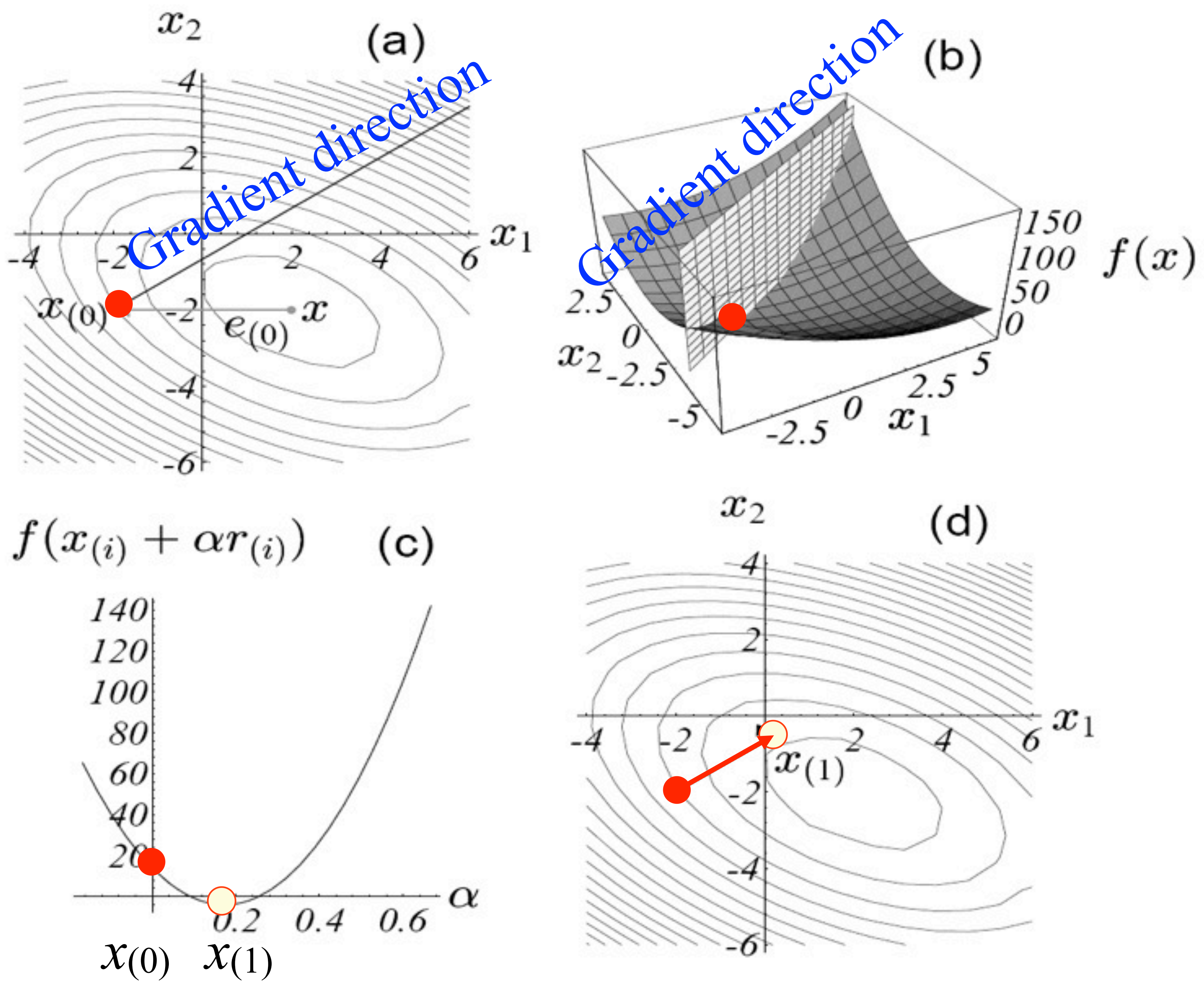
Steepest descent/ascent

- **Pick residual (negative gradient) direction**
  - $Ax_{(i)}$-b



Steepest descent/ascent

- **Pick residual (negative gradient) direction**
  - $-Ax_{(i)}-b$
- **Find optimum in this direction**



Energy along the gradient direction

Optimal along gradient direction

- $\mathbf{x_{(1)}}=\mathbf{x_{(0)}}+\boldsymbol{\alpha}\mathbf{r_{(0)}}$
- **make derivative along direction zero:**
- $$\frac{d}{d\alpha}f(x_{(1)}) = \underbrace{f'(x_{(1)})}_{b-Ax_{(1)}}\underbrace{\frac{dx_{(1)}}{d\alpha}}_{r_{(0)}}$$

$$(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} = 0$$

$$\alpha = \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}}$$

Recap: Gradient Descent

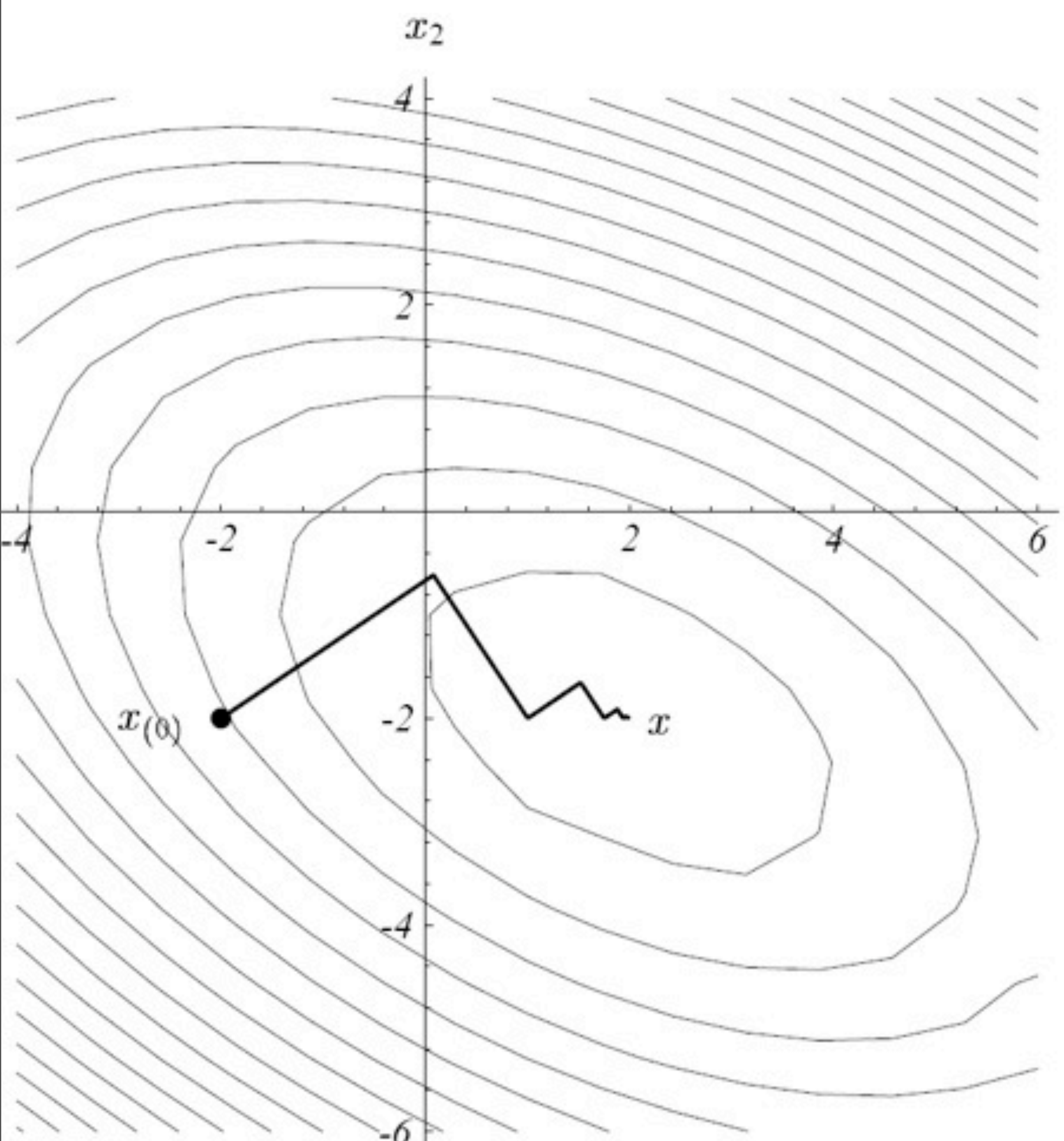
- Residual = - gradient : $r_{(i)}=b-Ax_{(i)}$
- Iteratively walk along residual: $x_{(i+1)}=x_{(i)}+a\ r_{(i)}$
- Find optimal along residual direction:

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

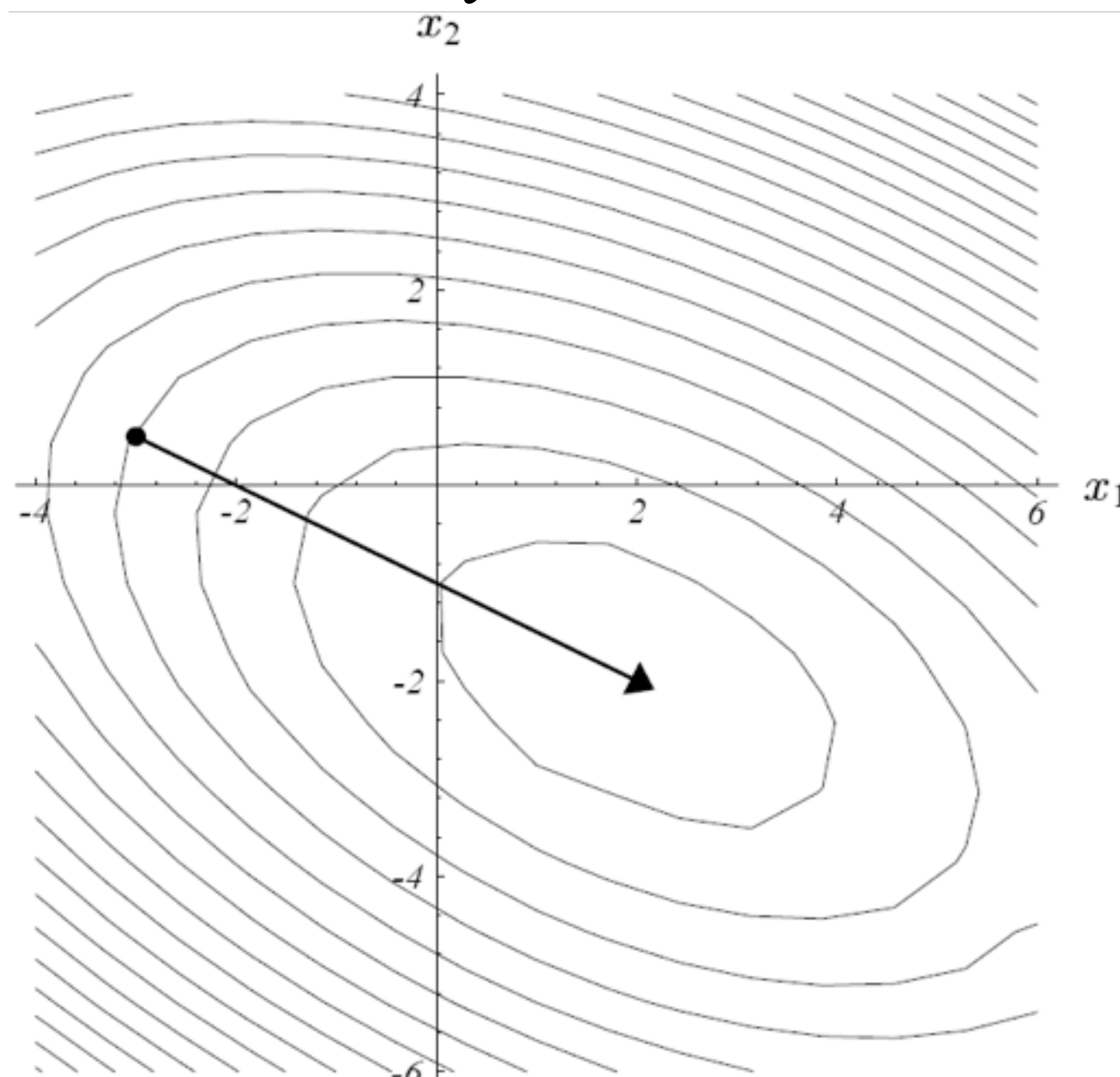
Behavior of gradient descent

- **Zigzag or goes straight depending if we're lucky**
  - Ends up doing multiple steps in the same direction

Unlucky



Lucky

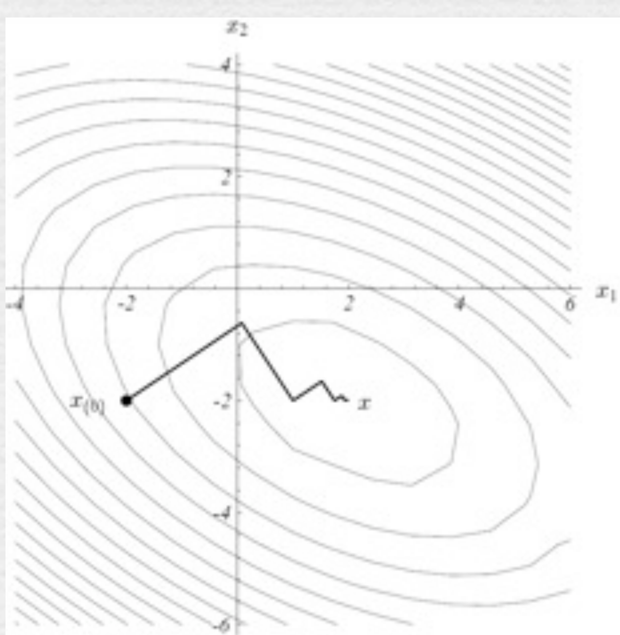


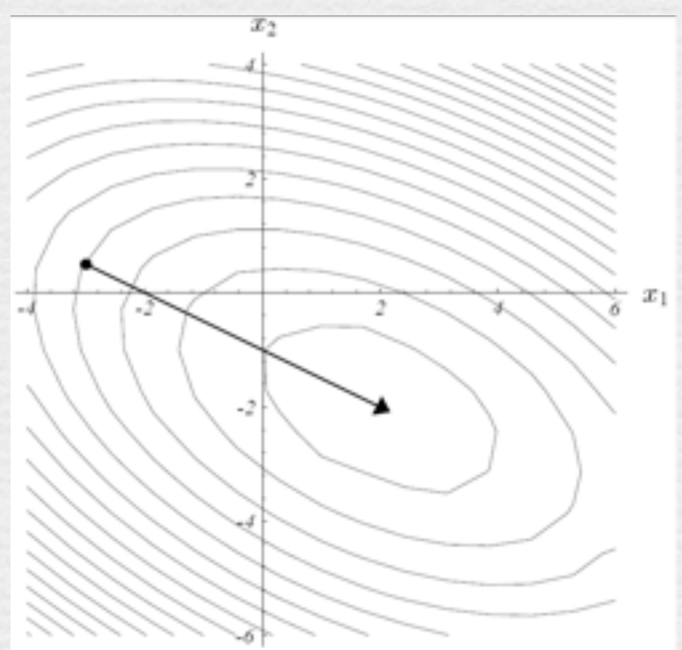
Recap: Gradient Descent

- Residual = - gradient : $r_{(i)} = b - Ax_{(i)}$
- Iteratively walk along residual: $x_{(i+1)} = x_{(i)} + a\, r_{(i)}$
- Find optimal along residual direction:

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

- Behavior: sometimes zigzag, sometimes straight





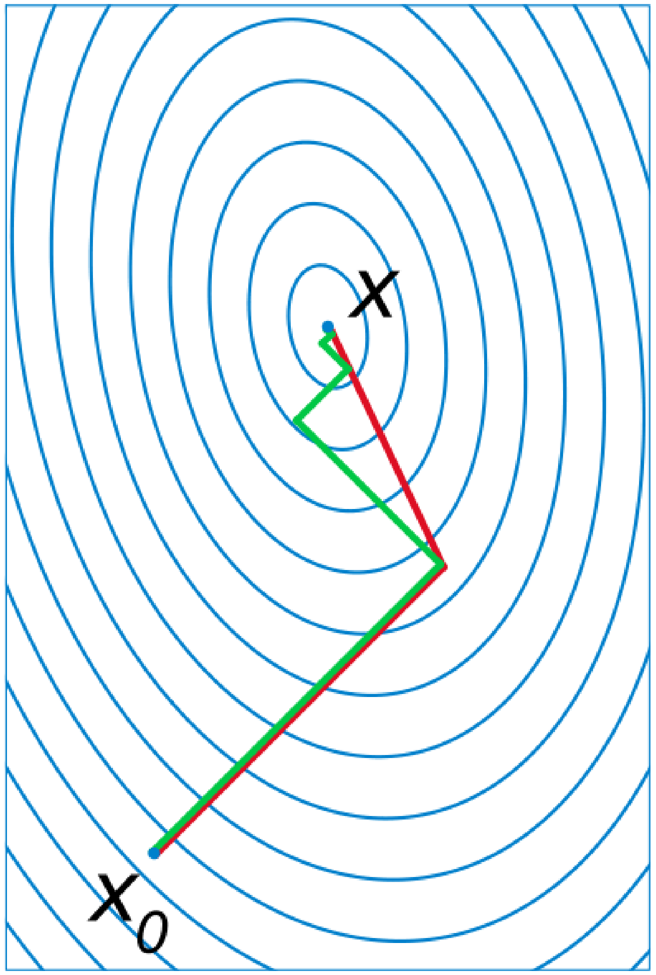
Plan

- Jacobi method
  - Standard but not very effective
- Gradient descent
  - Mostly as a basis for conjugate gradient
- Conjugate gradient
  - Easy and effective
- More advanced stuff
  - preconditioning
  - multigrid
- All at a rather high level
  - Take a course in linear algebra and numerical methods

CONJUGATE GRADIENT

Overview

- **Naive iterative solver: Zigzag**
  - Ends up doing multiple steps in the same direction
- **Conjugate gradient: make sure never go twice in the same direction**
  - Don't go exactly along gradient direction



Green: standard iterations
Red: conjugate gradient

Good news: the code is simple

```
function [x] = conjgrad(A,b,x0)
    r = b - A*x0;
    w = -r;
    z = A*w;
    a = (r'*w)/(w'*z);
    x = x0 + a*w;
    B = 0;
    for i = 1:size(A);
        r = r - a*z;
        if( norm(r) < 1e-10 )
            break;
        B = (r'*z)/(w'*z);
        w = -r + B*w;
        z = A*w;
        a = (r'*w)/(w'*z);
        x = x + a*w;
```

http://en.wikipedia.org/wiki/Image:Conjugate_gradient_illustration.svg

Conjugate gradient

- **Smarter choice of direction**
  - Ideally, step directions should be orthogonal to one another (no redundancy)
  - But tough to achieve
  - Next best thing: make them A-orthogonal (conjugate) That is, orthogonal when transformed by √A
    - Turn the ellipses into circles

$$d_{(i)}^T A d_{(j)} = 0$$

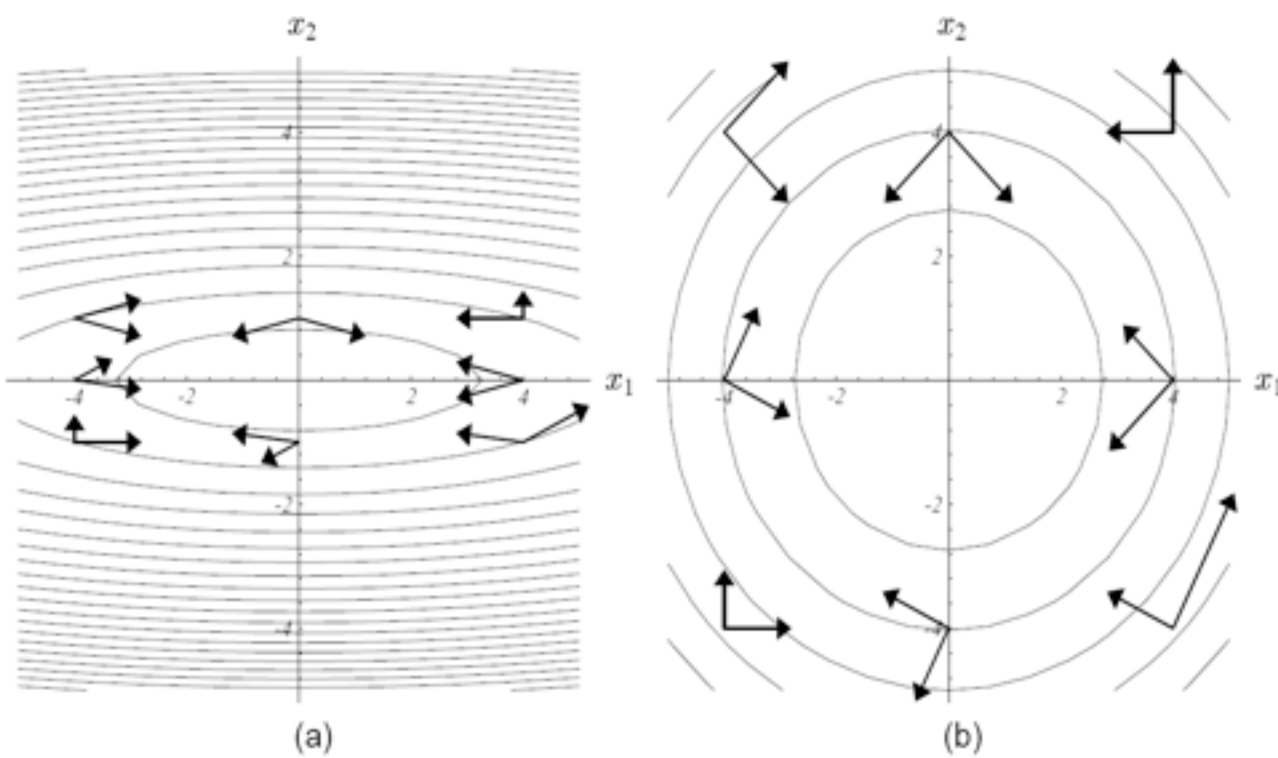


Figure 22: These pairs of vectors are A-orthogonal . . . because these pairs of vectors are orthogonal.

Conjugate gradient

- **For each step:**
  - Take the residual (gradient)
  - Make it A-orthogonal to the previous ones
  - Find minimum along this direction

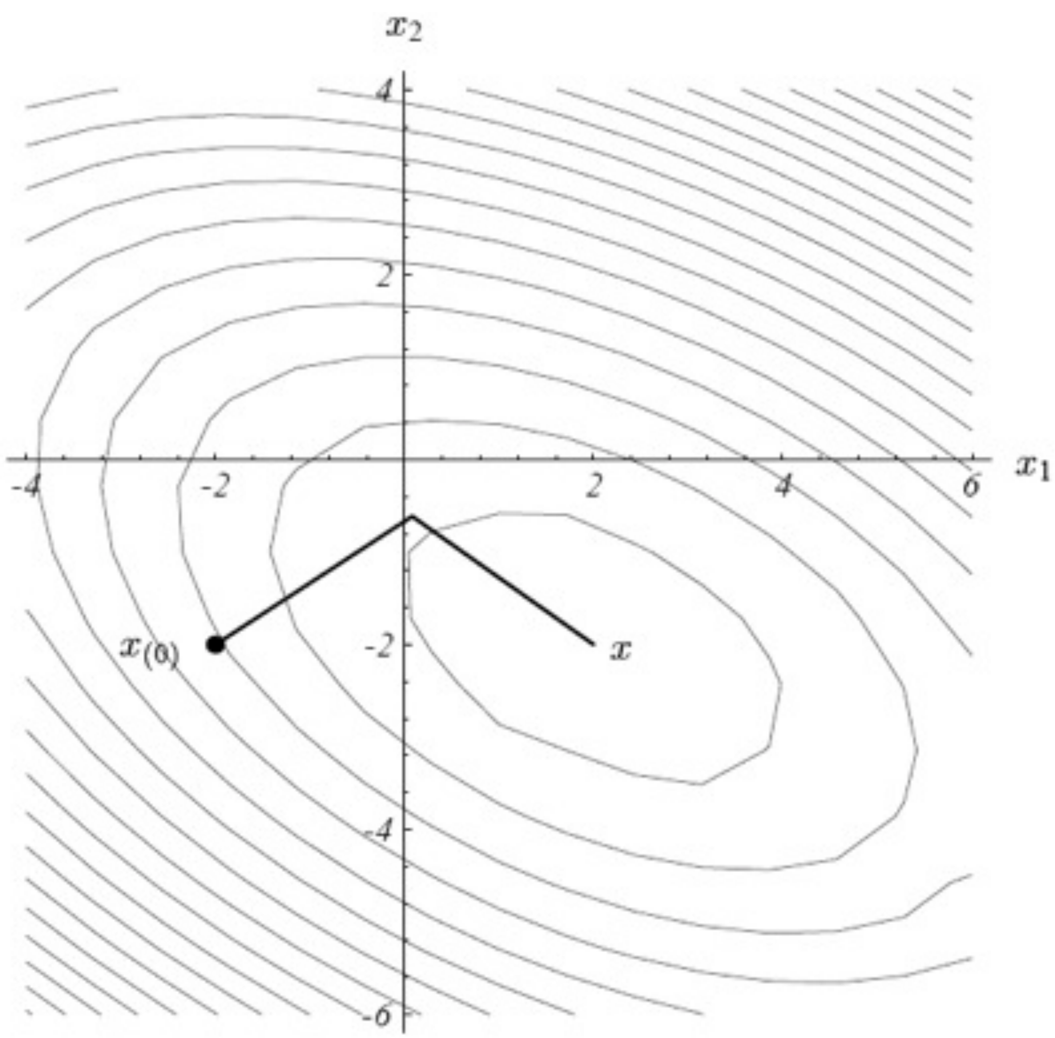
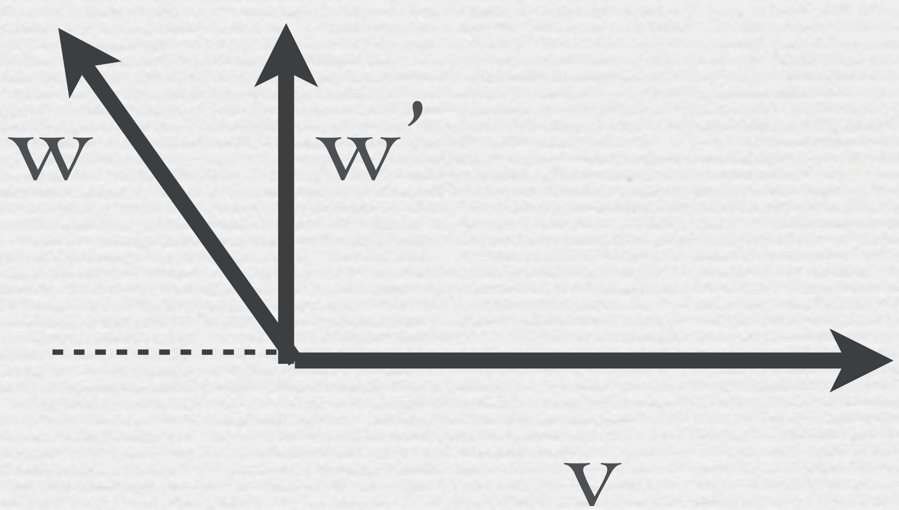


Figure 30: The method of Conjugate Gradients.

How to make vectors orthogonal

- Subtract the non-orthogonal component
- Use dot product
- $w' = v - \gamma w$
- where $\gamma = v^T w / v^T v$
  - denominator needed when v is not unit length
- Gram Schmidt generalizes this to n vectors



Making vectors A-orthogonal

- Start with residual $r_{(i+1)}$
- Perform Gram-Schmidt:
  subtract the non-A-orthogonal component
  - turns out we need to take care of only the previous direction $d_{(i)}$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

  - where

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$$

Similar to previous formula, but involves r's to make orthogonal to d's

  - See Shewchuck's text for derivation

Comparison & recap

- Gradient descent
- $r_{(i)} = b - Ax_{(i)}$
- $x_{(i+1)} = x_{(i)} + \alpha_{(i)} r_{(i)}$

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

- Conjugate gradient
- $r_{(i)} = b - Ax_{(i)}$
- $x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)}$

$$\alpha_{(i)} = \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

Saving some computation

- Bottleneck: matrix-vector products
- Can avoid one:
- $r_{(i+1)}=b-Ax_{(i+1)}$
  $=b-A(x_{(i)}+\alpha_{(i)}d_{(i)})$
  $=(b-Ax_{(i)})+\alpha_{(i)}Ad_{(i)}$
  $=r(i)+\alpha_{(i)}Ad_{(i)}$
- Same as the one needed for $\alpha_{(i)}$

- $r_{(i)}=b-Ax_{(i)}$
- $x_{(i+1)}=x_{(i)}+\alpha_{(i)}d_{(i)}$

$$\alpha_{(i)} = \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T Ad_{(i)}}$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)}d_{(i)}$$

Bells and whistles

- Update $r_{(i)}$ incrementally (previous slide)
  - Compute product Ad once only
  - Pitfall: could drift
  - maybe reset once in a while with full calculation
- Only need to be able to apply matrix A to a vector
  - Often you don't even store A, but use a procedure
- Conjugate gradient is guaranteed to converge in n iterations for n unknowns
  - But we usually want to stop way earlier

The Algorithm

```
function [x] = conjgrad(A,b,x0)      x0: initial guess
    r = b - A*x0;                    residual
    d = -r;                          first iteration: direction = residual
    z = A*d;                         save common term
    a = (r'*d)/(d'*z);               alpha
    x = x0 + a*d;                    update x
    B = 0;                           beta
    for i = 1:size(A);               guaranteed to converge in size(A) steps
        r = r - a*z;                 update residual
        if( norm(r) < 1e-10 )        early termination criterion
            break;
        B = (r'*z)/(d'*z);           beta
        d = -r + B*d;                make residual A-orthogonal
        z = A*d;                     save common term
        a = (r'*d)/(d'*z);           alpha
        x = x + a*d;                 update x
```

Conjugate gradient

- **For each step:**
  - Take the residual (gradient)
  - Make it A-orthogonal to the previous ones
  - Find minimum along this direction
- **Plus life is good:**
  - In practice, you only need the previous one
  - You can show that the new residual $r_{(i+1)}$ is already A-orthogonal to all previous directions d but $d_{(i)}$

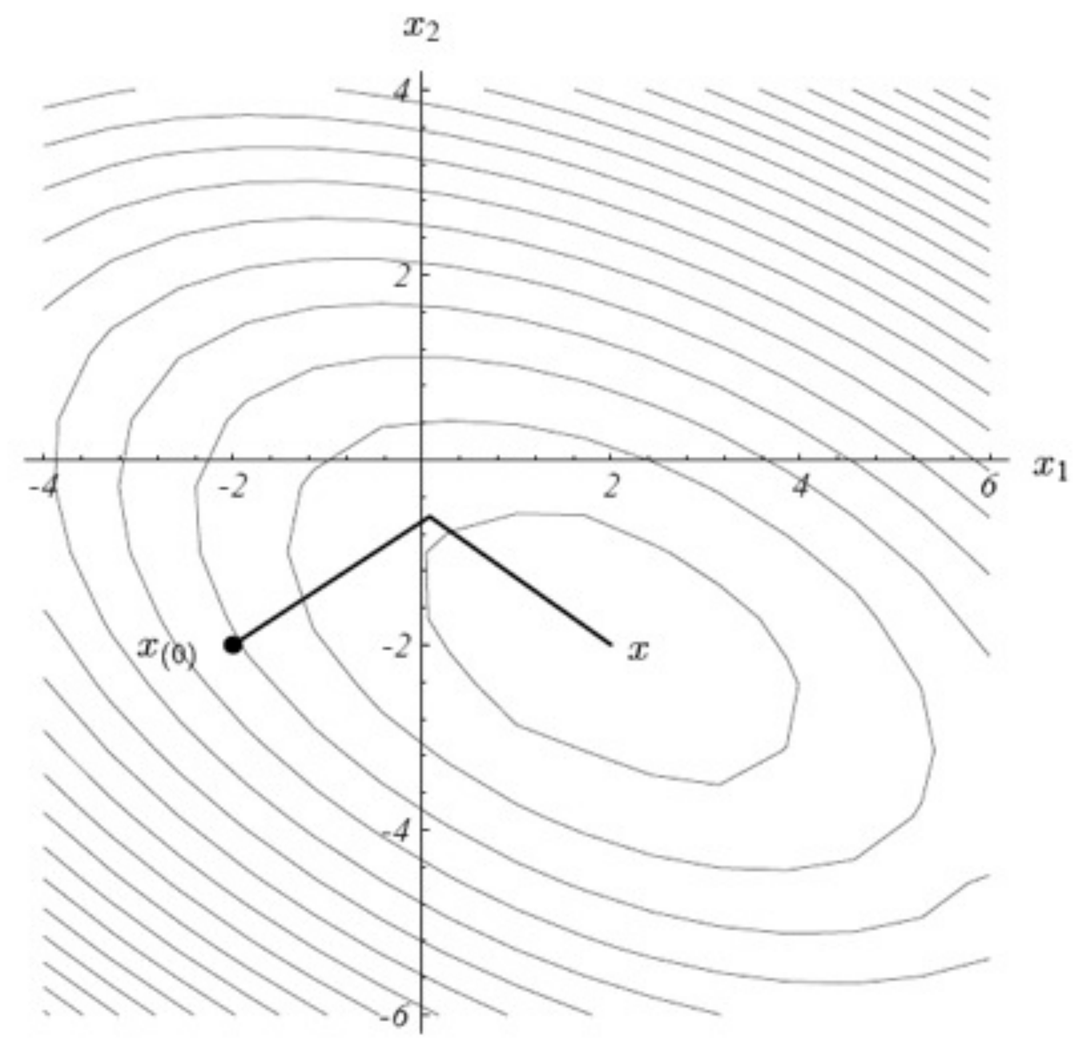
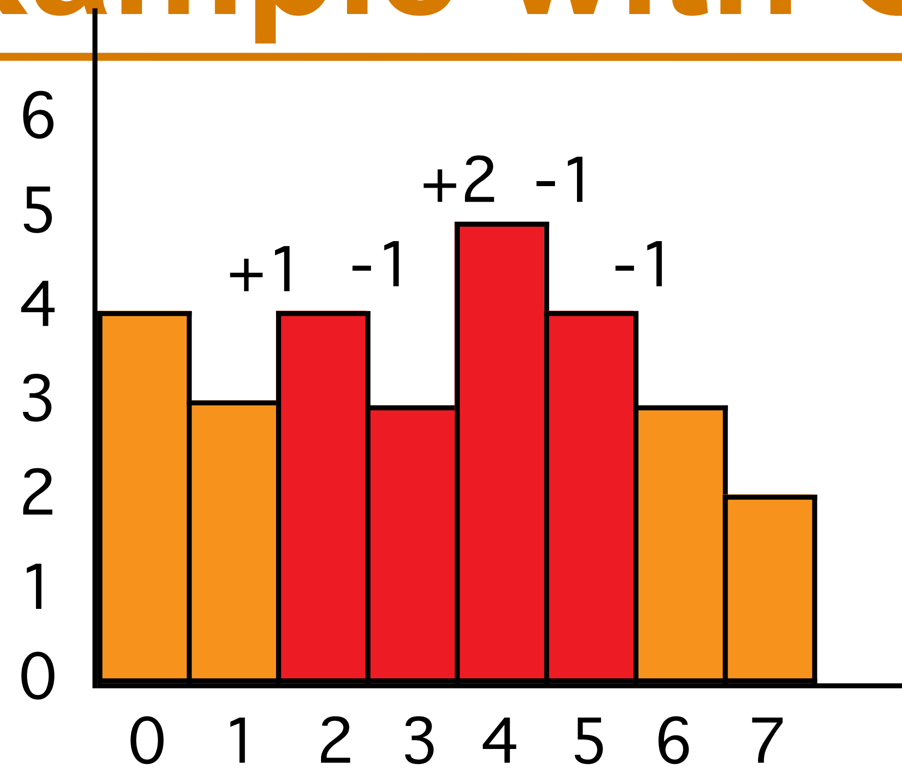
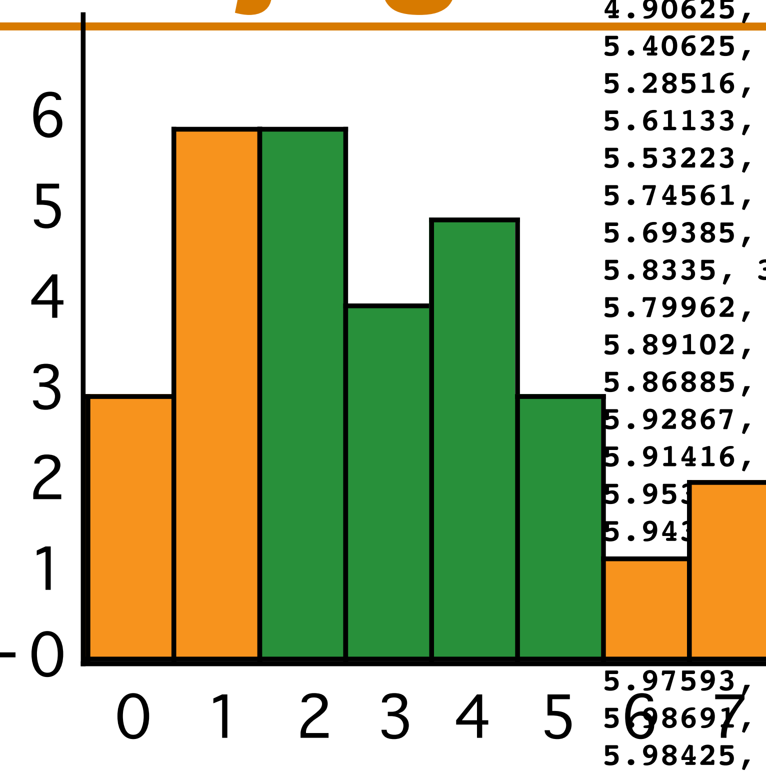


Figure 30: The method of Conjugate Gradients.

1D example with Conjugate





Jacobi:

```
4, -1.5, 1.5, 0.5,
3.25, 1.25, 1, 1.25,
4.625, 0.625, 2.75, 1,
4.3125, 2.1875, 2.3125, 1.875,
5.09375, 1.8125, 3.53125, 1.65625,
4.90625, 2.8125, 3.23438, 2.26562,
5.40625, 2.57031, 4.03906, 2.11719,
5.28516, 3.22266, 3.84375, 2.51953,
5.61133, 3.06445, 4.37109, 2.42188,
5.53223, 3.49121, 4.24316, 2.68555,
5.74561, 3.3877, 4.58838, 2.62158,
5.69385, 3.66699, 4.50464, 2.79419,
5.8335, 3.59924, 4.73059, 2.75232,
5.79962, 3.78204, 4.67578, 2.8653,
5.89102, 3.7377, 4.82367, 2.83789,
5.86885, 3.85735, 4.7878, 2.91183,
5.92867, 3.82832, 4.88459, 2.8939,
5.91416, 3.90663, 4.86111, 2.9423,
5.95 3.88764, 4.92446, 2.93056,
5.94 3.93889, 4.9091, 2.96223,
 3.92646, 4.95056, 2.95455,
 3.96, 4.9405, 2.97528,
 5187, 4.96764, 2.97025,
5.97593, 3.97382, 4.96106, 2.98382,
5.98697, 3.9685, 4.97882, 2.98053,
5.98425, 3.98287, 4.97451, 2.98941,
5.99143, 3.97938, 4.98614, 2.98726,
5.98969, 3.98879, 4.98332, 2.99307,
5.99439, 3.9865, 4.99093, 2.99166,
5.99325, 3.99266, 4.98908, 2.99546,
5.99633, 3.99117, 4.99406, 2.99454,
5.99558, 3.9952, 4.99285, 2.99703,
5.9976, 3.99422, 4.99611, 2.99643,
5.99711, 3.99686, 4.99532, 2.99806,
5.99843, 3.99622, 4.99746, 2.99766,
5.99811, 3.99794, 4.99694, 2.99873,
5.99897, 3.99752, 4.99834, 2.99847,
5.99876, 3.99865, 4.998, 2.99917,
5.99933, 3.99838, 4.99891, 2.999,
5.99919, 3.99912, 4.99869, 2.99946,
5.99956, 3.99894, 4.99929, 2.99934,
5.99947, 3.99942, 4.99914, 2.99964,
5.99971, 3.99931, 4.99953, 2.99957,
5.99965, 3.99962, 4.99944, 2.99977,
5.99981, 3.99955, 4.99969, 2.99972,
5.99977, 3.99975, 4.99963, 2.99985,
5.99988, 3.9997, 4.9998, 2.99982,
5.99985, 3.99984, 4.99976, 2.9999,
5.99992, 3.99981, 4.99987, 2.99988,
5.9999, 3.99989, 4.99984, 2.99993,
5.99995, 3.99987, 4.99991, 2.99992,
5.99994, 3.99993, 4.9999, 2.99996,
5.99997, 3.99992, 4.99994, 2.99995,
5.99996, 3.99995, 4.99993, 2.99997,
```

Conjugate gradient:

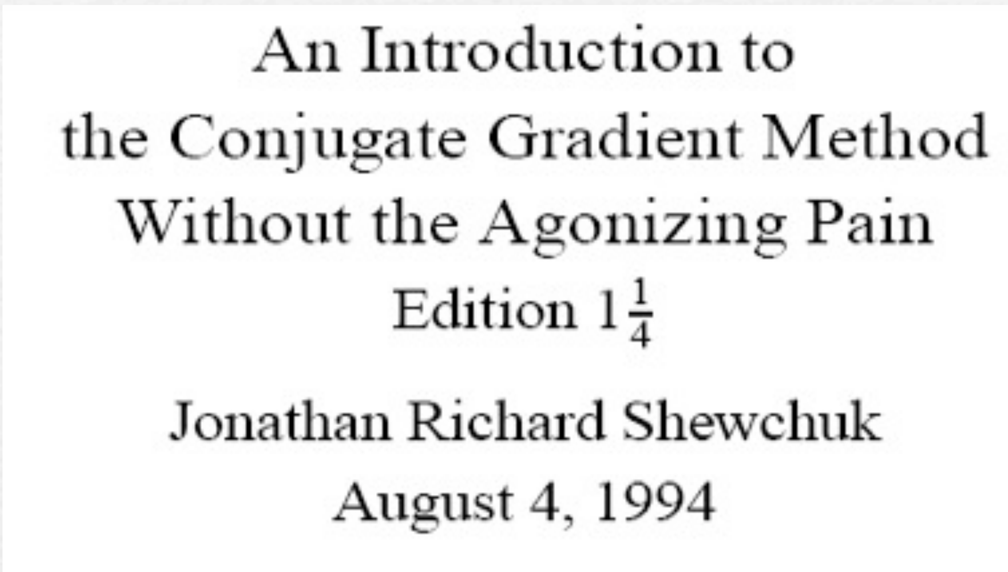
2.9381	-1.1018	1.1018	0.3673
5.2027	1.5933	1.6370	1.8617
6.1724	3.9337	4.3370	2.0983
6.0000	4.0000	5.0000	3.0000

When use Conjugate Gradient?

- Ax=b
- A is positive definite
- A is sparse

- Disadvantage compared to factorization +backsubstitution:
  - you start from scratch for every new b
  - error if not converged
- Bottomline: use \ when you can afford it, conjugate gradient otherwise

The two references



An Introduction to
the Conjugate Gradient Method
Without the Agonizing Pain
Edition $1\frac{1}{4}$

Jonathan Richard Shewchuk
August 4, 1994

- http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

- Iterative methods for sparse linear systems (2nd edition)
Yousef Saad
- http://www-users.cs.umn.edu/~saad/books.html
- http://books.google.com/books?id=Uoe7xBOhS5AC&dq=saad+iterative&printsec=frontcover&source=in&hl=en&ei=Y2GtSerjMdW5twft78CHBg&sa=X&oi=book_result&resnum=11&ct=result#PPR5,M1

Plan

- Jacobi method
  - Standard but not very effective
- Gradient descent
  - Mostly as a basis for conjugate gradient
- Conjugate gradient
  - Easy and effective
- More advanced stuff
  - preconditioning
  - multigrid
- All at a rather high level
  - Take a course in linear algebra and numerical methods

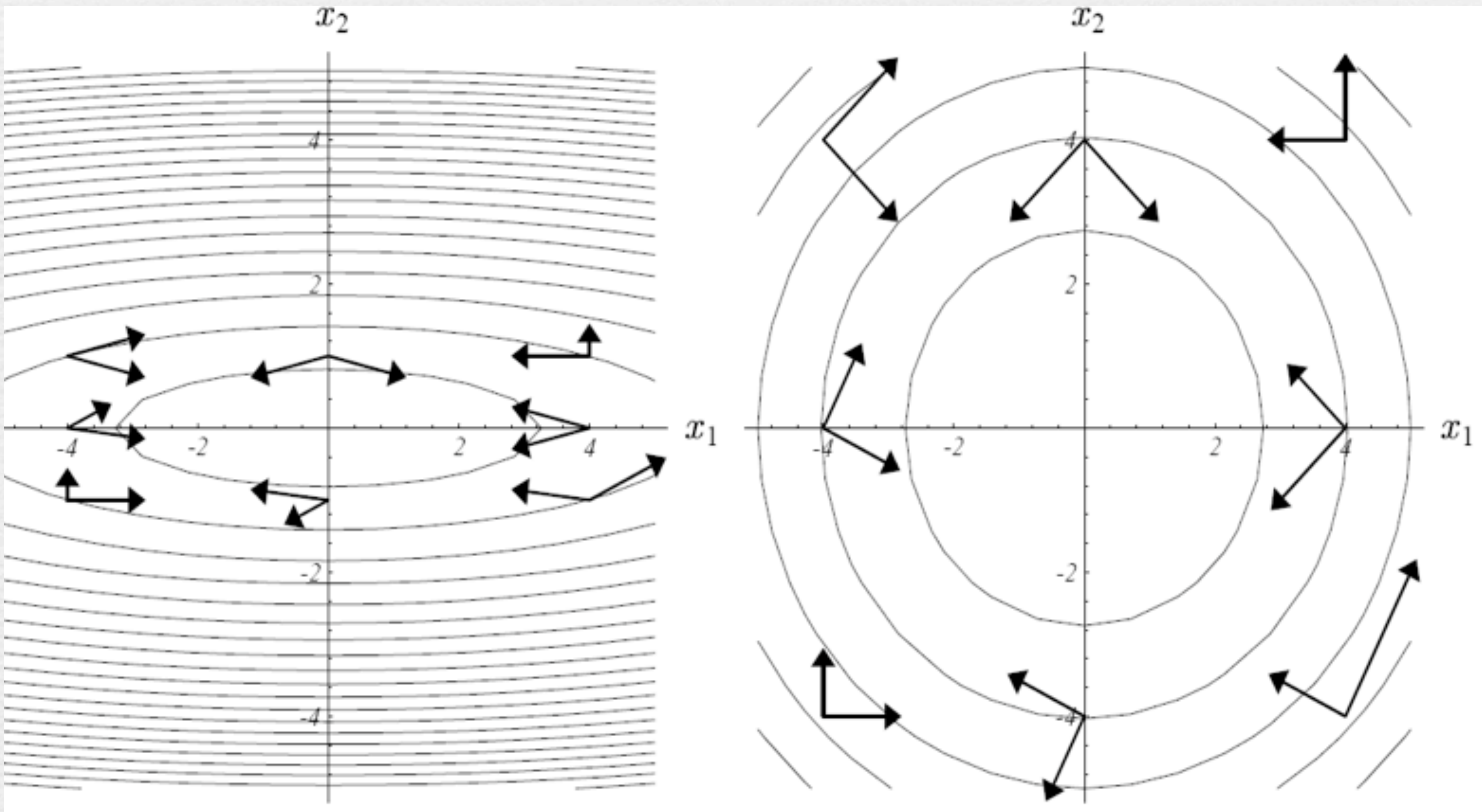
PRECONDIT IONING

Idea

- We want to solve Ax=b
- Fo any invertible matrix M, this is the same as solving MAx=Mb
- Maybe some M make the problem easier
- Preconditioning seeks a matrix M that accelerates convergence
- In practice, M does not need to be applied to A, only to direction vectors d

Preconditioning

- At a high level, try to turn the ellipses into circles
  - Then even gradient descent could work well.



(a) (b)

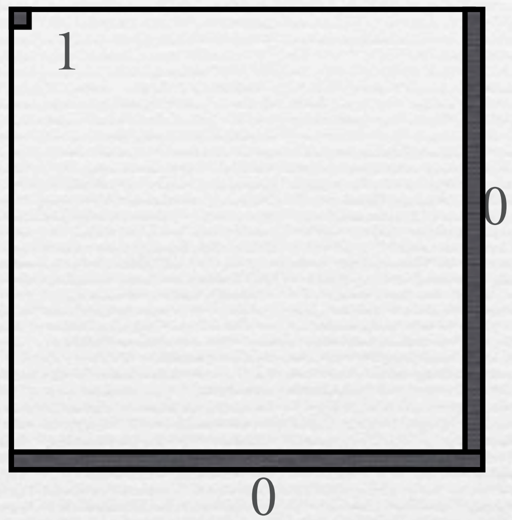
Preconditioning

- Perfect preconditioning involves the inverse matrix
  - Probably too costly an acceleration!
- Simplest preconditioning: divide by diagonal elements (good if matrix has strong diagonal)
- Run a solver (e.g. Cholesky decomposition) but only partially
- Or use smart basis functions such as wavelets or pyramids
  - http://portal.acm.org/citation.cfm?id=1142005

MULTIGRID

Motivation

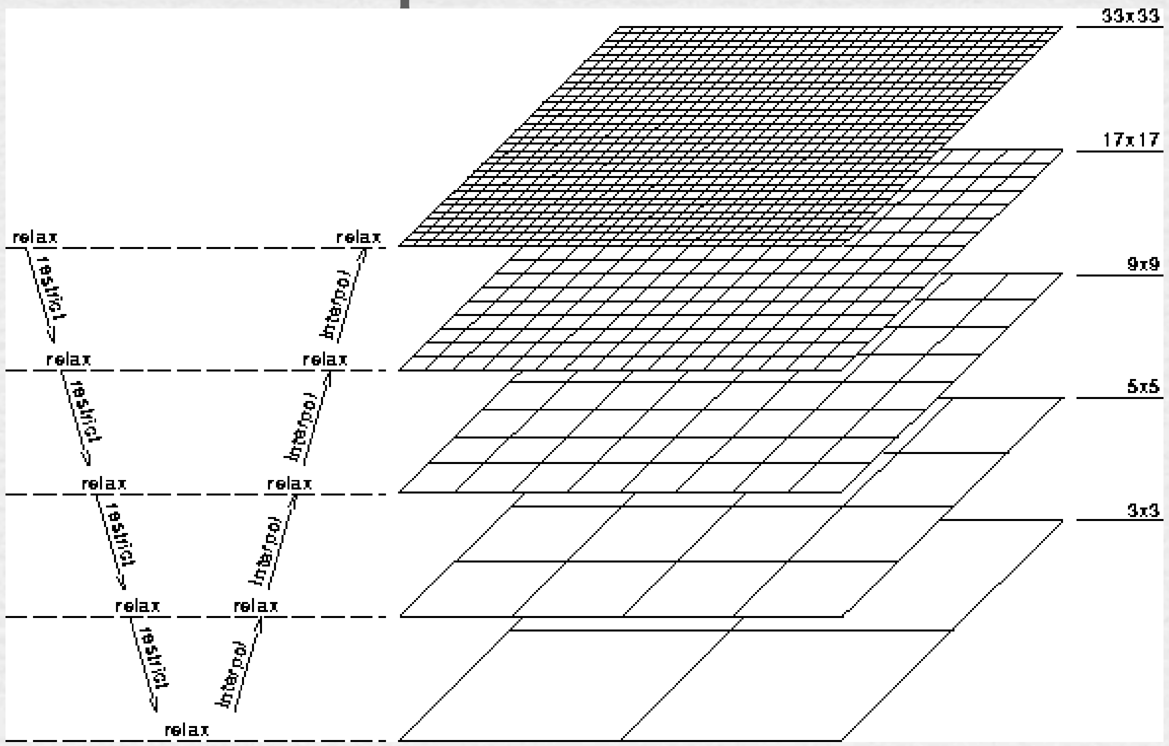
- Laplace equation (minimize square gradient)
- Boundary conditions:
  1 at one corner,
  zero on opposite 2 borders



- Initialize with e.g.
  zero everywhere
- 1st iteration only updates pixels connected to corner
- 2nd iteration only updates their neighbors
- Takes width to reach border: slow

Multigrid

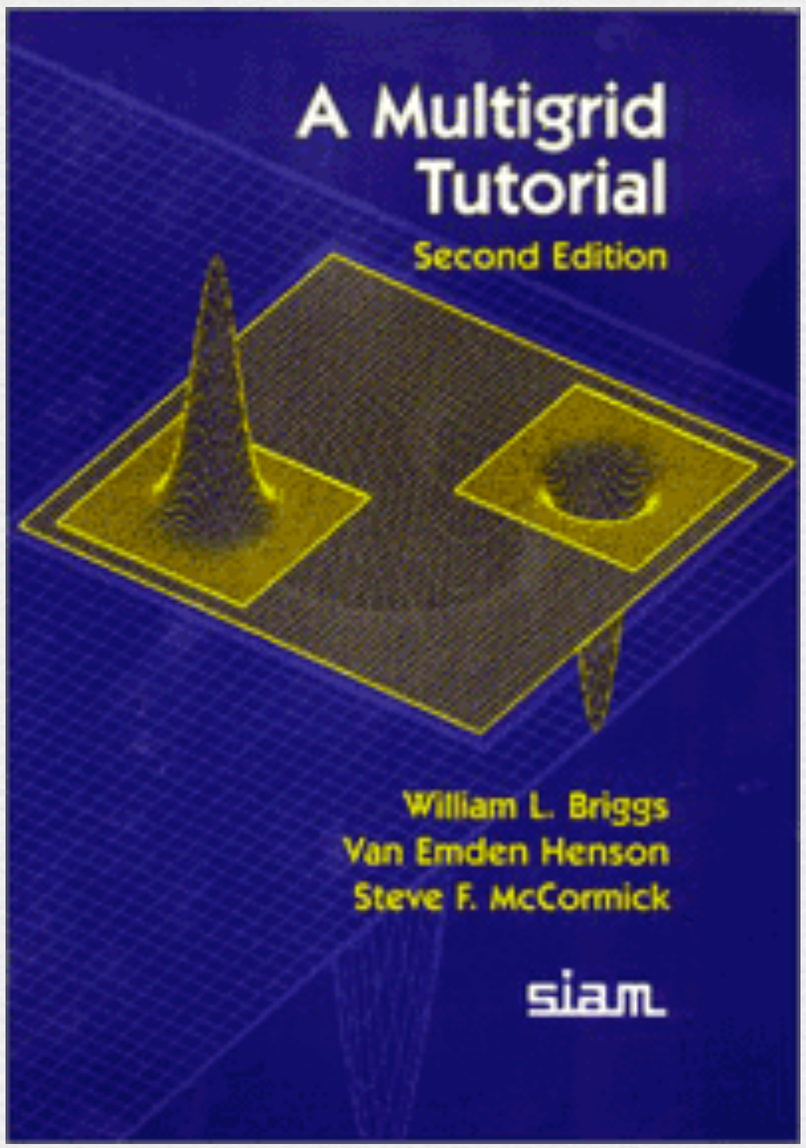
- Solve the problem at multiple resolutions
- In particular, also solve a lower-resolution version where propagation is faster
- Initialize high-resolution version with upsampled-coarser resolution
- Also update coarser solution with finer solution



http://www.mgnet.org/mgnet/tutorials/xwb/mg.html

The reference

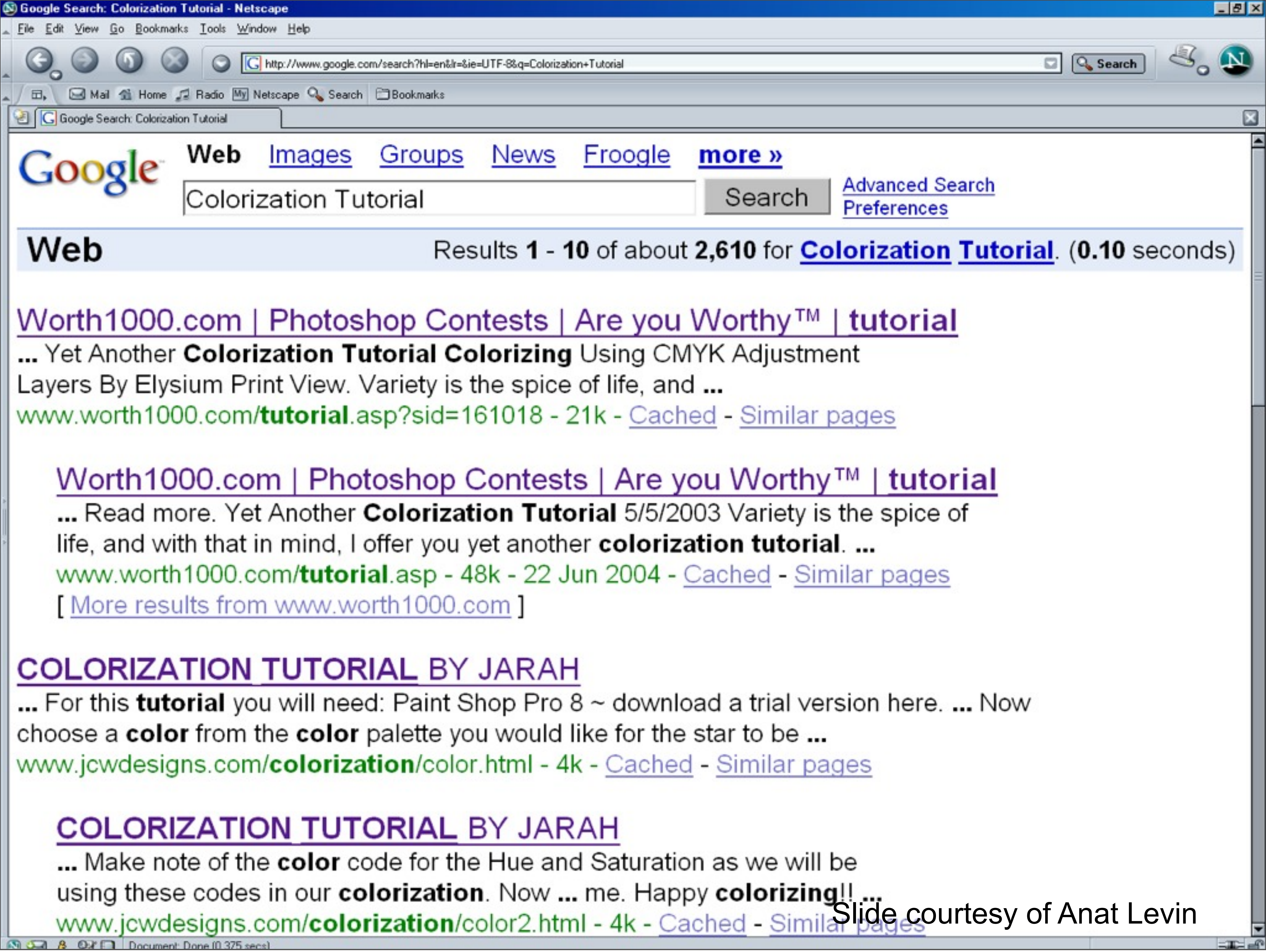
- https://computation.llnl.gov/casc/people/henson/mgtut/welcome.html



Refs

- http://www.cs.huji.ac.il/~yweiss/Colorization/
- http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf
- http://www.llnl.gov/casc/people/henson/mgtut/welcome.html
  - http://www.math.ust.hk/~mawang/teaching/math532/mgtut.pdf
  - http://books.google.com/books?id=SRAZwqAkrQMC&dq=multigrid+tutorial&printsec=frontcover&source=bn&hl=en&ei=KS6tSZjRO4vltgf-mc2LBg&sa=X&oi=book_result&resnum=4&ct=result
- http://www-users.cs.umn.edu/~saad/books.html
- http://en.wikipedia.org/wiki/Conjugate_gradient_method

EXTRA MATERIAL



Google Search: Colorization Tutorial - Netscape

http://www.google.com/search?hl=en&lr=&ie=UTF-8&q=Colorization+Tutorial

Web Images Groups News Froogle more »

Colorization Tutorial | Search | Advanced Search | Preferences

Web Results **1** - **10** of about **2,610** for **Colorization Tutorial**. (**0.10** seconds)

Worth1000.com | Photoshop Contests | Are you Worthy™ | **tutorial**

... Yet Another **Colorization Tutorial Colorizing** Using CMYK Adjustment Layers By Elysium Print View. Variety is the spice of life, and ...

www.worth1000.com/**tutorial**.asp?sid=161018 - 21k - Cached - Similar pages

Worth1000.com | Photoshop Contests | Are you Worthy™ | **tutorial**

... Read more. Yet Another **Colorization Tutorial** 5/5/2003 Variety is the spice of life, and with that in mind, I offer you yet another **colorization tutorial**. ...

www.worth1000.com/**tutorial**.asp - 48k - 22 Jun 2004 - Cached - Similar pages

[More results from www.worth1000.com]

COLORIZATION TUTORIAL BY JARAH

... For this **tutorial** you will need: Paint Shop Pro 8 ~ download a trial version here. ... Now choose a **color** from the **color** palette you would like for the star to be ...

www.jcwdesigns.com/**colorization**/color.html - 4k - Cached - Similar pages

COLORIZATION TUTORIAL BY JARAH

... Make note of the **color** code for the Hue and Saturation as we will be using these codes in our **colorization**. Now ... me. Happy **colorizing**!! ...

www.jcwdesigns.com/**colorization**/color2.html - 4k - Cached - Similar pages

Slide courtesy of Anat Levin

Typical Colorization Process

Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**

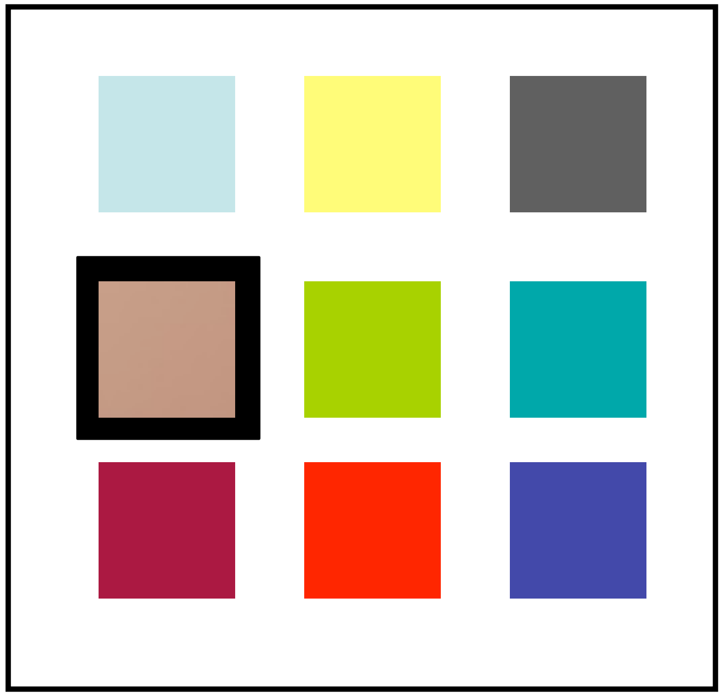
Images from: “Yet Another Colorization Tutorial”

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**
- **Choose region color from palette.**



Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- Delineate region boundary
- Choose region color from palette.

Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Typical Colorization Process

- **Delineate region boundary**
- **Choose region color from palette.**

Images from: "Yet Another Colorization Tutorial"

http://www.worth1000.com/tutorial.asp?sid=161018

Slide courtesy of Anat Levin

Video Colorization Process

- Delineate region boundary
- Choose region color from palette.
- Track regions across video frames

Slide courtesy of Anat Levin

Colorization Process Discussion

Time consuming and labor intensive

- **Fine boundaries.**
- **Failures in tracking.**

Slide courtesy of Anat Levin