

# Fitting Surface Models to Data

CVPR 2016 Tutorial

Andrew Fitzgibbon, Microsoft  
Jonathan Taylor, PerceptivelO

0900	Intro: Applications in vision and graphics.
	Lots of exciting and inspirational examples of model fitting: <ul style="list-style-type: none"> <li>• Kinetre (Siggraph 12)</li> <li>• Dolphins (PAMI 13)</li> <li>• Nonrigid tracking (Siggraph 14)</li> <li>• FlexSense (CHI 15)</li> <li>• Hand tracking (Siggraph 16)</li> </ul>
0920	Session I: Matrix and vector calculus, nonlinear optimization <ul style="list-style-type: none"> <li>• vector functions and the Jacobian, generalized Jacobian</li> <li>• advanced matrix operations: block operations, kronecker products etc</li> <li>• derivatives of matrix expressions</li> <li>• sparse matrices and sparse storage</li> <li>• finite-difference versus symbolic derivatives</li> <li>• nonlinear optimization, Gauss-Newton and Levenberg-Marquardt algorithms</li> <li>• gradient descent vs Newton</li> <li>• linear vs quadratic convergence</li> </ul>
1030	Coffee
1045	Session II: Curves and Correspondences <ul style="list-style-type: none"> <li>• What is a curve? Parametric descriptions of curves and surfaces</li> <li>• Curves and data points: closest point operations</li> <li>• Fitting curves to data: correspondences</li> <li>• Iterated closest points</li> <li>• “Lifting” correspondences</li> <li>• Worked example: Gauss's Ceres problem</li> </ul>
1140	Break and stretch
1145	Session III: Surfaces <ul style="list-style-type: none"> <li>• Splines and subdivision surfaces in 3D</li> <li>• Optimizing with subdivision</li> <li>• Implementing for speed</li> </ul>
1230	Lunch

[http://awf.fitzgibbon.ie/cvpr16\\_tutorial](http://awf.fitzgibbon.ie/cvpr16_tutorial)

1400	Session IV: Robustness and speed <ul style="list-style-type: none"> <li>• Models             <ul style="list-style-type: none"> <li>• ARAP</li> <li>• Background - DT ok for tracking, not for personalization</li> <li>• Priors on correspondences, e.g. piecewise continuous contour generator</li> <li>• Exposing Structure in Sum of Squares Form</li> <li>• Error metric                 <ul style="list-style-type: none"> <li>• Robust terms</li> <li>• square root trick</li> <li>• A great example of where “lifting” really helps</li> </ul> </li> </ul> </li> </ul>
1500	Coffee/Stretch
1515	Session V: Software <ul style="list-style-type: none"> <li>• OpenSubdiv</li> <li>• Eigen</li> <li>• Ceres</li> <li>• Opt (Guest lecture from Matthias Niessner)</li> <li>• AD tools: Theano etc</li> </ul>
1615	More coffee, more stretching
1630	Session VI: Conclusions, open problems, misc... <ul style="list-style-type: none"> <li>• Topology adaptation</li> <li>• Where are the local minima?</li> <li>• And where lifting really hurts: VarPro algorithms</li> <li>• Implementing rotations: quaternions vs infinitesimals with recentering</li> <li>• derivatives of minimization problems</li> <li>• Schur complement QR</li> </ul>
1715	Close

**Finding Nemo: Deformable Object Class Modelling using Curve Matching** CVPR '10

Mukta Prasad, Andrew Fitzgibbon, Andrew Zisserman, Luc Van Gool

**KinÊtre: Animating the World with the Human Body** UIST '12

Jiawen (Kevin) Chen, Shahram Izadi, Fitzgibbon

**The Vitruvian Manifold: Inferring dense correspondences for one-shot human pose estimation** CVPR '12

Jonathan Taylor, Jamie Shotton, Toby Sharp, Fitzgibbon

**What shape are dolphins? Building 3D morphable models from 2D images** PAMI '13

Tom Cashman, Fitzgibbon

**User-Specific Hand Modeling from Monocular Depth Sequences** CVPR '14

Taylor, Richard Stebbing, Varun Ramakrishna, Cem Keskin, Shotton, Izadi, Fitzgibbon, Aaron Hertzmann

**Real-Time Non-Rigid Reconstruction Using an RGB-D Camera** SIGGRAPH '14

Michael Zollhöfer, Matthias Nießner, Izadi, Christoph Rhemann, Christopher Zach,  
Matthew Fisher, Chenglei Wu, Fitzgibbon, Charles Loop, Christian Theobalt, Marc Stamminger

**Learning an Efficient Model of Hand Shape Variation from Depth Images** CVPR '15

Sameh Khamis, Taylor, Shotton, Keskin, Izadi, Fitzgibbon

**Efficient and Precise Interactive Hand Tracking through Joint, Continuous Optimization of Pose and Correspondences** SIGGRAPH '16

Taylor, Lucas Bordeaux, Cashman, Bob Corish, Keskin, Sharp, Eduardo Soto, David Sweeney, Julien Valentin,  
Ben Luff, Arran Topalian, Erroll Wood, Khamis, Kohli, Izadi, Richard Banks, Fitzgibbon, Shotton.

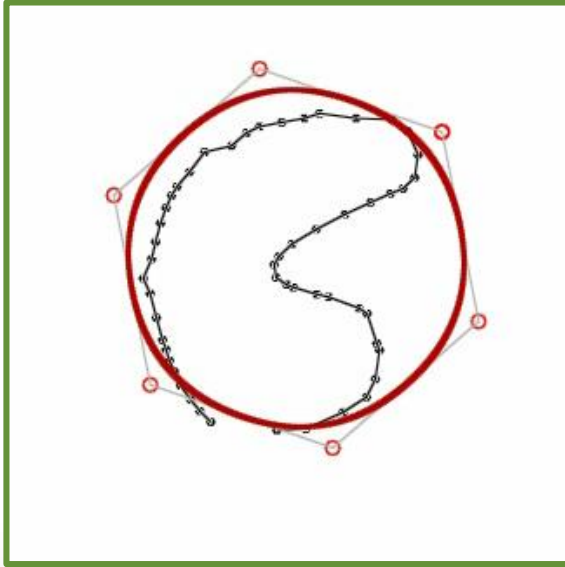
**Fits Like a Glove: Rapid and Reliable Hand Shape Personalization.** CVPR '16

David Joseph Tan, Cashman, Taylor, Fitzgibbon, Daniel Tarlow, Khamis, Izadi, Shotton.

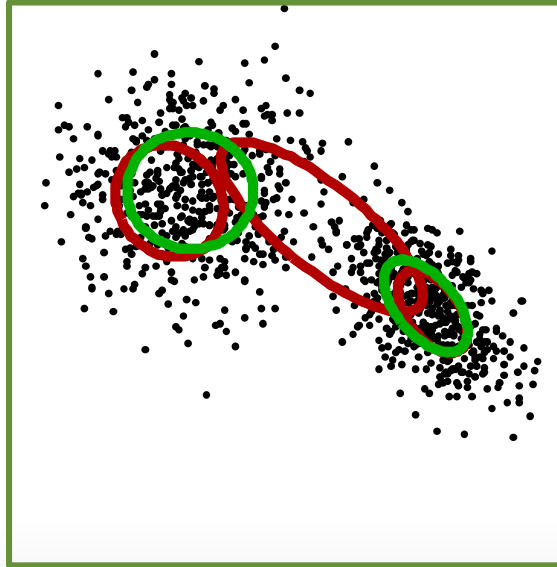
# Goal

LEARN HOW TO SOLVE HARD VISION PROBLEMS,  
USING TOOLS THAT MAY APPEAR INELEGANT,  
BUT ARE MUCH SMARTER THAN THEY LOOK.





Curve/surface fitting



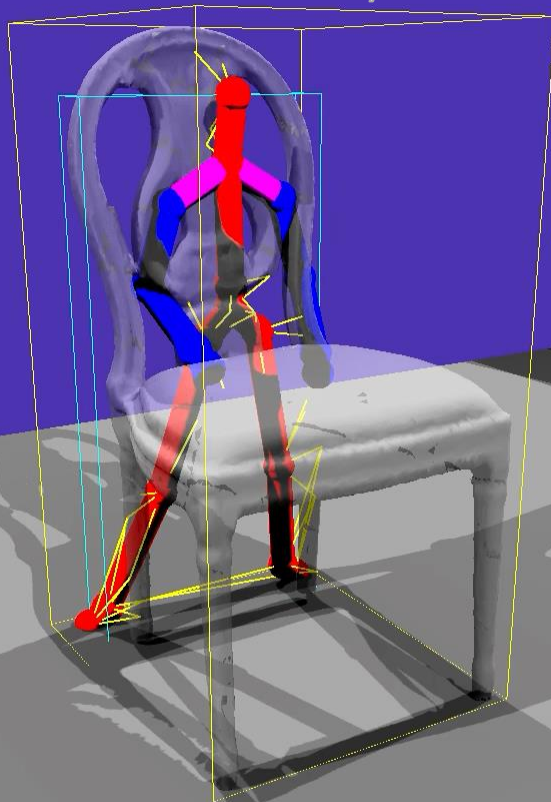
Parameter estimation



“Bundle adjustment”  
(Video from our friends at Google)

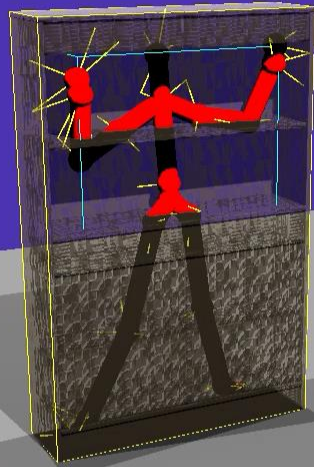
# "KinEtre: Animating the World with the Human Body ^"

Chen et al. UIST 2012



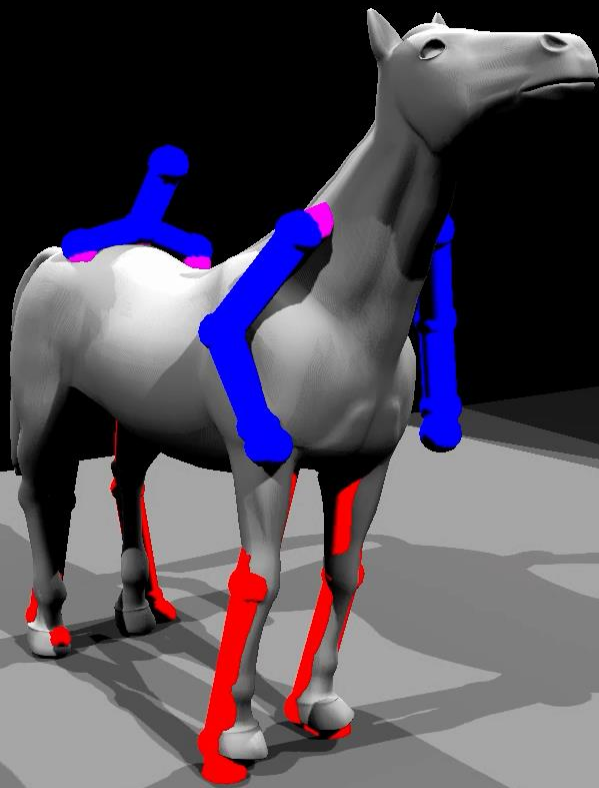
# "KinEtre: Animating the World with the Human Body ^"

Chen et al. UIST 2012



# "KinEtre: Animating the World with the Human Body ^"

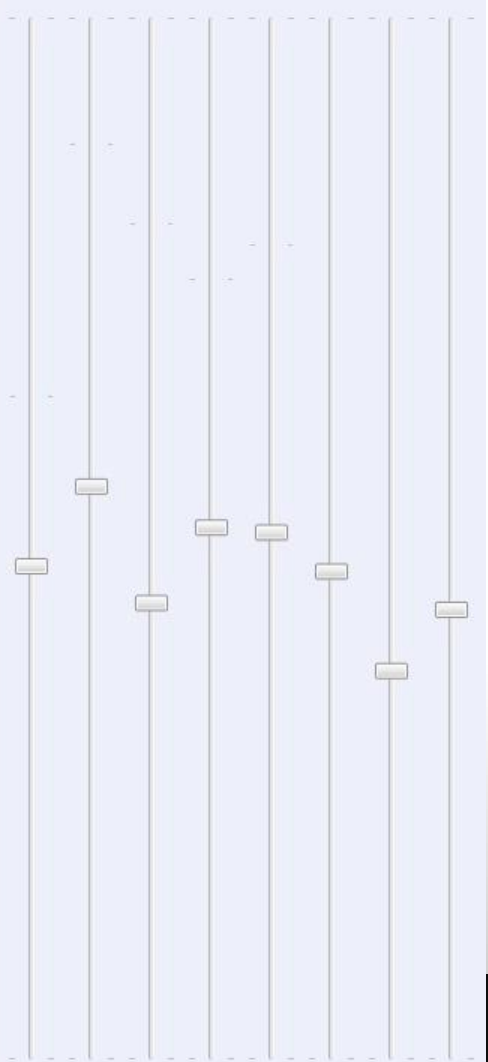
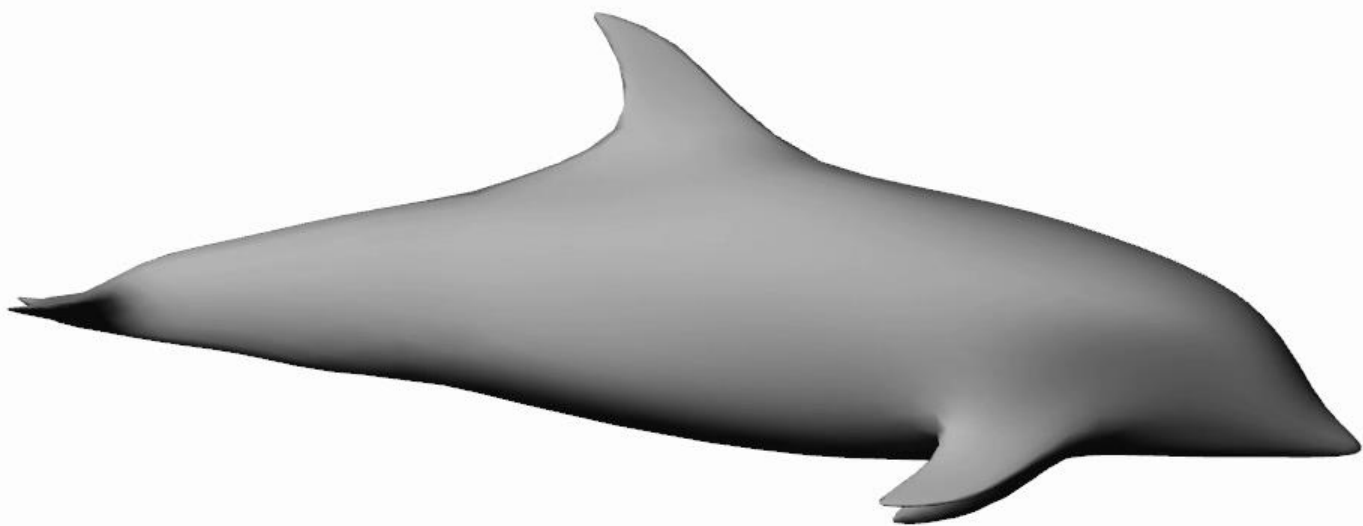
Chen et al. UIST 2012







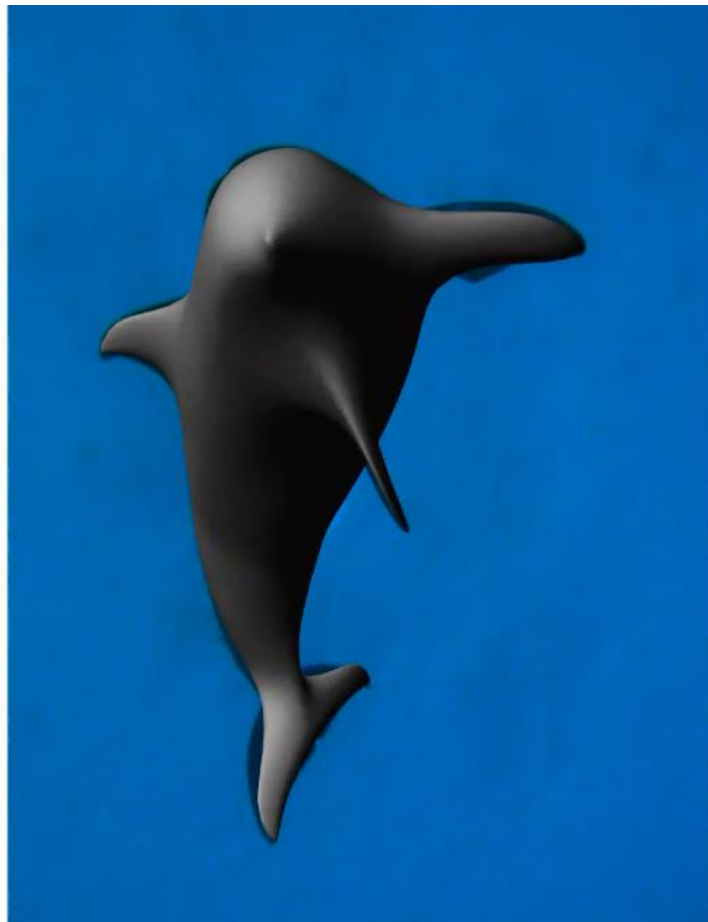




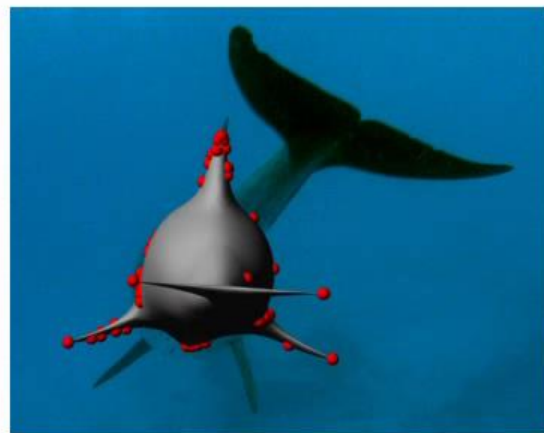
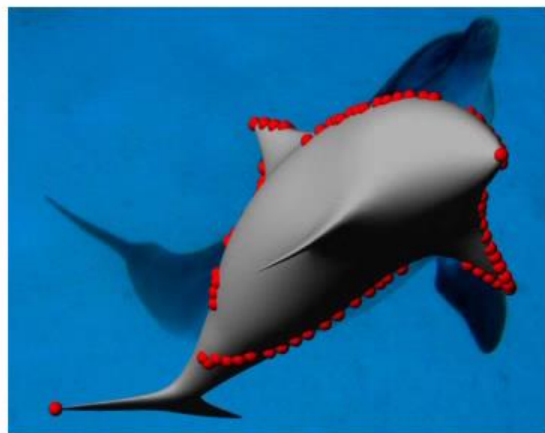
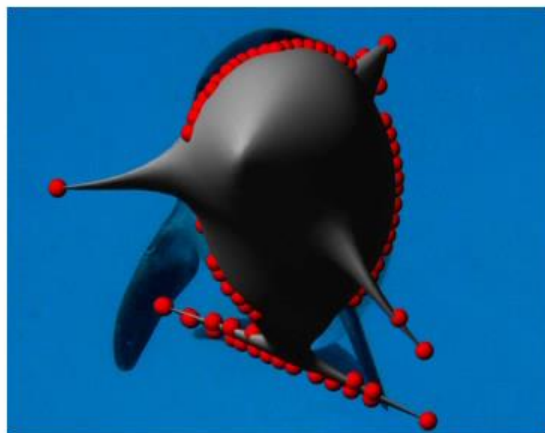
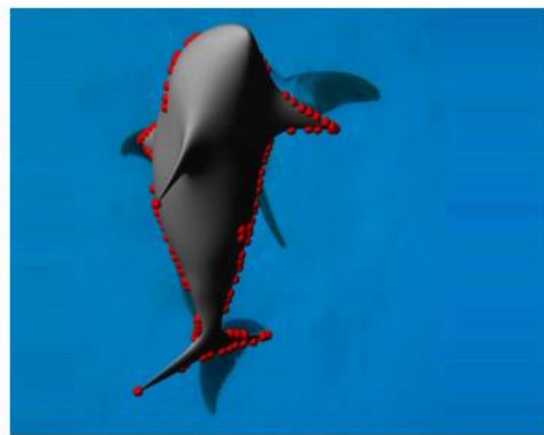
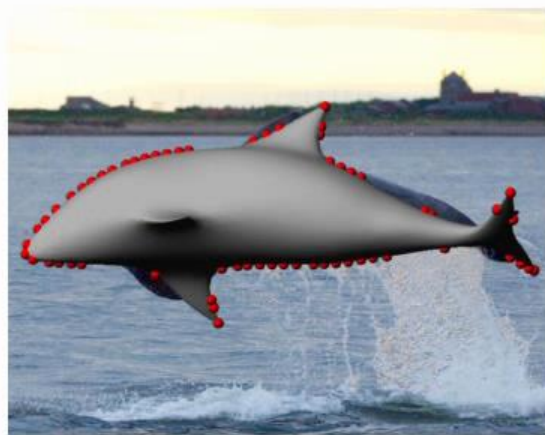


FITTING SUBDIVISION SURFACES TO 2D DATA





FITTING SUBDIVISION SURFACES TO  $2D$  DATA





FITTING POLYGON MESHES TO VIDEO



Input Kinect Stream

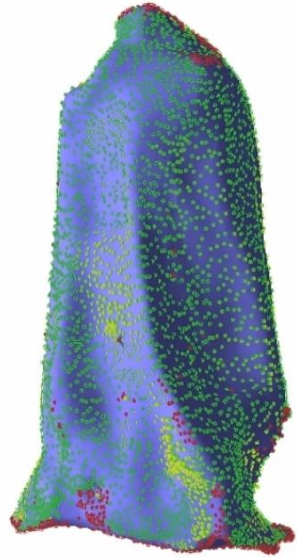


KinectFusion

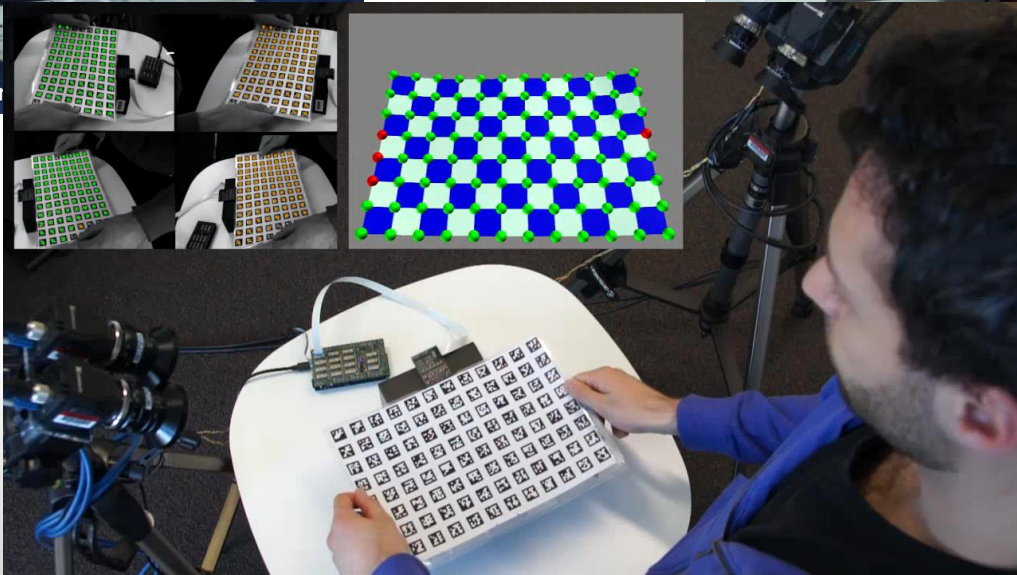
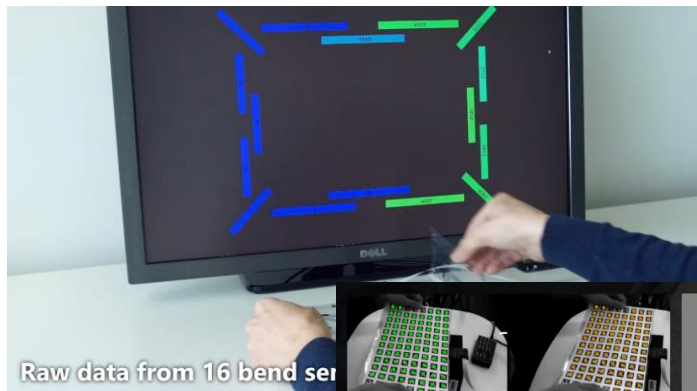


Deformable Fusion

[3D Scanning Deformable Objects with a Single RGBD Sensor, Dou et al, CVPR15]

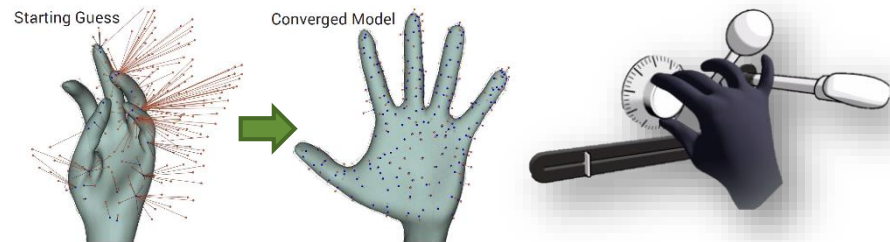
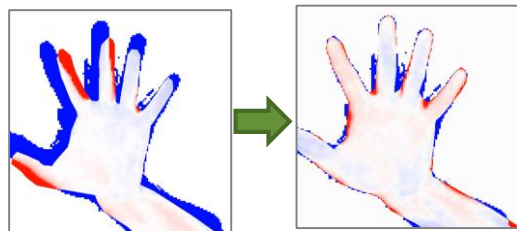


[Zollhöfer & al, SIGGRAPH '14]

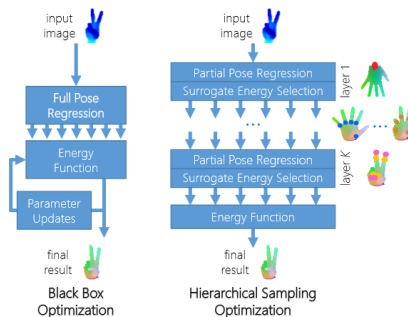
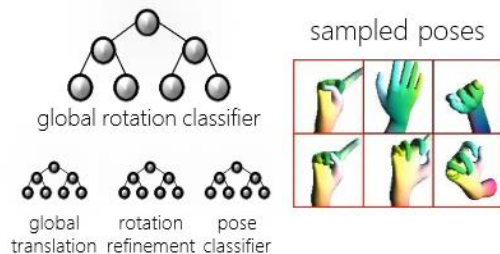




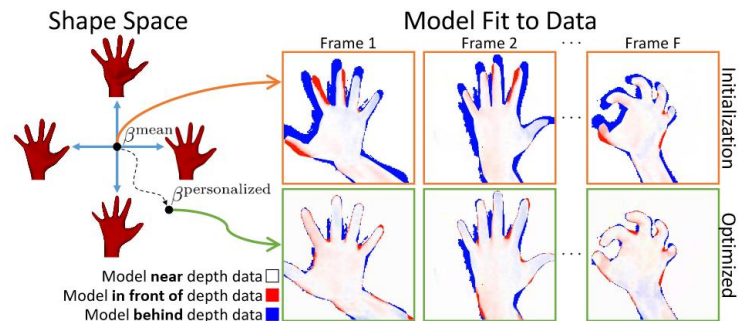
- Hand Pose Estimation via **Model Fitting** (read “Hand Tracking”)
  - CHI 2015, SIGGRAPH 2016



- Discriminative Hand Pose **Reinitialization**
  - ICCV 2015, CHI 2015



- Hand Shape **Personalization**:
  - CVPR 2014, CVPR 2015, CVPR 2016

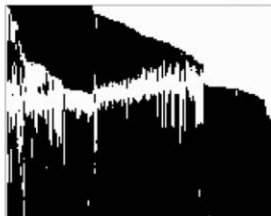




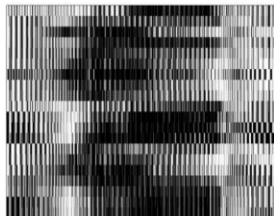




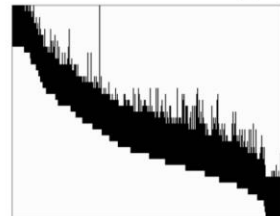
$240 \times 167$   
30% missing



$20 \times 2944$   
42% missing



$72 \times 319$   
72% missing



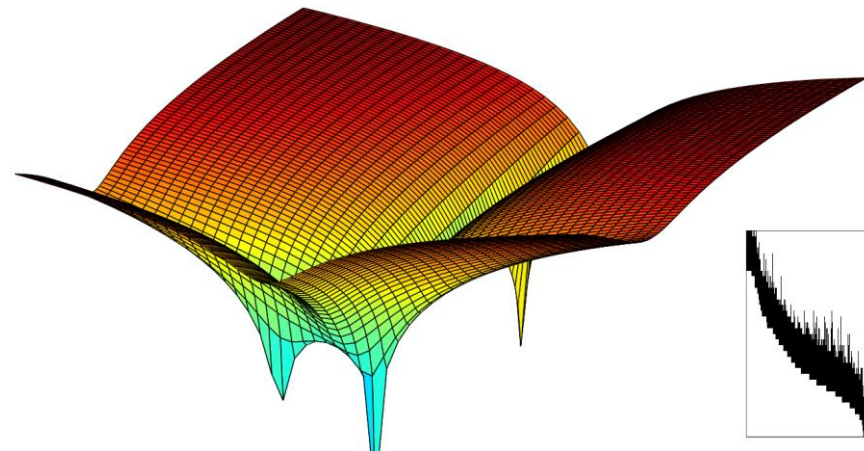
**Task:** Given noisy observation  $\mathbf{M}$ , and weight matrix  $\mathbf{W}$ , compute

$$\operatorname{argmin}_{\mathbf{A}, \mathbf{B}} \|\mathbf{W} \odot (\mathbf{M} - \mathbf{AB}^\top)\|_F^2$$

where  $\mathbf{R} = \mathbf{P} \odot \mathbf{Q} \Leftrightarrow r_{ij} = p_{ij}q_{ij}$ .

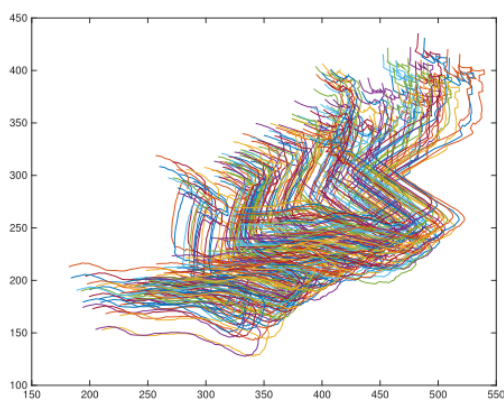
2D slice through  $\log \epsilon(\mathbf{A}, \mathbf{B})$  where

$$\epsilon(\mathbf{A}, \mathbf{B}) := \|\mathbf{W} \odot (\mathbf{M} - \mathbf{AB}^\top)\|_F^2$$

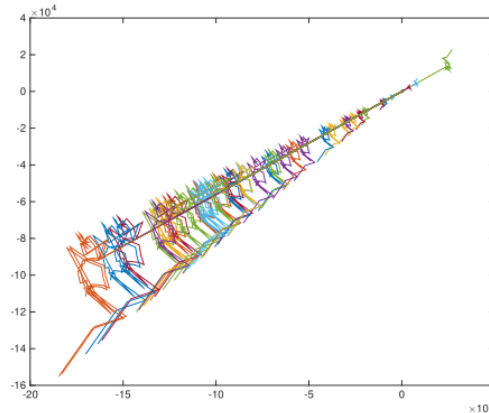


Algorithm	Framework	Manifold retraction
ALS [5]	RW3 (ALS)	None
PowerFactorization [5, 27]	RW3 (ALS)	$q$ -factor
LM-S [8]	Newton + $\langle \text{Damping} \rangle$	orth (replaced by $q$ -factor)
LM-S <sub>GN</sub> [9, 13]	RW1 (GN) + $\langle \text{Damping} \rangle$ (DRW1 equiv.)	orth (replaced by $q$ -factor)
LM-M [8]	Reduced <sub>r</sub> Newton + $\langle \text{Damping} \rangle$	orth (replaced by $q$ -factor)
LM-M <sub>GN</sub> [8]	Reduced <sub>r</sub> RW1 (GN) + $\langle \text{Damping} \rangle$	orth (replaced by $q$ -factor)
Wiberg [18]	RW2 (Approx. GN)	None
Damped Wiberg [19]	RW2 (Approx. GN) + $\langle \text{Projection const.} \rangle_P + \langle \text{Damping} \rangle$	None
CSF [13]	RW2 (Approx. GN) + $\langle \text{Damping} \rangle$ (DRW2 equiv.)	$q$ -factor
RTRMC [4]	Projected <sub>p</sub> Newton + $\langle \text{Regularization} \rangle + \langle \text{Trust Region} \rangle$	$q$ -factor
LM-S <sub>RW2</sub>	RW2 (Approx. GN) + $\langle \text{Damping} \rangle$ (DRW2 equiv.)	$q$ -factor
LM-M <sub>RW2</sub>	Reduced <sub>r</sub> RW2 (Approx. GN) + $\langle \text{Damping} \rangle$	$q$ -factor
DRW1	RW1 (GN) + $\langle \text{Damping} \rangle$	$q$ -factor
DRW1P	RW1 (GN) + $\langle \text{Projection const.} \rangle_P + \langle \text{Damping} \rangle$	$q$ -factor
DRW2	RW2 (Approx. GN) + $\langle \text{Damping} \rangle$	$q$ -factor
DRW2P	RW2 (Approx. GN) + $\langle \text{Projection const.} \rangle_P + \langle \text{Damping} \rangle$	$q$ -factor

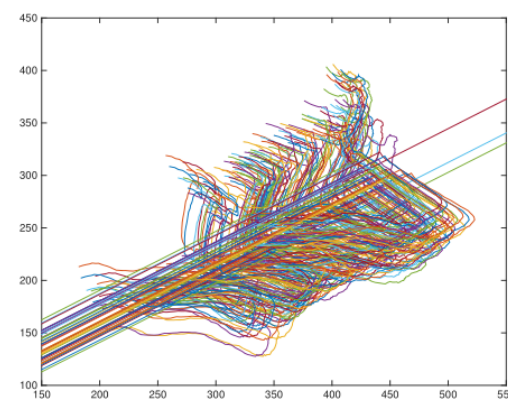
$$\begin{aligned}
\frac{1}{2}\mathbf{H}^* = & \mathbf{P}_r^\top \left( \tilde{\mathbf{V}}^{*\top} (\mathbf{I}_p - [\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\dagger]_{RW2}) \tilde{\mathbf{V}}^* + [\mathbf{K}_{mr}^\top \mathbf{Z}^* (\tilde{\mathbf{U}}^\top \tilde{\mathbf{U}})^{-1} \mathbf{Z}^{*\top} \mathbf{K}_{mr}]_{RW1} \times [-1]_{FN} \right. \\
& \left. + [\mathbf{K}_{mr}^\top \mathbf{Z}^* \tilde{\mathbf{U}}^\dagger \tilde{\mathbf{V}}^* \mathbf{P}_p + \mathbf{P}_p \tilde{\mathbf{V}}^{*\top} \tilde{\mathbf{U}}^\dagger \tilde{\mathbf{U}}^\top \mathbf{Z}^{*\top} \mathbf{K}_{mr}]_{FN} + \langle \alpha \mathbf{I}_r \otimes \mathbf{U}\mathbf{U}^\top \rangle_P + \langle \lambda \mathbf{I}_{mr} \rangle \right) \mathbf{P}_r
\end{aligned}$$



(a) Best known minimum (0.3228)



(b) Second best solution (0.3230)



(c) Second best, zoomed to image

Figure 1: Illustration that a solution with function value just .06% above the optimum can have significantly worse extrapolation properties. This is a reconstruction of point trajectories in the standard “Giraffe” sequence. Even when zooming in to eliminate gross outliers (not possible for many reconstruction problems), it is clear that numerous tracks have been incorrectly reconstructed.

# Towards Pointless Structure from Motion: 3D reconstruction from 3D curves

Irina Nurutdinova, Andrew Fitzgibbon, ICCV '15





# Towards Pointless Structure from Motion: 3D reconstruction from 3D curves

Irina Nurutdinova, Andrew Fitzgibbon, ICCV '15



[Zoom]

Dense reconstruction (PMVS) using  
cameras estimated from points only



[Zoom]

Dense reconstruction (PMVS) using  
cameras estimated from points and curves

Write energy describing the image collection

$$\sum_{f=1}^F E_{\text{data}}(I_f, \boldsymbol{\theta}_f) + E_{\text{reg}}(\boldsymbol{\theta}_f, \boldsymbol{\theta}_{\text{core}})$$

Where:

$\boldsymbol{\theta}_f$  are (unknown) parameters of surface model in frame  $f$

$\boldsymbol{\theta}_{\text{core}}$  are (unknown) parameters of some shape model (e.g. linear combination) and  $E_{\text{reg}}$  measures distance, e.g. ARAP

And optimize it using Levenberg-Marquardt

- (i.e. any Newton-like algorithm, making maximum use of problem structure)

- So, you can do lots of things by “fitting models to data”.
- How do you do it right?
- Let’s look at some examples.

# CONTINUOUS OPTIMIZATION

Andrew Fitzgibbon

Microsoft Research Cambridge





Given function

$$f(x): \mathbb{R}^d \mapsto \mathbb{R},$$

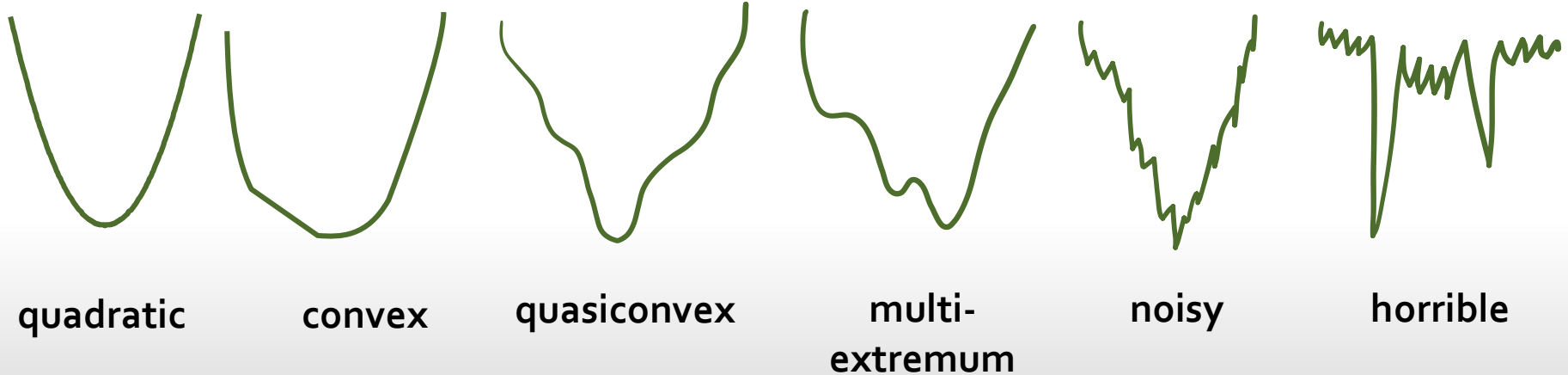
Devise strategies for finding  $x$  which minimizes  $f$

- Gradient descent++: Stochastic, Block, Minibatch
- Coordinate descent++: Block
- Newton++: Gauss, Quasi, Damped, Levenberg Marquardt, dogleg, Trust region, Doublestep LM, [L-]BFGS, Nonlin CG
- Not covered
  - Proximal methods: Nesterov, ADMM...

Given function

$$f(x): \mathbb{R}^d \mapsto \mathbb{R}$$

Devise strategies for finding  $x$  which minimizes  $f$

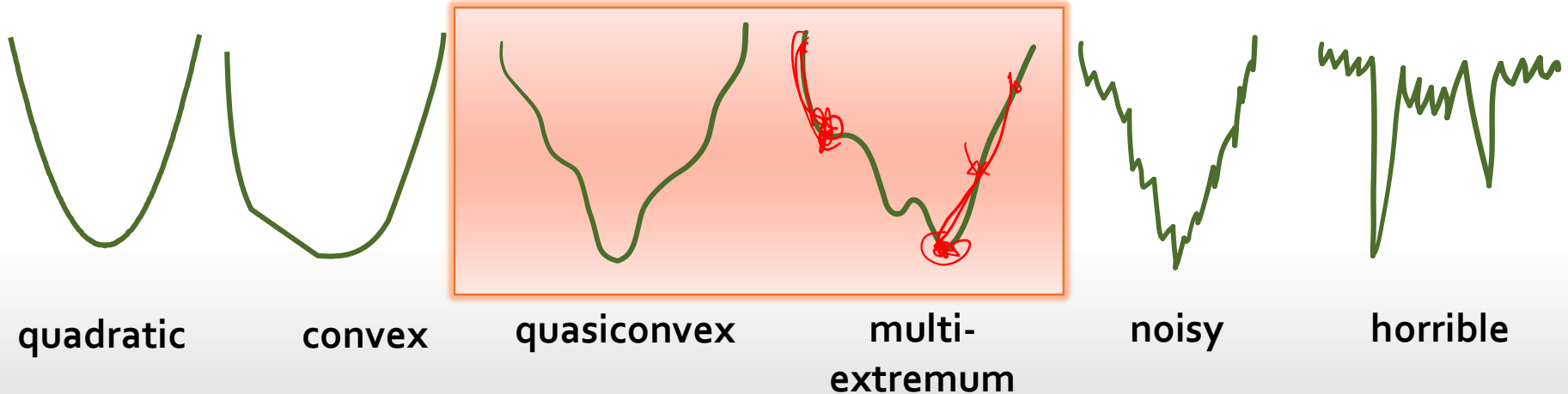


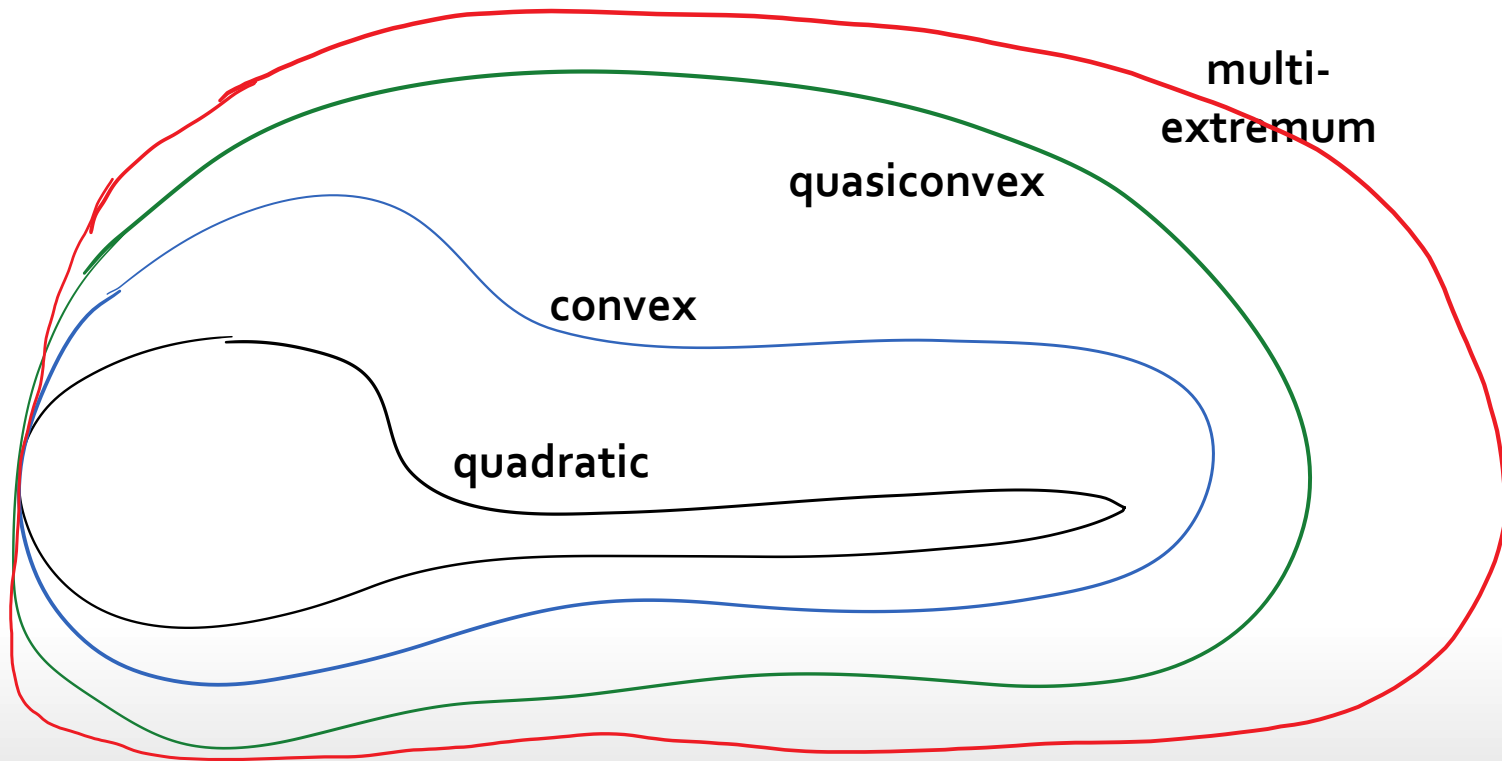
Given function

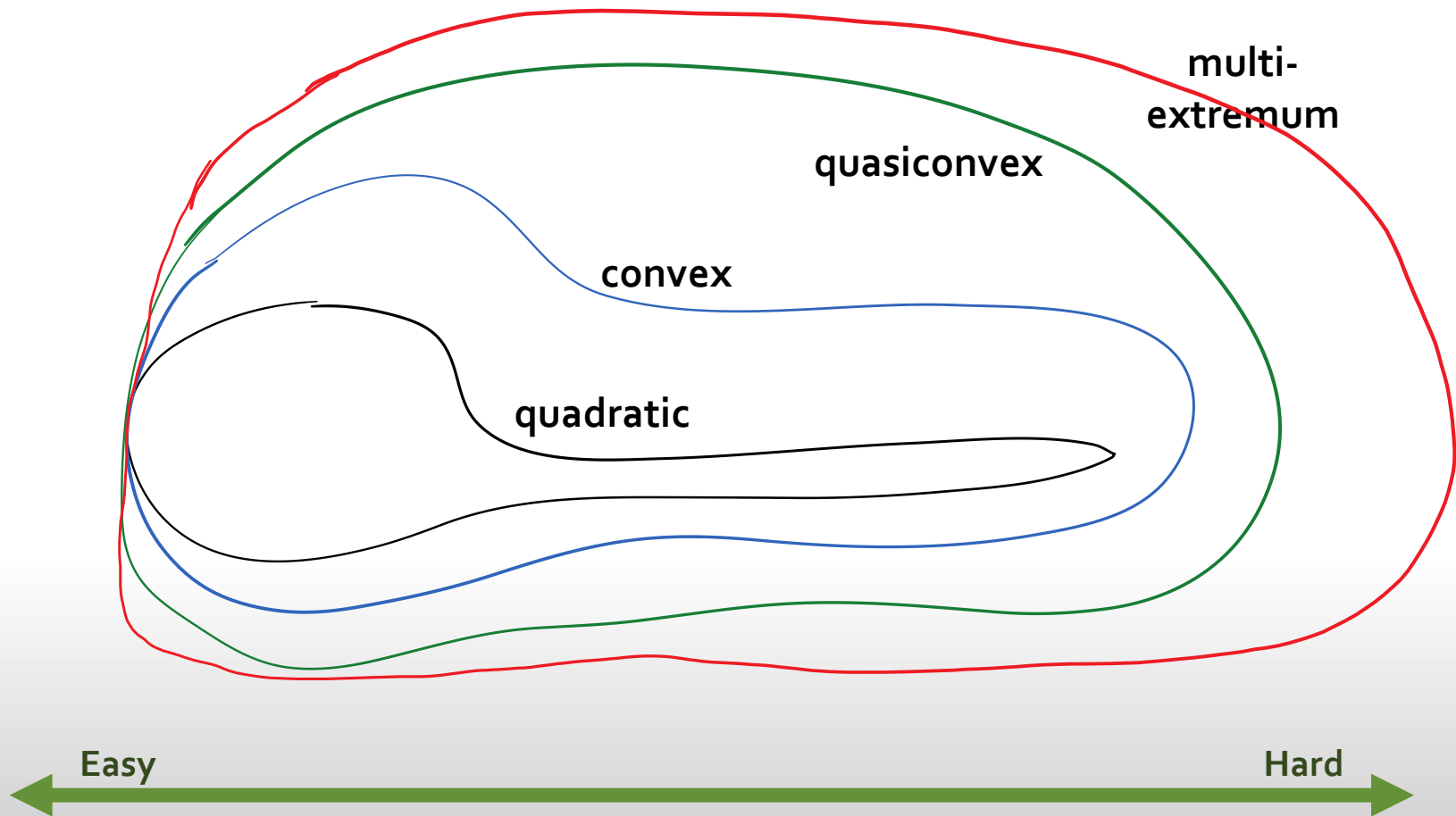
also

$$f(x): \mathbb{R}^d \mapsto \mathbb{R}$$

Devise strategies for finding  $x$  which minimizes  $f$







# Fast minimization depends on derivatives

- Gradient

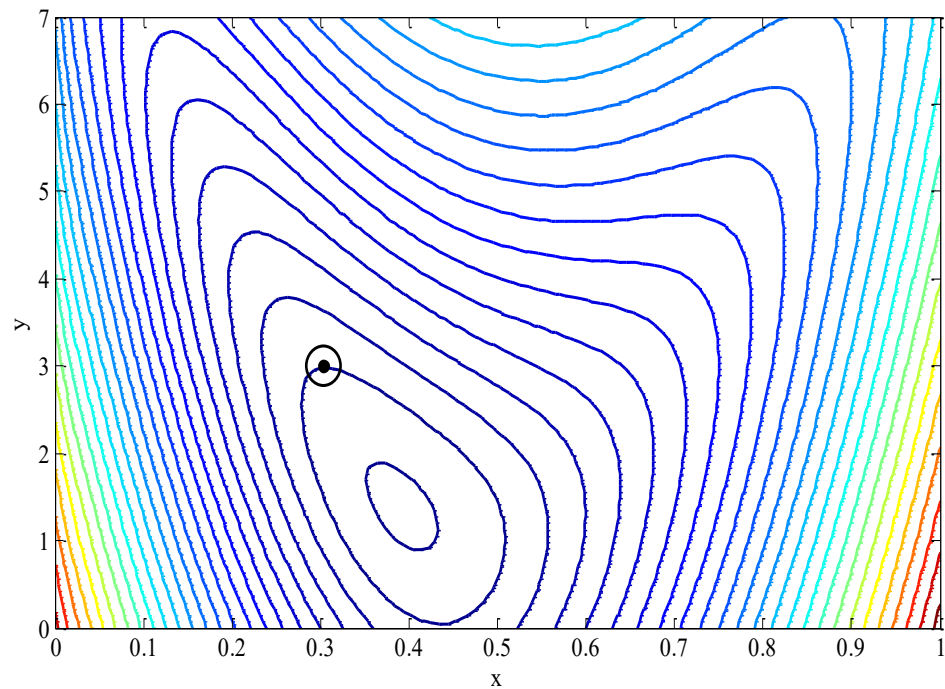
$$f: \mathbb{R}^n \mapsto \mathbb{R}$$

- When

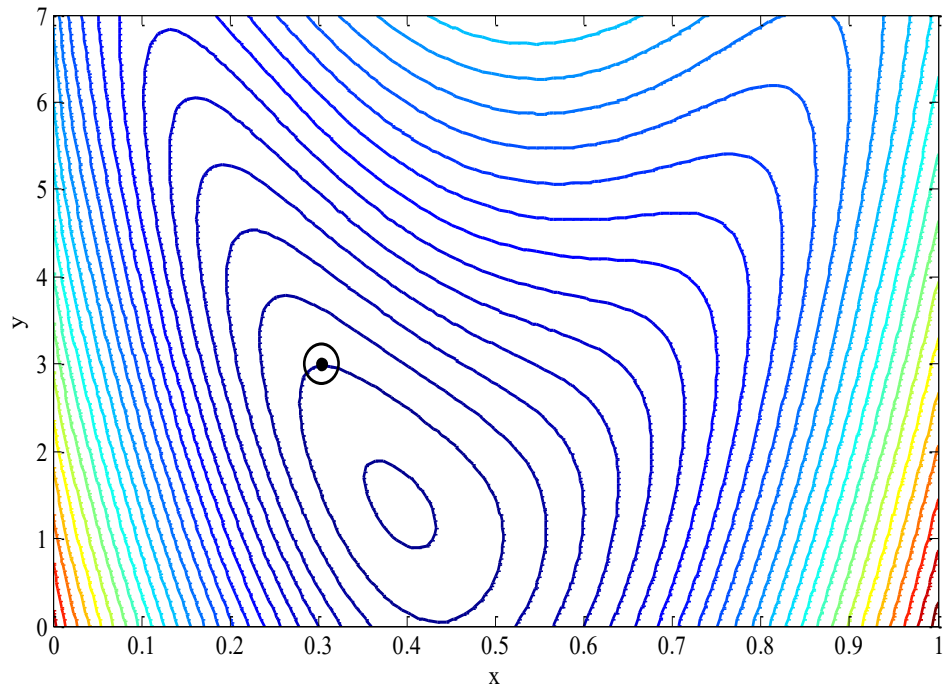
$$f(\mathbf{x}) = \|\mathbf{F}(\mathbf{x})\|^2 = \sum_i f_i(\mathbf{x})^2$$
$$\mathbf{F}: \mathbb{R}^n \mapsto \mathbb{R}^m$$

use *Jacobian*

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$$

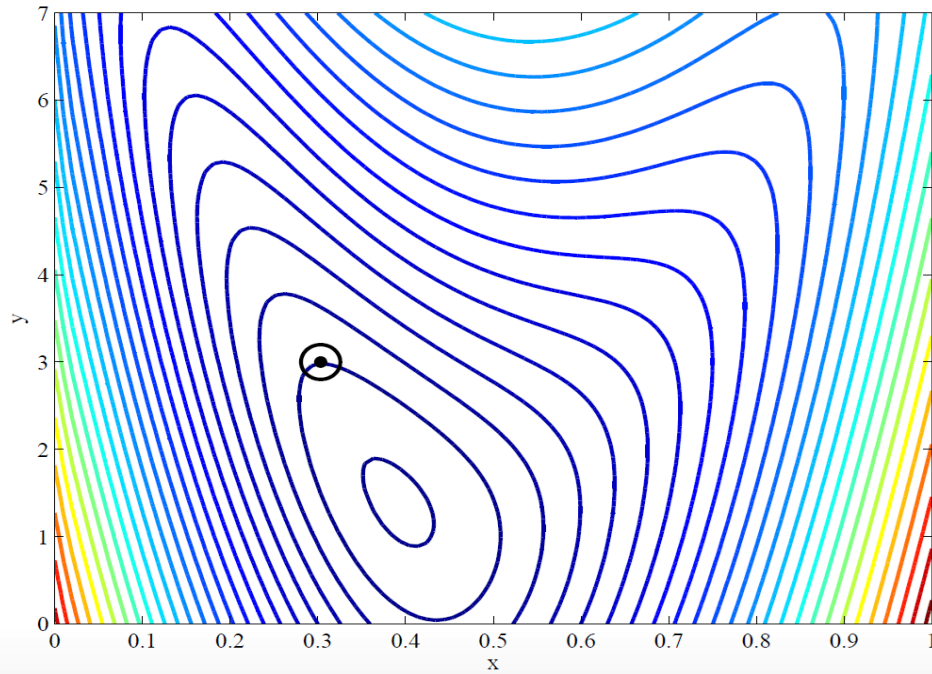


EXAMPLE



```
>> print -dmeta
```



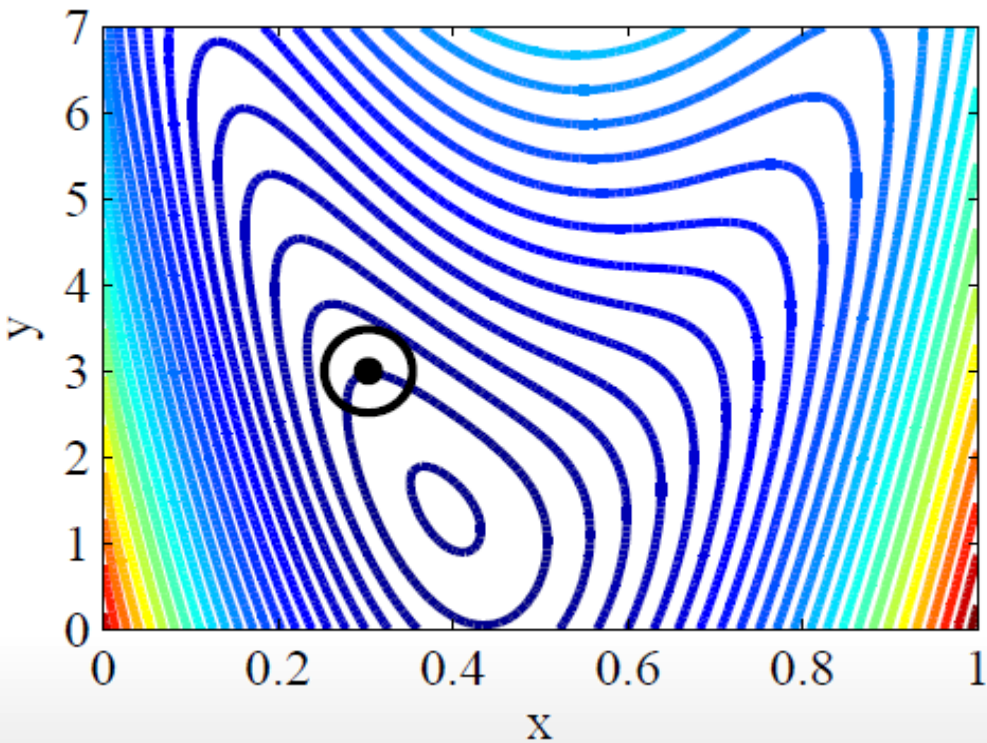


>> print -dpdf % then go to pdf and paste

OR

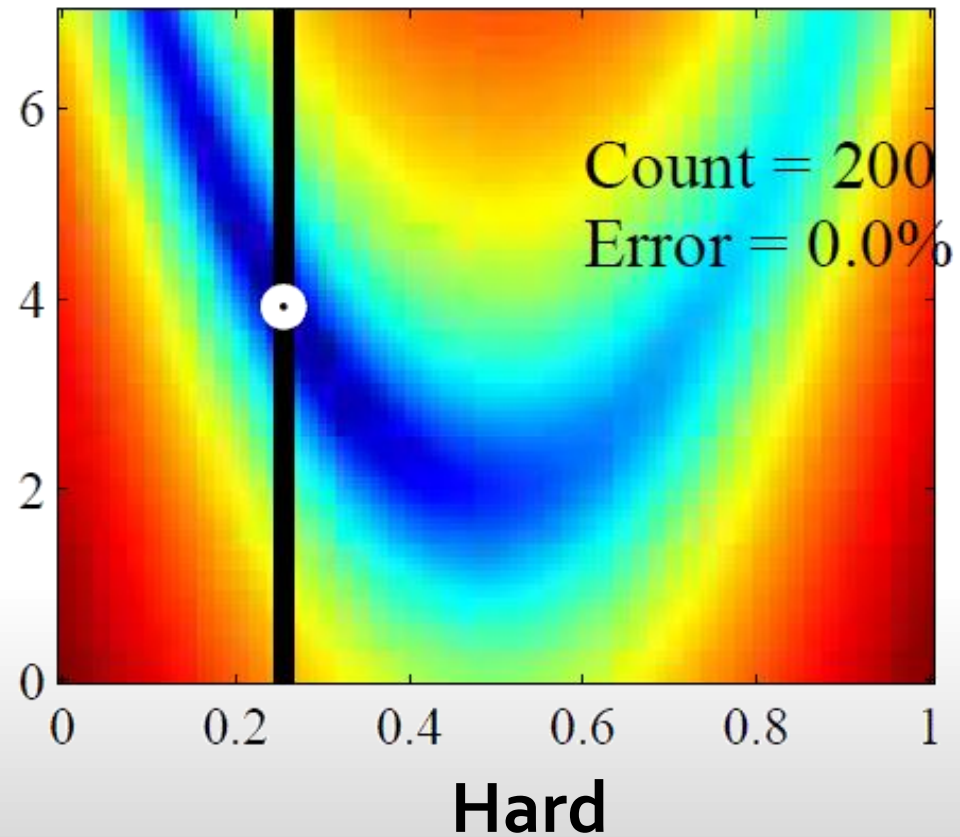
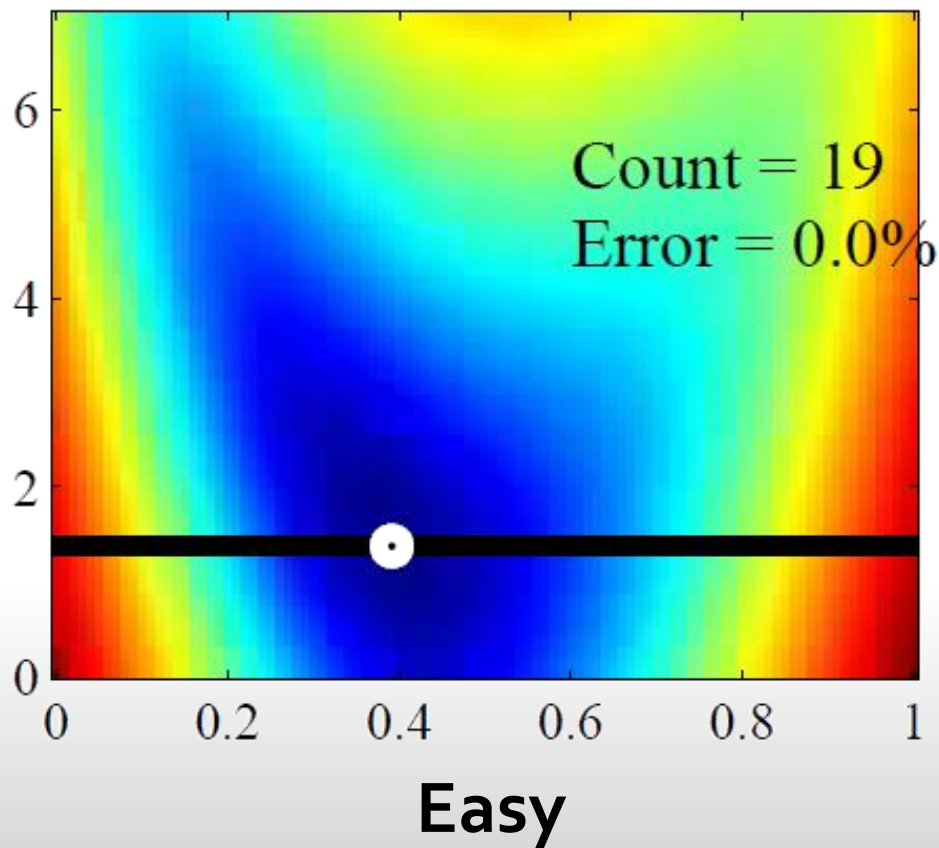
>> set(findobj(1, 'type', 'line'), 'linesmoothing', 'on') % then screengrab

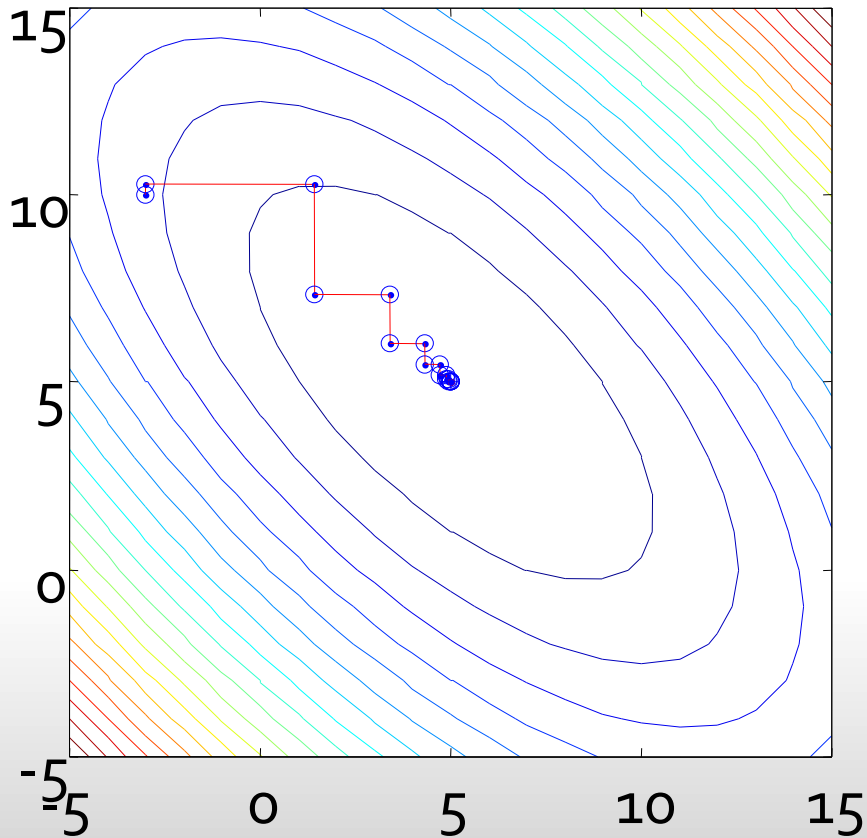
EXAMPLE



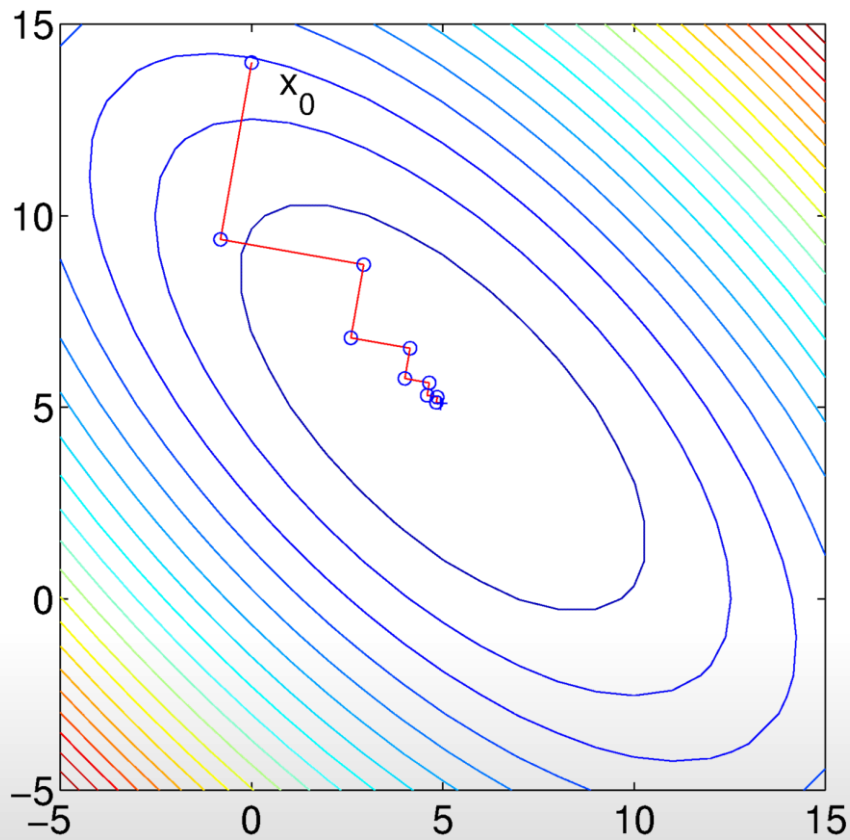
```
>> set(gcf, 'paperUnits', 'centimeters', 'paperposition', [1 1 9 6.6])  
>> print -dpdf % then go to pdf and paste
```

SWITCH TO MATLAB...



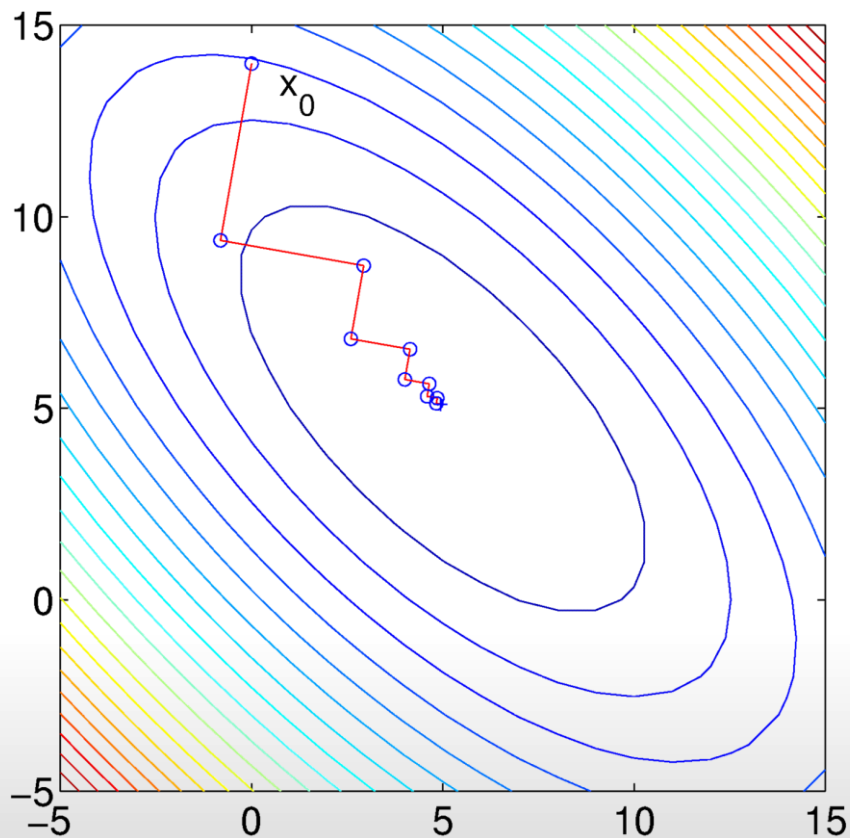


- Alternation is slow because valleys may not be axis aligned
- So try gradient descent?



Steepest descent ( $x_0 = [0, 14]$ )

- Alternation is slow because valleys may not be axis aligned
- So try gradient descent?

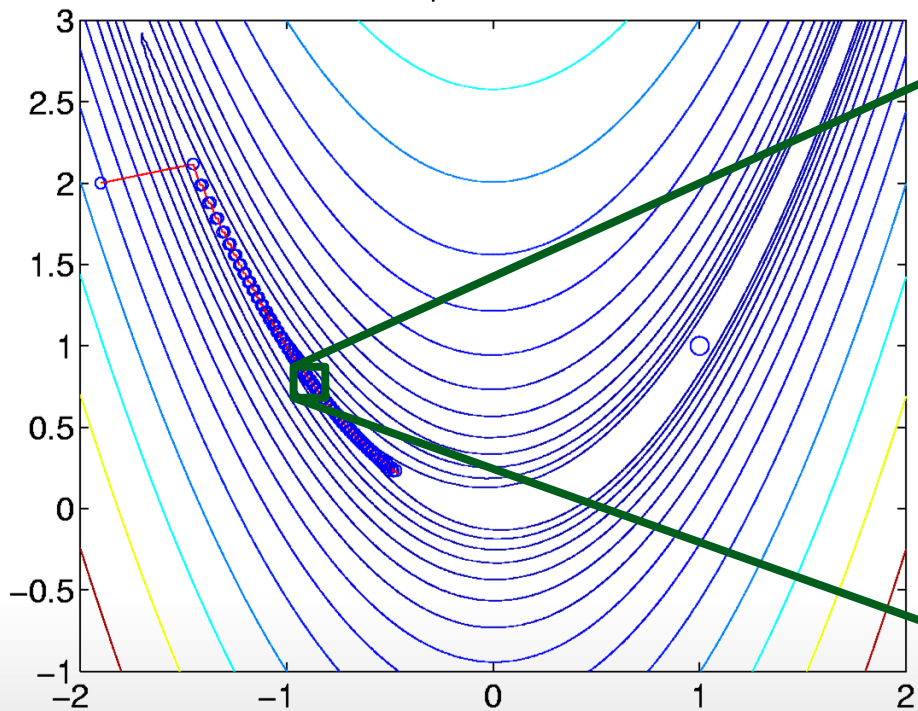


Steepest descent ( $x_0 = [0, 14]$ )

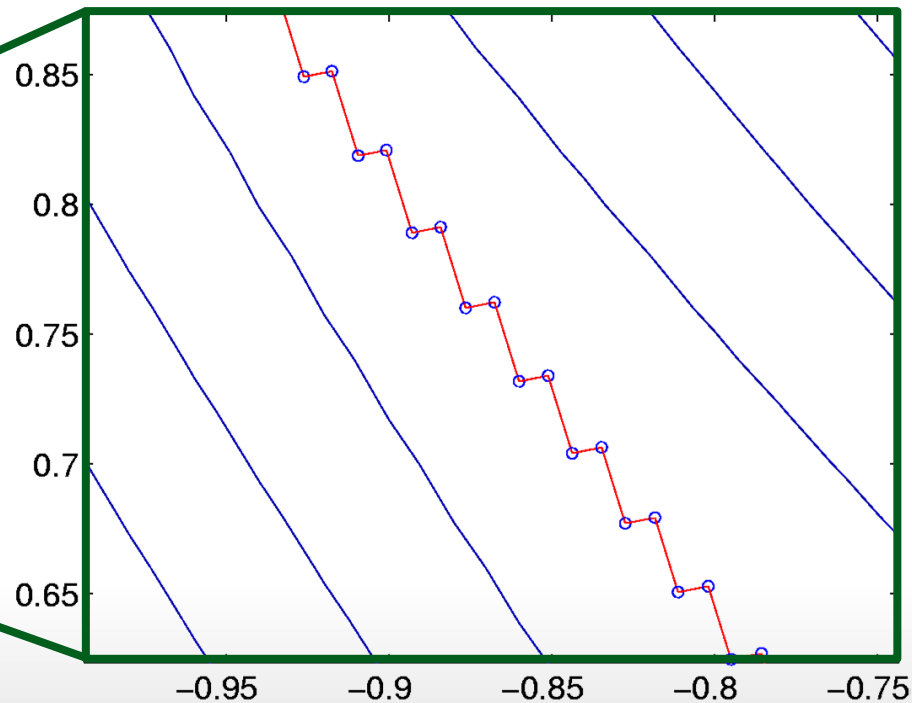
- Alternation is slow because valleys may not be axis aligned
- So try gradient descent?
- Note that convergence proofs are available for both of the above
- But so what?



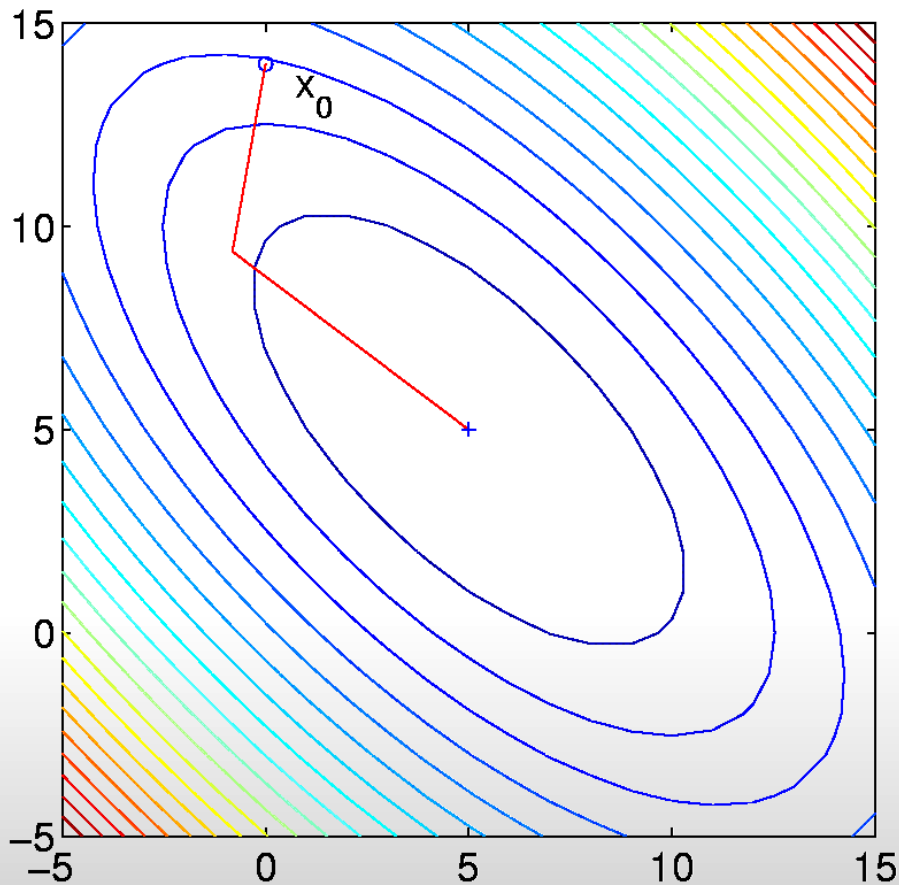
Steepest Descent



Steepest Descent

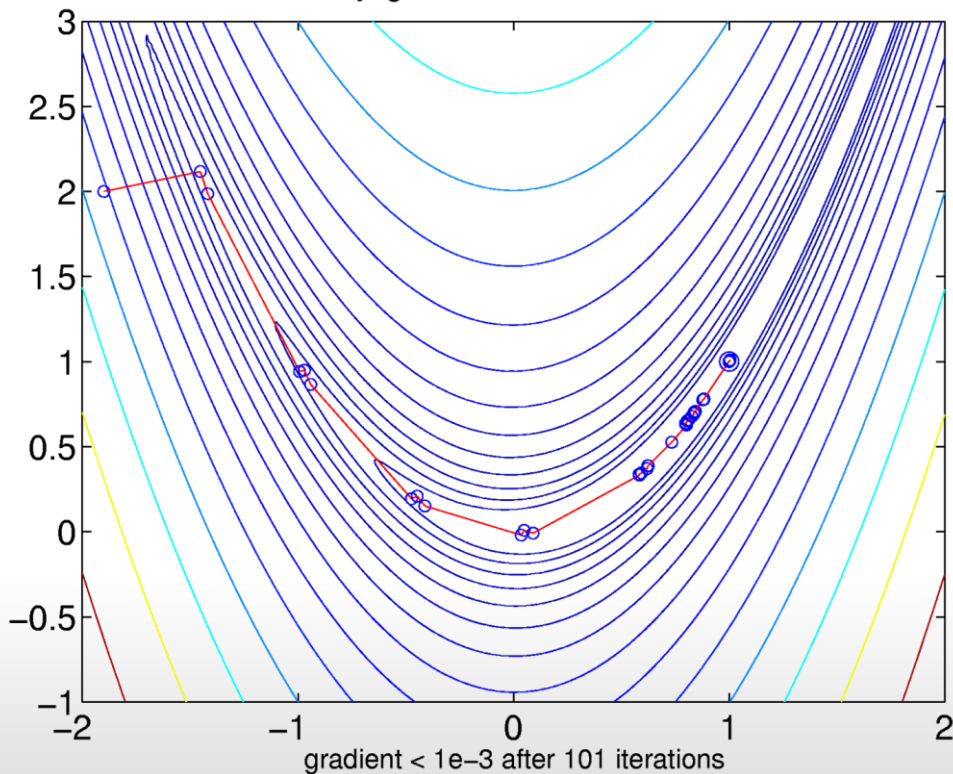


AND ON A HARD PROBLEM



- (Nonlinear) conjugate gradients
- Uses 1<sup>st</sup> derivatives only
- Avoids “undoing” previous work

## Conjugate Gradient Descent



- (Nonlinear) conjugate gradients
- Uses 1<sup>st</sup> derivatives only
- And avoids “undoing” previous work
- 101 iterations on this problem

BUT WE CAN DO BETTER...

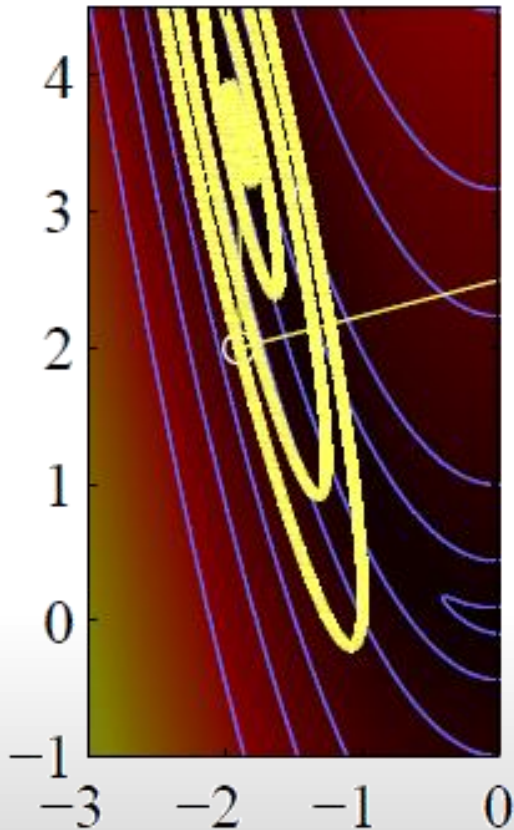
- Starting with  $\mathbf{x}$  how can I choose  $\boldsymbol{\delta}$  so that  $f(\mathbf{x} + \boldsymbol{\delta})$  is better than  $f(\mathbf{x})$ ?
- So compute

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^d} f(\mathbf{x} + \boldsymbol{\delta})$$

- But hang on, that's the same problem we were trying to solve?

- Starting with  $x$  how can I choose  $\delta$  so that  $f(x + \delta)$  is better than  $f(x)$ ?
- So compute

$$\begin{aligned} & \min_{\delta} f(x + \delta) \\ & \approx \min_{\delta} f(x) + \delta^{\top} g(x) + \frac{1}{2} \delta^{\top} H(x) \delta \\ & \quad g(x) = \nabla f(x) \\ & \quad H(x) = \nabla \nabla^{\top} f(x) \end{aligned}$$



- How does it look?

$$f(x) + \delta^\top g(x) + \frac{1}{2} \delta^\top H(x) \delta$$

$$g(x) = \nabla f(x)$$

$$H(x) = \nabla \nabla^\top f(x)$$



- Choose  $\delta$  so that  $f(x + \delta)$  is better than  $f(x)$ ?
- Compute

$$\min_{\delta} f + \delta^T g + \frac{1}{2} \delta^T H \delta$$

[derive]

$$\frac{d}{d\delta} = g + H\delta = 0$$

$$\delta = -H^{-1}g$$

$$\left( I + A^{-1} \right)^{-1}$$

$$\left( \frac{1}{k} \cdot \frac{1}{q+1} \right)$$

$$\left( A + I \right)^{-1}$$

Tom Minka  
Matrix derivatives

- Choose  $\delta$  so that  $f(x + \delta)$  is better than  $f(x)$ ?
- Compute

$$\min_{\delta} f + \delta^{\top} g + \frac{1}{2} \delta^{\top} H \delta$$

$$\delta = -H^{-1}g$$

```
>> use demos
```

```
>> demo_taylor_2d(0, 'newton', 'rosenbrock')
```

```
>> demo_taylor_2d(0, 'newton', 'sqrt_rosenbrock')
```

```
>> demo_taylor_2d(1, 'damped newton ls', 'rosenbrock')
```

- Choose  $\delta$  so that  $f(x + \delta)$  is better than  $f(x)$ ?
- Updates:

$$\delta_{\text{Newton}} = -H^{-1}g$$

$$\delta_{\text{GradientDescent}} = -\lambda g$$

- Updates:

$$\delta_{\text{Newton}} = -H^{-1}g$$

$$\delta_{\text{GradientDescent}} = -\lambda g$$

- So combine them:

$$\begin{aligned}\delta_{\text{DampedNewton}} &= -(H + \lambda^{-1}I_d)^{-1}g \\ &= -\lambda(\lambda H + I_d)^{-1}g\end{aligned}$$

- $\lambda$  small  $\Rightarrow$  conservative gradient step
- $\lambda$  large  $\Rightarrow$  Newton step

$\lambda = 10^{-3}; \lambda' = 3;$

**while**  $\lambda < 10^9$

$[f, \mathbf{g}, \mathbf{H}] = \text{error\_function}(\mathbf{x}_k)$

$\boldsymbol{\delta} = -(\mathbf{H} + \lambda \mathbf{I}) \backslash \mathbf{g}$

$\mathbf{x}_{new} = \mathbf{x}_k + \boldsymbol{\delta}$

if  $\text{error\_function}(\mathbf{x}_{new}) < f$ :

$\mathbf{x}_k = \mathbf{x}_{new}$

$\lambda = \lambda / \lambda'; \lambda' = 3$

else

$\lambda = \lambda \lambda'; \lambda' = 3\lambda'$

*% Perhaps Gauss-Newton for H*

*% Many ways to do this efficiently*

*% Decreased error, accept the new x*

*% Doing well—decrease  $\lambda$*

*% Doing badly—increase  $\lambda$  quick*

# Levenberg-Marquardt

- Just damped Newton with approximate  $H$
- For a special form of  $f$

$$f(x) = \sum_i f_i(x)^2 = \sum_i (\sqrt{g_i(x)})^2$$

- where  $f_i(x)$  are
  - zero-mean
  - small at the optimum



# Levenberg Marquardt

- Just damped Newton with approximate  $H$
- For a special form of  $f$

$$f(x) = \sum_i f_i(x)^2$$

$$\nabla f(x) = \sum_i \underbrace{2 f_i(x)} \underbrace{\nabla f_i(x)}$$

$$\nabla \nabla^T f(x) = \sum_i \underbrace{2 \nabla f_i(x) \cdot \nabla^T f_i(x)} + \cancel{2 f_i(x) \nabla \nabla^T f_i(x)} \rightarrow 0$$

# Levenberg Marquardt

- Just damped Newton with approximate  $H$
- For a special form of  $f$

$$f(x) = \sum_i f_i(x)^2$$

$$\nabla f(x) = \sum_i 2f_i(x) \nabla f_i(x)$$

$$\nabla \nabla^T f(x) = 2 \sum_i \left( \cancel{f_i(x) \nabla \nabla^T f_i(x)} \approx 0 \right) + \underbrace{\nabla f_i(x)}_{\mathbb{R}^d} \nabla^T f_i(x)$$

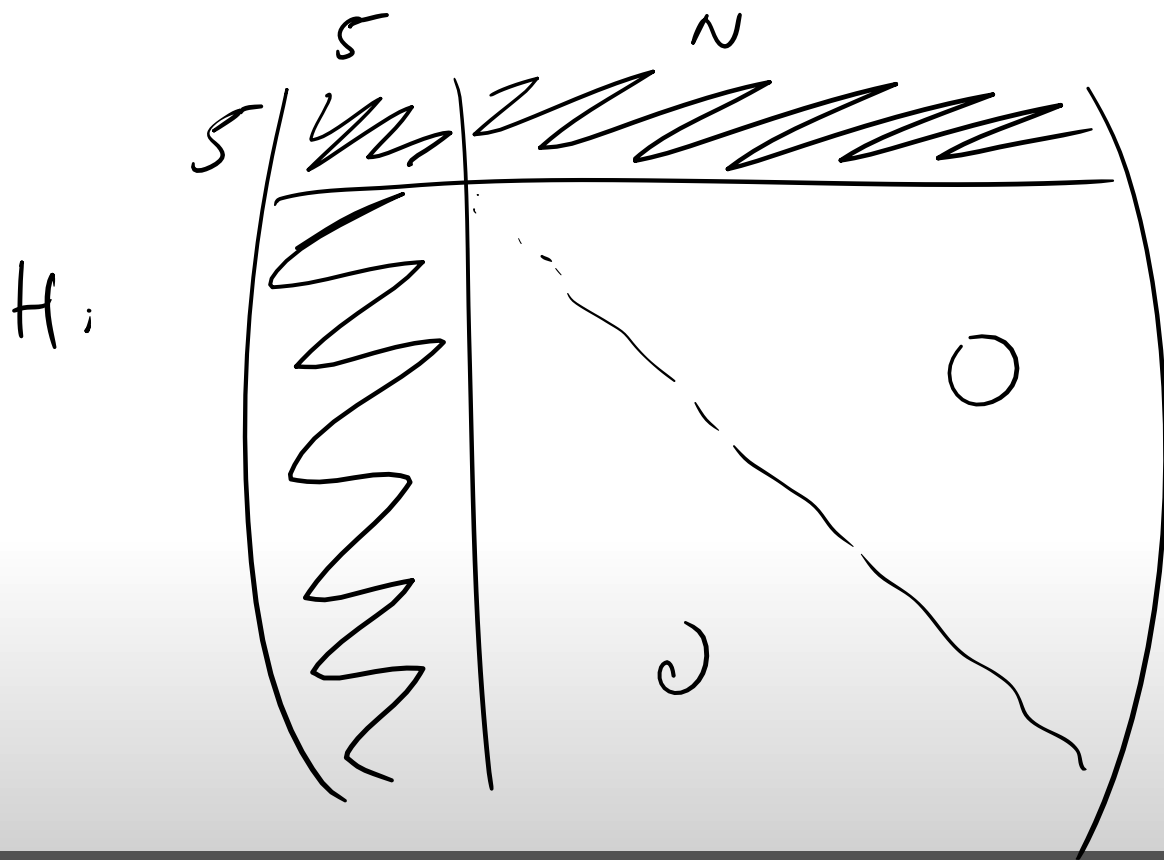
- Not  $O(n^3)$  if you exploit sparsity of Hessian or Jacobian

$$\delta = -\lambda g$$

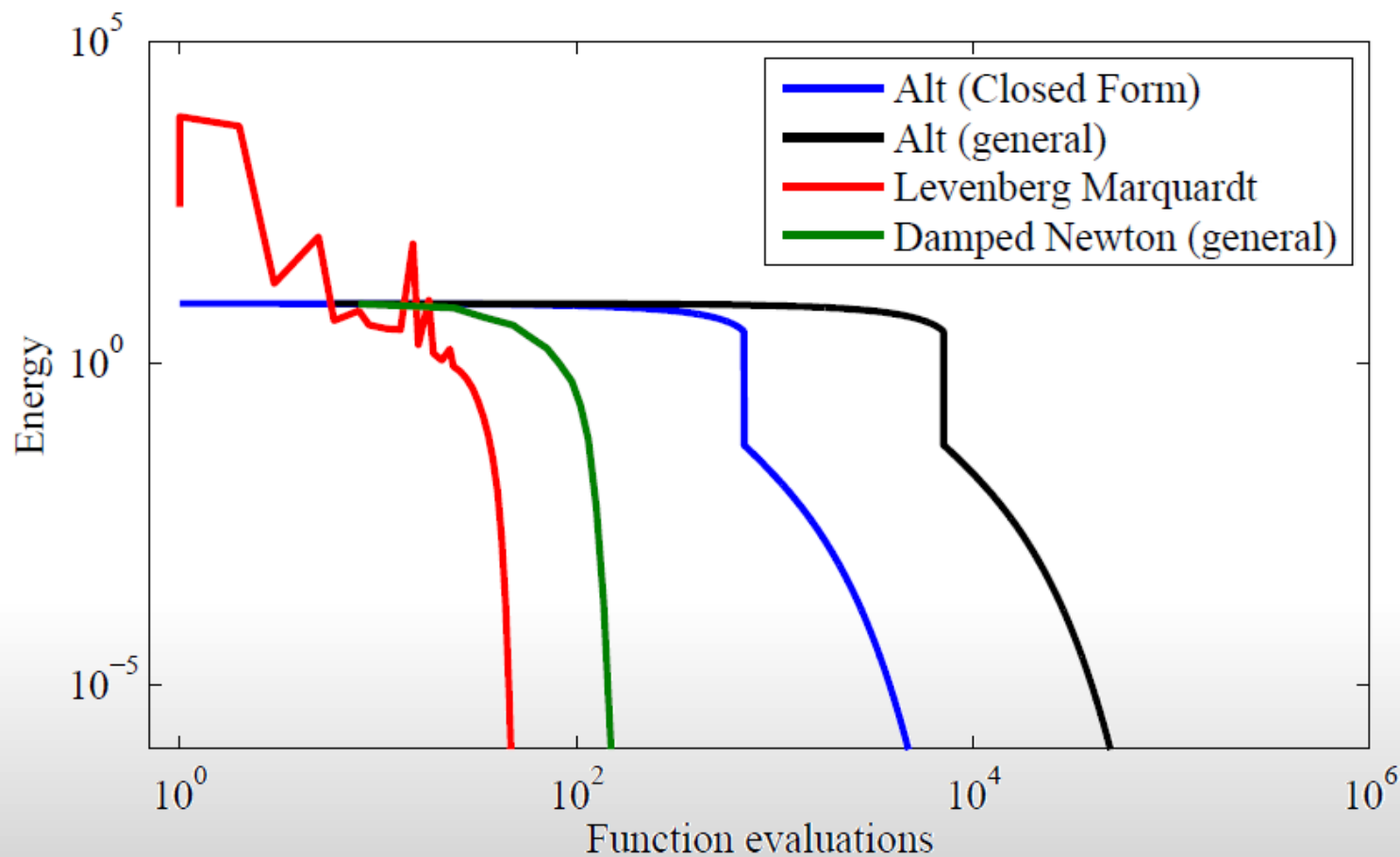
$$\delta = -H' s$$

8  
100,000 x 100,000

$$J = \begin{bmatrix} \nabla f_1(x) \\ \vdots \\ \nabla f_n(x) \end{bmatrix}$$

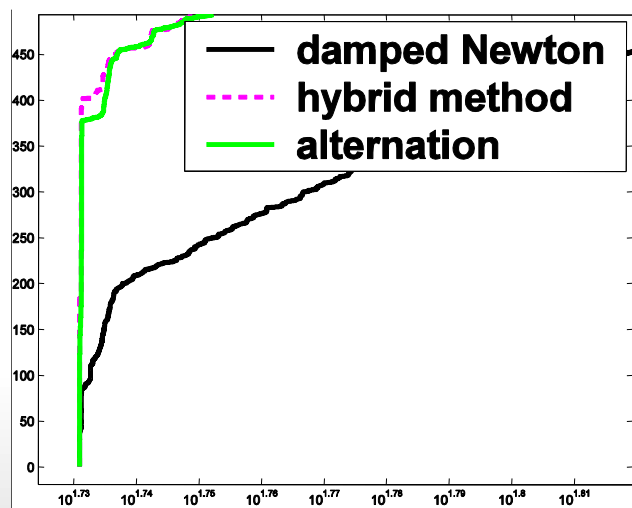


$$O(N^S) \rightarrow O(N)$$





GIRAFFE



500 runs

```
for k=1:500
     $x_0 = randn(n, 1);$ 
     $x^* = minimize(f, x_0);$ 
     $E[k] = f(x^*)$ 
end
plot(sort(E));
```



GIRAFFE

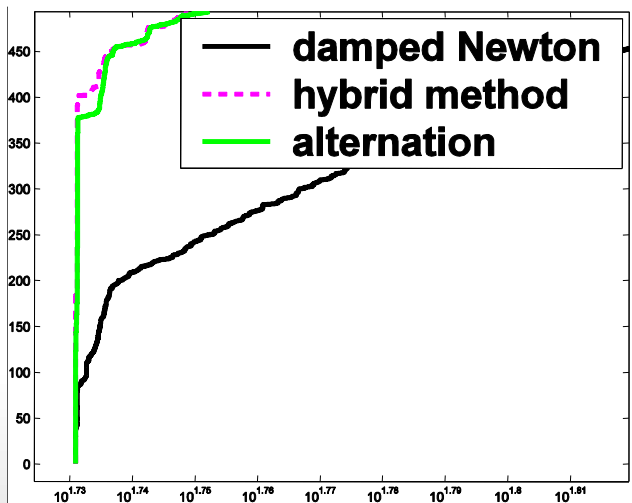


FACE

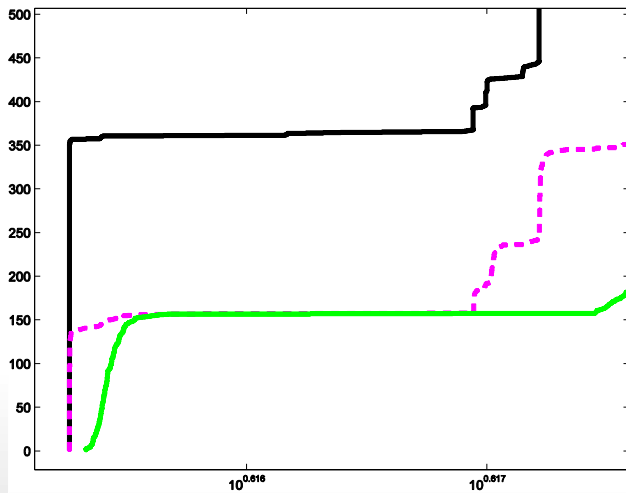


DINOSAUR

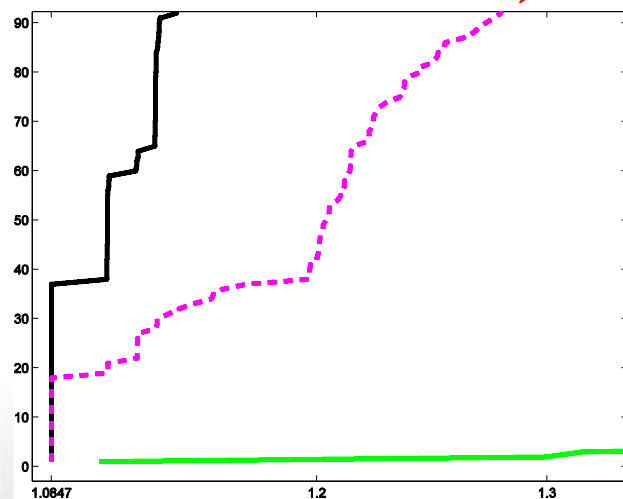
(114 uuuuuuuu) ~~u~~



500 runs



1000 runs



1000 runs

CONCLUSION: YMMV



- On many problems, alternation is just fine
  - Indeed always start with a couple of alternation steps
- Computing 2<sup>nd</sup> derivatives is a pain
  - But you don't need to for LM
- But just alternation is not
  - Unless you're willing to problem-select
- Convergence guarantees are fine, but practice is what matters
- Inverting the Hessian is rarely  $O(n^3)$

There is no universal optimizer

$$\nabla f = \frac{1}{\mu} \begin{bmatrix} f(x + e_1) - f(x) \\ \vdots \\ f(x + e_d) - f(x) \end{bmatrix}$$

Handwritten notes:  $(1, 0, 0, \dots)$  above the first row,  $\frac{df}{dx} = \frac{f(x+\delta) - f(x)}{\delta}$  to the right.

- Surprisingly accurate for e.g.  $\mu = 10^{-5}$  (in double prec.)
- Incredibly slow.. Unless (see next slide)
- Useful for checking your analytic derivatives
- Incredibly slow. Try Powell or Simplex instead. ?
- Central differences twice as slow, somewhat more accurate

- Normally try  $e_1$  to  $e_d$  sequentially
- But if we know the nonzero structure of the Jacobian, can go rather faster.

$$g = \nabla f \quad g_i = \frac{f(x + \delta e_i) - f(x)}{\delta}$$

$$[f(x)]_i = x_i^2$$

an-optimize problem

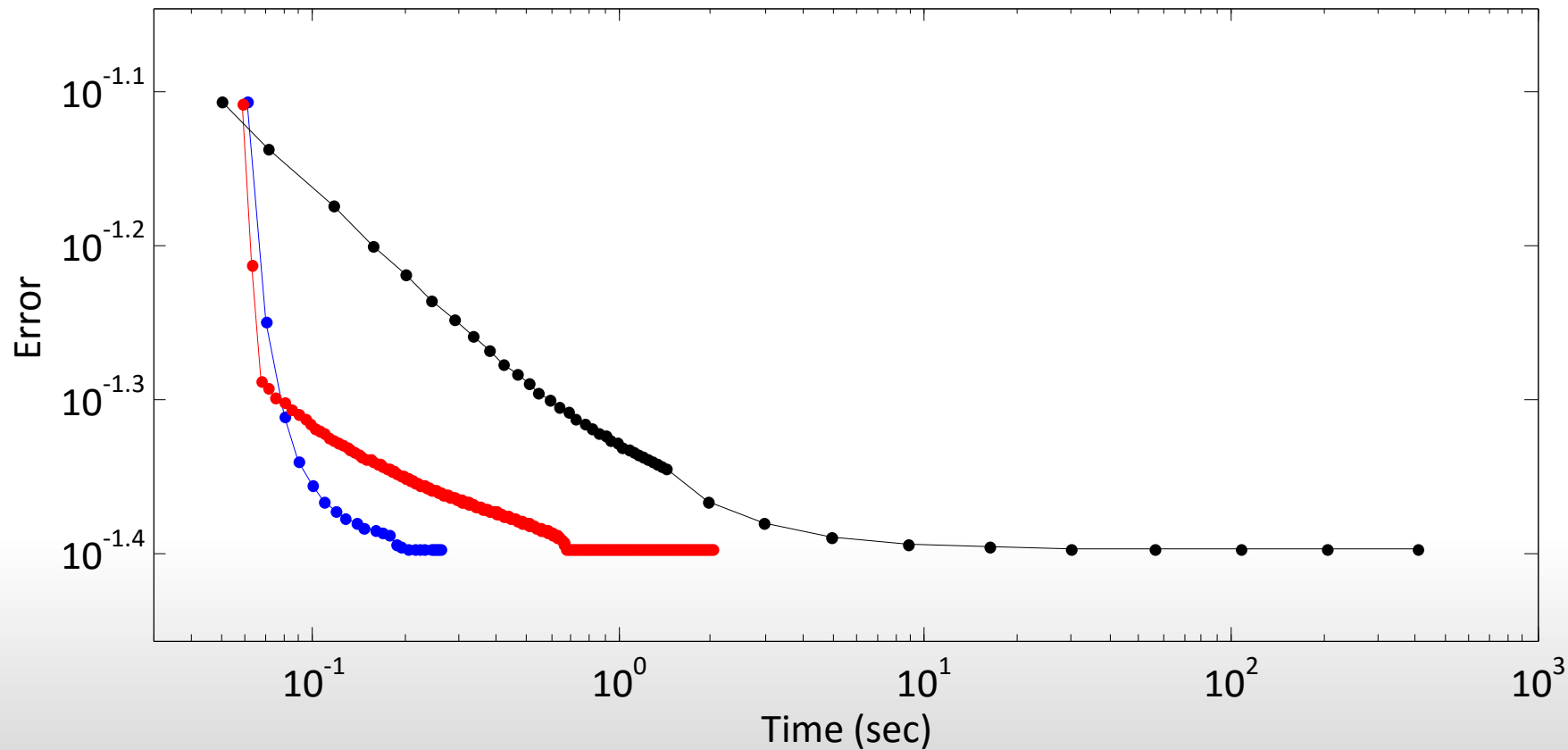
- We're minimizing  $f(x)$
- Many algorithms will be happier if entries of  $x$  are all "around 1".
  - E.g. don't have angle in degrees and distances in km
- Many algorithms may want  $f$  values to be "close to  $x$  or close to zero at the optimum".
  - Specifically, think about roundoff in quantities like  $f(x_{k+1}) - f(x_k)$  being compared to numbers like  $10^{-6}$

- What about stochastic gradient descent?
  - You can do analogous 2<sup>nd</sup> order things.
- What about LBFGS?
  - I haven't had much success with it, other folk love it...
- I tried lsqnonlin and it was really slow—why?
  - Wrong derivatives (e.g. finite-differences)
  - Didn't use sparsity correctly — *just use mls*
  - Didn't set "options.Algorithm" or "options.LargeScale".

## ■ Resources:

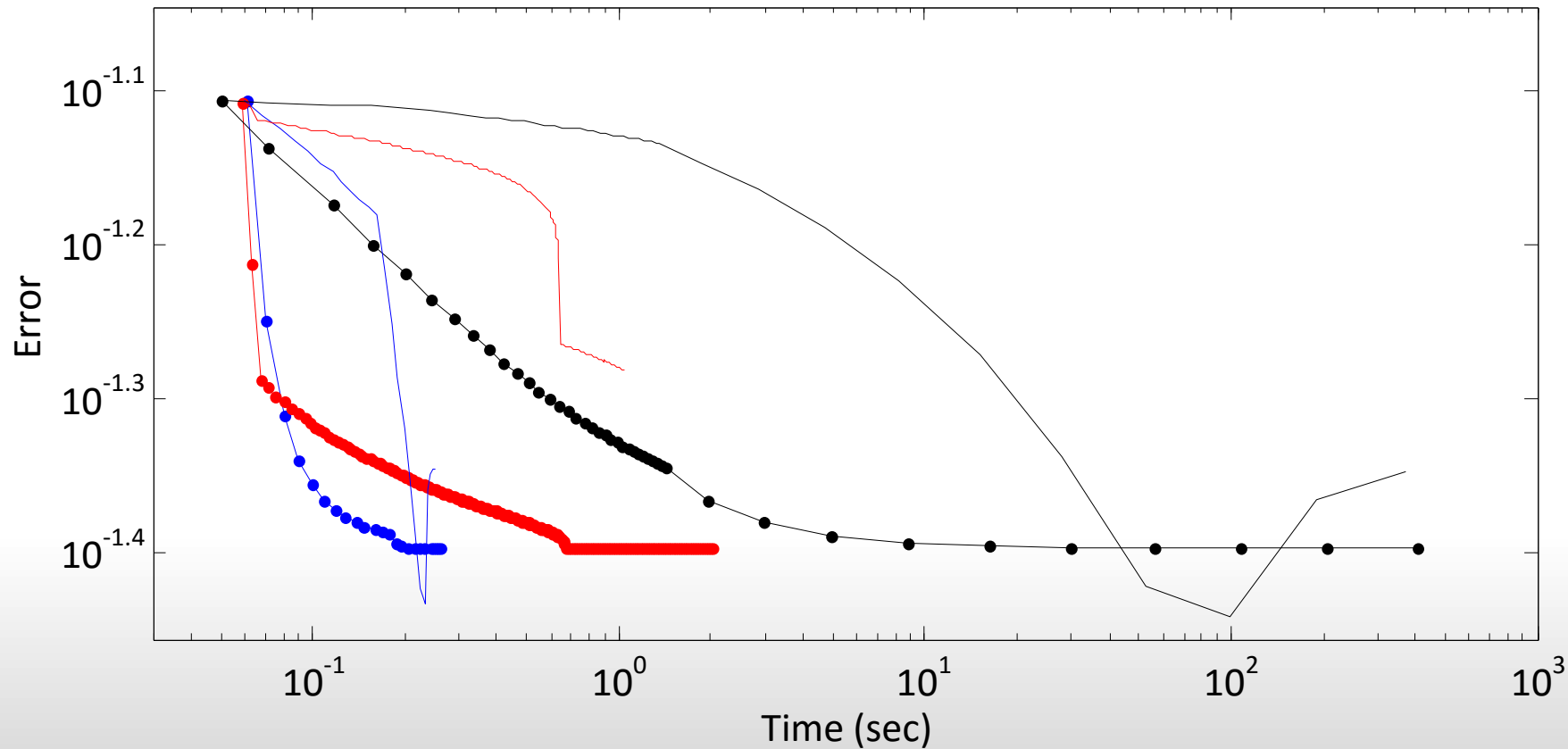
1. Matlab fminsearch and fminunc documentation
2. [awful.codeplex.com](http://awful.codeplex.com) au\_optimproblem
3. Tom Minka webpage on matrix derivatives
4. Google “ceres” solver
5. UToronto “Theano” system for Python

- Gotchas with Isqnonlin
  - `opts.LargeScale = 'on';`
  - `opts.Jacobian = 'on';`
- Need non-rank-def J?
- Need to implement JacobMult?



CONVERGENCE CURVES, ONE INSTANCE

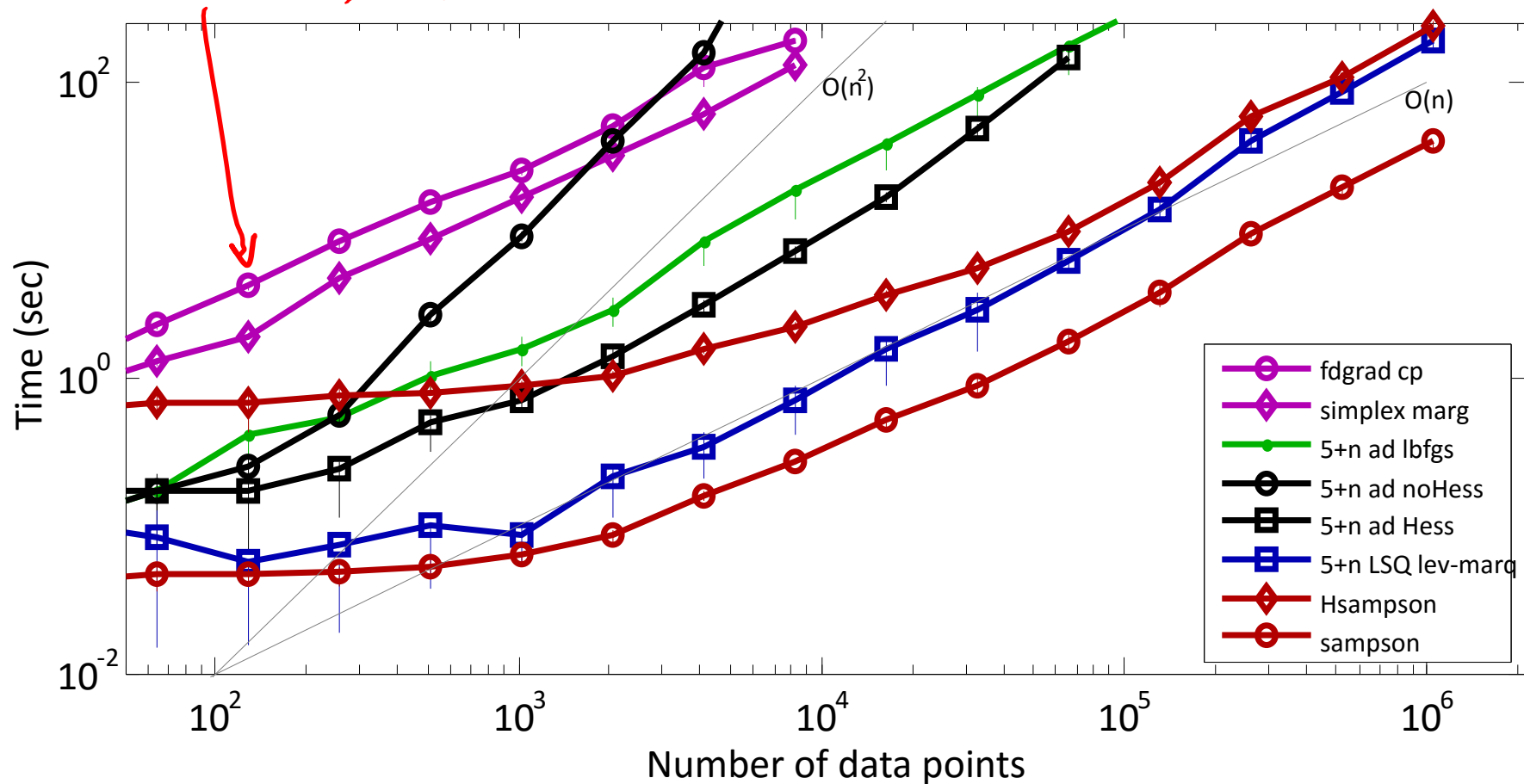




CONVERGENCE CURVES, ONE INSTANCE

$f_{\min,unc}(f, x_0)$

Timings for n-point ellipse fit

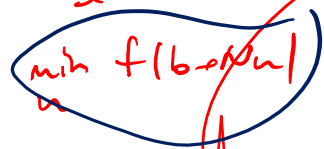


$$\min_x f(x)$$

$$Ax = b$$

$$x = b + Nu$$

$$\min_u f(b + Nu)$$



$$\sum_i f_i^2(x) + p(\cdot)$$

$$a_1, \dots, a_m$$

$$\sum a_i = 1$$

$$a_i > 0 \quad \forall i$$

# EXAMPLE: SHAPE FITTING

$$\min_{|q|=1} f(q)$$

$$g(x) := f\left(\frac{x}{|x|}\right) \quad \nabla g = \nabla f\left(\frac{x}{|x|}\right) \cdot \frac{x}{|x|^2}$$

$$\sum_{i=1}^{10000} \|x_i - f(\theta_i, y)\|^2$$

$\xrightarrow{10000} \theta_1 \dots \theta_{10000} \quad \xrightarrow{10000} y$

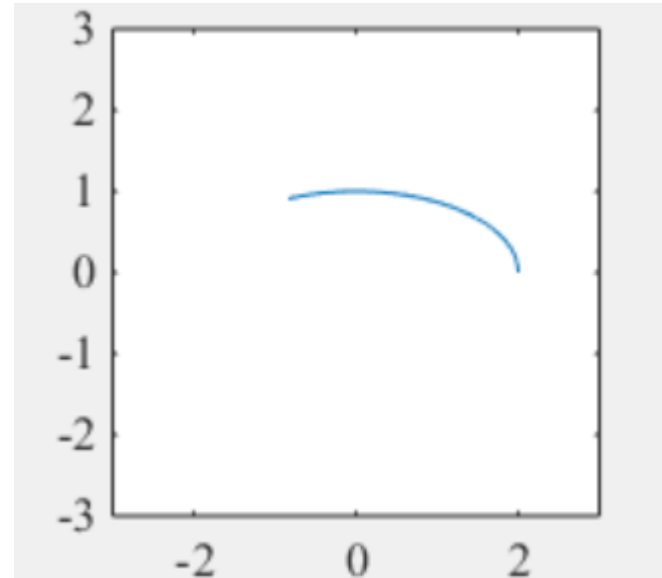
$J * x$   
 $J^T * x$

```
t = 0:.01:2;
```

```
plot(cos(t)*2, sin(t));
```

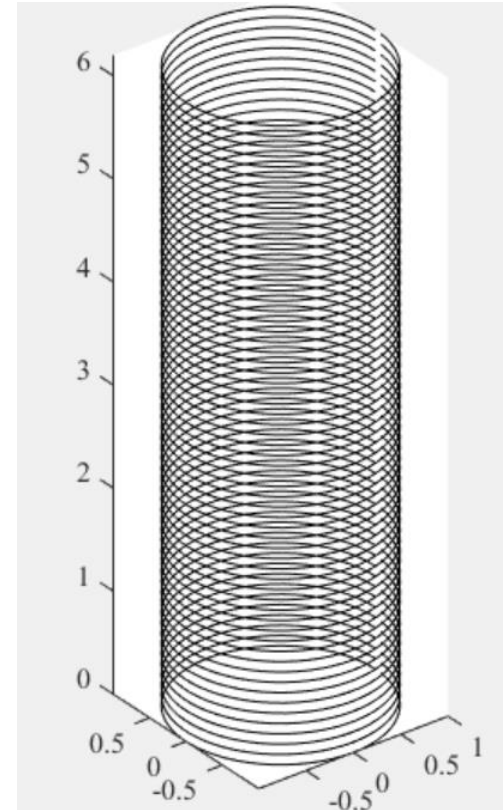
```
t = 0:.01:2;
```

```
plot(cos(t)*2, sin(t));
```



```
>> u = 0:.1:2*pi; v = 0:.1:2*pi;  
>> l = ones(size(v));  
>> u = u'*l;  
>> v = l*v;  
>> plot3(cos(u), sin(u), v, 'k.')
```

```
>> u = 0:.1:2*pi; v = 0:.1:2*pi;  
>> l = ones(size(v));  
>> u = u'*l;  
>> v = l*v;  
>> plot3(cos(u), sin(u), v, 'k.')
```

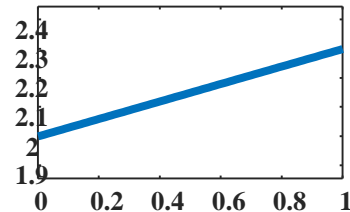




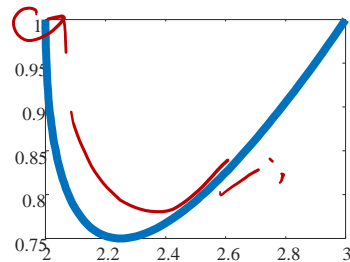
# What is a shape?

- Functions
- Curves
- Surfaces

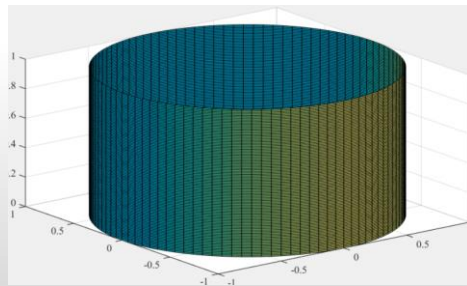
```
function y(x::Real)::Real = .3*x + 2
```



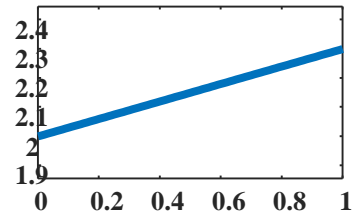
```
function C(t::Real)::Point2D =  
    Point2D(t^2 + 2, t^2 - t + 1)
```



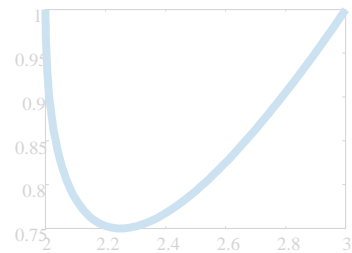
```
function S(u::Real, v::Real)::Point3D =  
    Point3D(cos(u), sin(u), v)
```



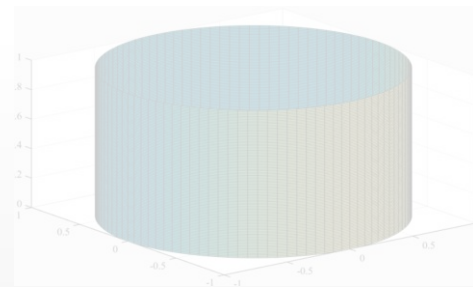
```
function y(x::Real)::Real = .3*x + 2
```

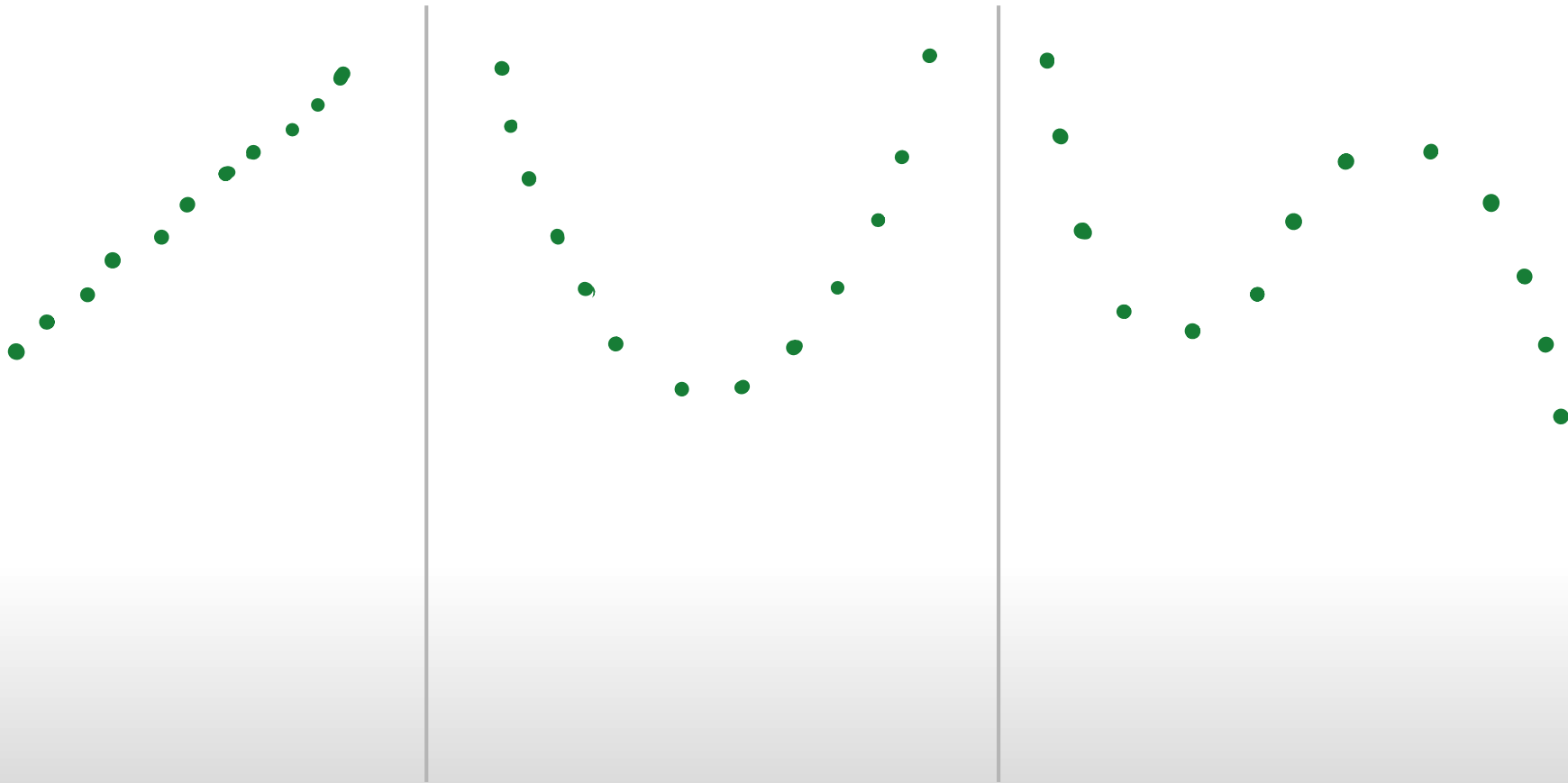


```
function C(t::Real)::Point2D =  
    Point2D(t^2 + 2, t^2 - t + 1)
```

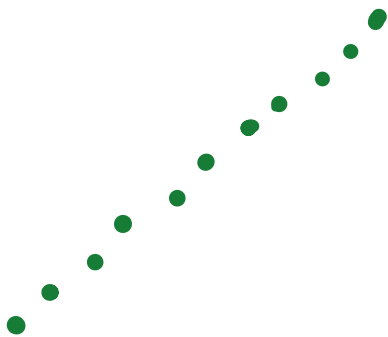


```
function S(u::Real, v::Real)::Point3D =  
    Point3D(cos(u), sin(u), v)
```

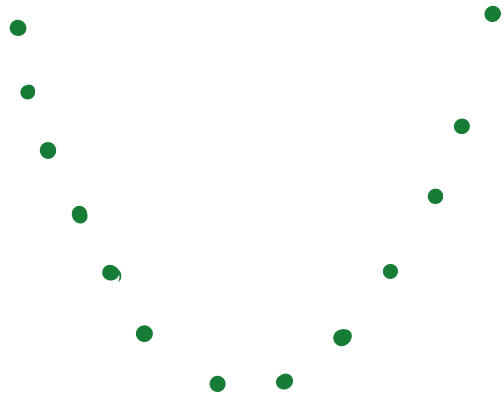




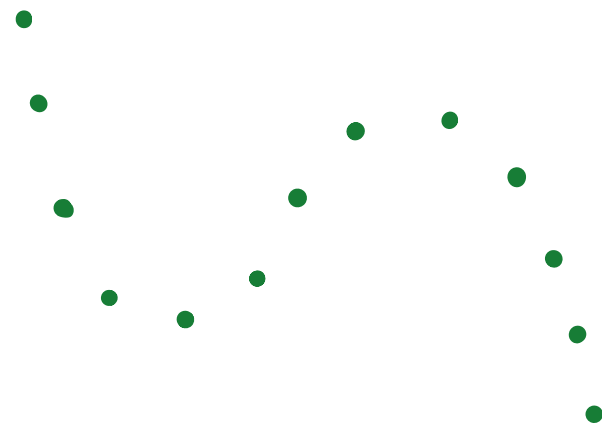
SHAPES DESCRIBE DATA



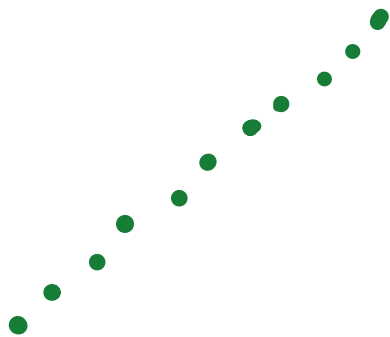
$$y = ax + b$$



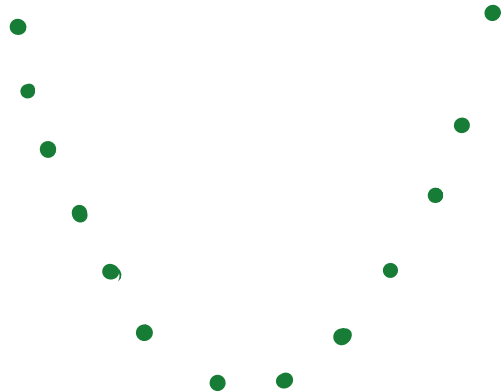
$$y = ax^2 + bx + c$$



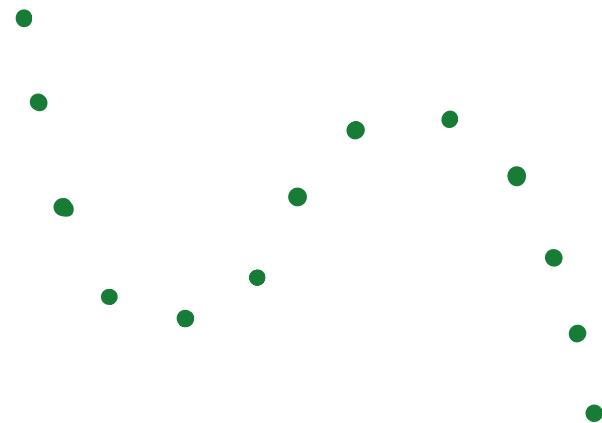
$$y = ax^3 + bx^2 + cx + d$$



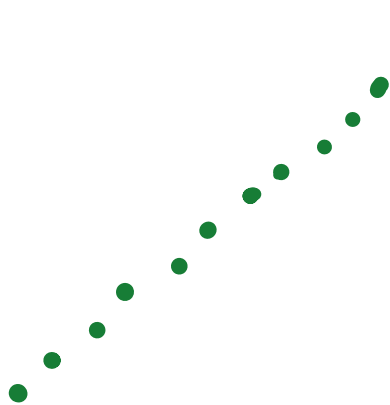
$$y = ax + b$$



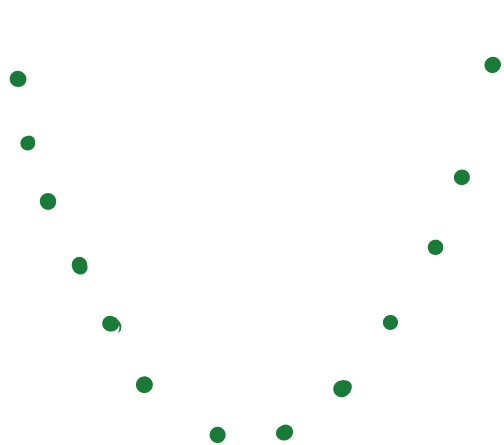
$$y = ax^2 + bx + c$$



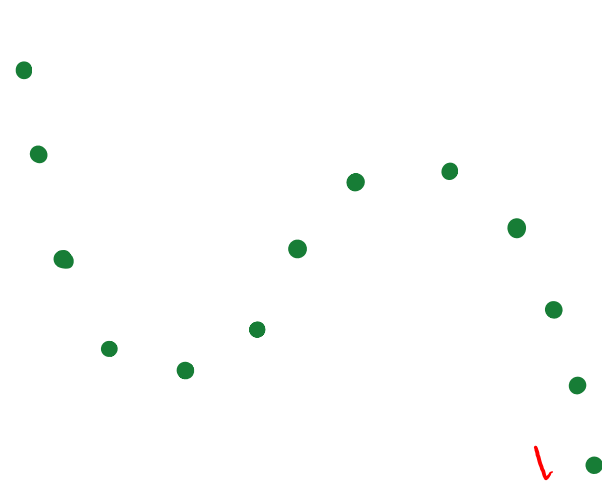
$$y = \sin(x) + ax + b$$



$$y = ax + b$$



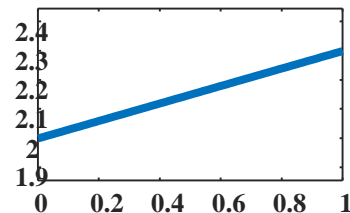
$$y = ax^2 + bx + c$$



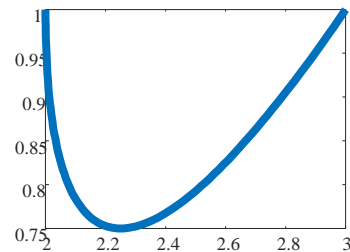
$$y = \begin{cases} \text{if } x < a \\ (x - b)^2 + c \\ \text{else} \\ -(x - d)^2 + e \end{cases}$$

↗  $\frac{dy}{dx} =$  if  $(x < a)$   
 $2(x - b)$   
 else  
 $-2(x - d)$

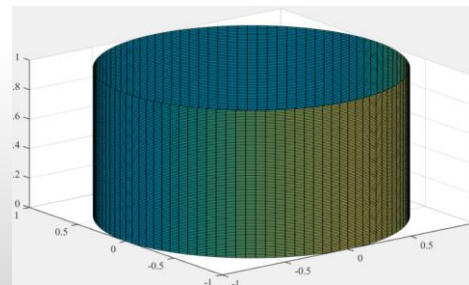
```
function y(x::Real)::Real = .3*x + 2
```



```
function C(t::Real)::Point2D =  
    Point2D(t^2 + 2, t^2 - t + 1)
```

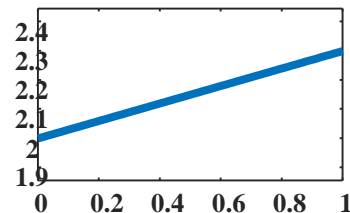


```
function S(u::Real, v::Real)::Point3D =  
    Point3D(cos(u), sin(u), v)
```

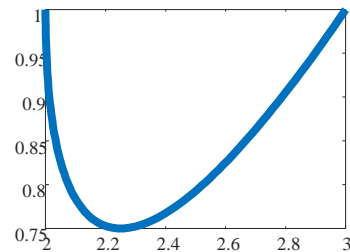




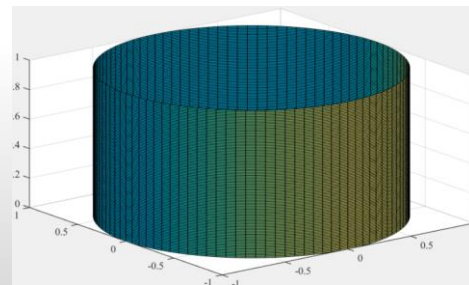
```
function y(x::Interval)::Real = .3*x + 2
```



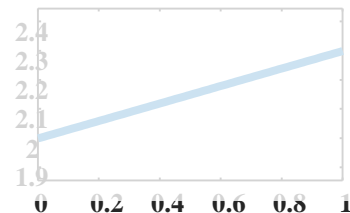
```
function C(t::Interval)::Point2D =  
    Point2D(t^2 + 2, t^2 - t + 1)
```



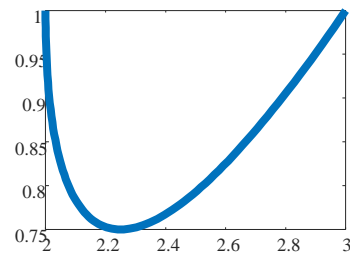
```
function S(u::Interval, v::Real)::Point3D =  
    Point3D(cos(u), sin(u), v)
```



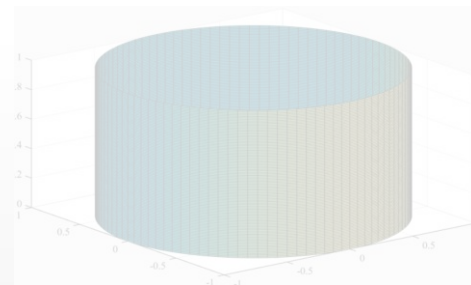
```
function y(x::Interval)::Real = .3*x + 2
```



```
function C(t::Interval)::Point2D =  
    Point2D(t^2 + 2, t^2 - t + 1)
```



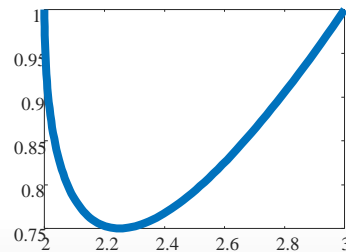
```
function S(u::Interval, v::Real)::Point3D =  
    Point3D(cos(u), sin(u), v)
```



```
abstract Curve {  
    method eval(t::Interval)::Point2D  
};
```

```
abstract Curve {  
    method eval(t::Interval)::Point2D  
};
```

```
type Conic < Curve {  
    eval(t) =  
        Point2D(t^2 + 2, t^2 - t + 1)  
};
```



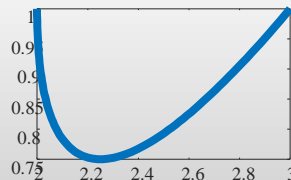
```

abstract Curve {
    method eval(t::Interval)::Point2D
};

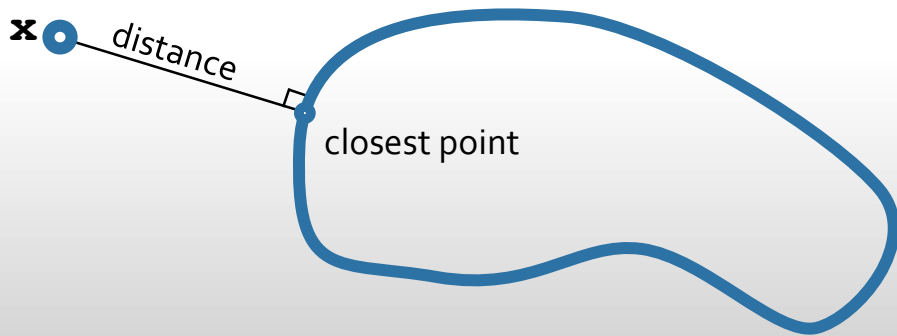
type Conic < Curve {
     $\theta$ ::Real[]; // Shape parameters
    eval(t) =
        Point2D(  $\theta[0]*t^2 + \theta[1]*t + \theta[2]$ ,
                   $\theta[3]*t^2 + \theta[4]*t + \theta[5]$  )
};

Conic([1,0,2,1,-1,1])

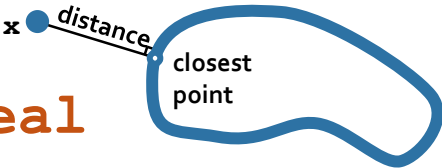
```



```
abstract Curve {  
    method eval(t :: Interval) :: Point2D  
  
    method distance(x :: Point2D) :: Real  
    method closest_point(x :: Point2D) :: Point2D  
};
```



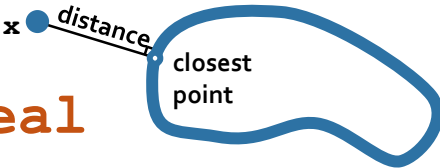
```
abstract Curve {  
    method eval(t::Interval)::Point2D  
  
    method distance(x::Point2D)::Real  
    method closest_point(x::Point2D)::Point2D  
};
```



A diagram illustrating the concept of distance from a point to a curve. A point labeled 'x' is shown to the left of a blue, irregular closed curve. A line segment labeled 'distance' connects point 'x' to the curve. The point on the curve is labeled 'closest point'.

```
distance(x) = norm(x - this.closest_point(x))
```

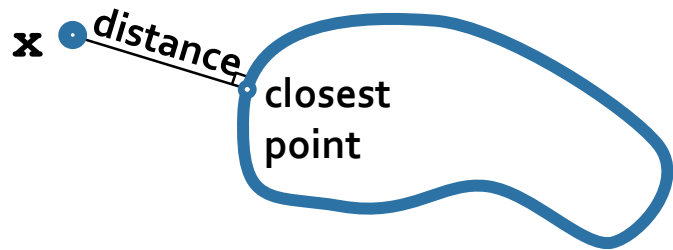
```
abstract Curve {  
  method eval(t :: Interval) :: Point2D  
  
  method distance(x :: Point2D) :: Real  
  method closest_point(x :: Point2D) :: Point2D  
};
```



```
distance(x) =  
  minimize( $\lambda$ (t) norm(this.eval(t) - x), 0.0)
```



```
abstract Curve {  
    method eval(t::Interval)::Point2D  
    method distance(x::Point2D)::Real  
    ...  
}
```



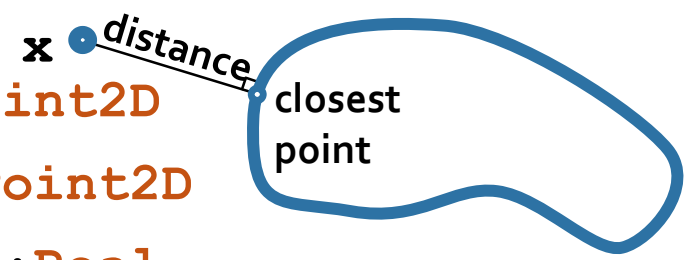
```
function f(t) = norm(this.eval(t) - x)^2  
distance(x) = minimize(f, Interval::Min)
```

```
function minimize(f, t)  
    while not converged  
        t -=  $\alpha$  * f'(t) // Compute derivative
```

```

abstract Curve {
  method eval (t :: Interval) :: Point2D
  method eval' (t :: Interval) :: Point2D
  method distance (x :: Point2D) :: Real
  method closest_point (x :: Point2D) :: Point2D
};

```



A diagram illustrating the concept of distance from a point  $x$  to a curve. A blue dot labeled  $x$  is shown above a blue curve. A line segment labeled "distance" connects  $x$  to the curve. A blue oval callout points to the intersection on the curve, labeled "closest point".

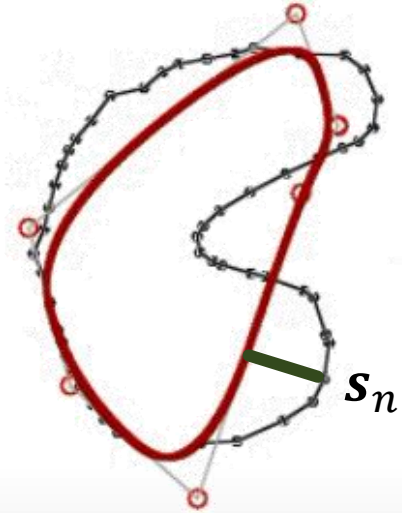
$$\begin{aligned}
 y &= \text{if } t < a \\
 &\quad (t - b)^2 + c \\
 &\text{else} \\
 &\quad f(t - d)^2 + e
 \end{aligned}$$

$$\begin{aligned}
 y' &= \text{if } t < a \\
 &\quad 2(t - b) \\
 &\text{else} \\
 &\quad 2f(t - d)
 \end{aligned}$$

# Shape, meet thy data

# Sum-of-min problems

$$\min_{\theta} \sum_{n=1}^N C(\theta).closest\_point(\mathbf{s}_n)$$

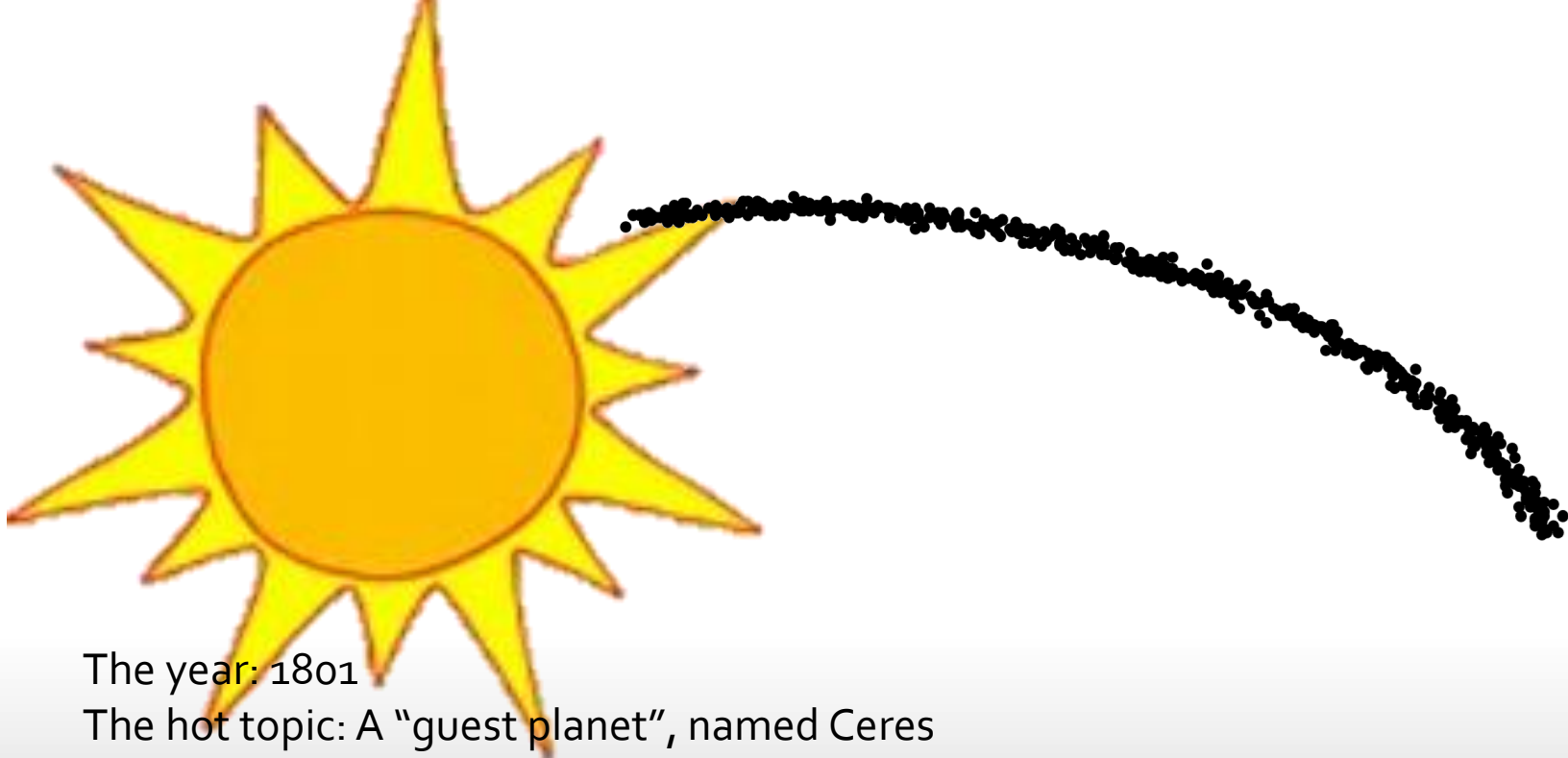


# AN EXEMPLARY PROBLEM

“Based on a true story”, not necessarily historically accurate

**Note well:** this problem is a good proxy for much more realistic problems:

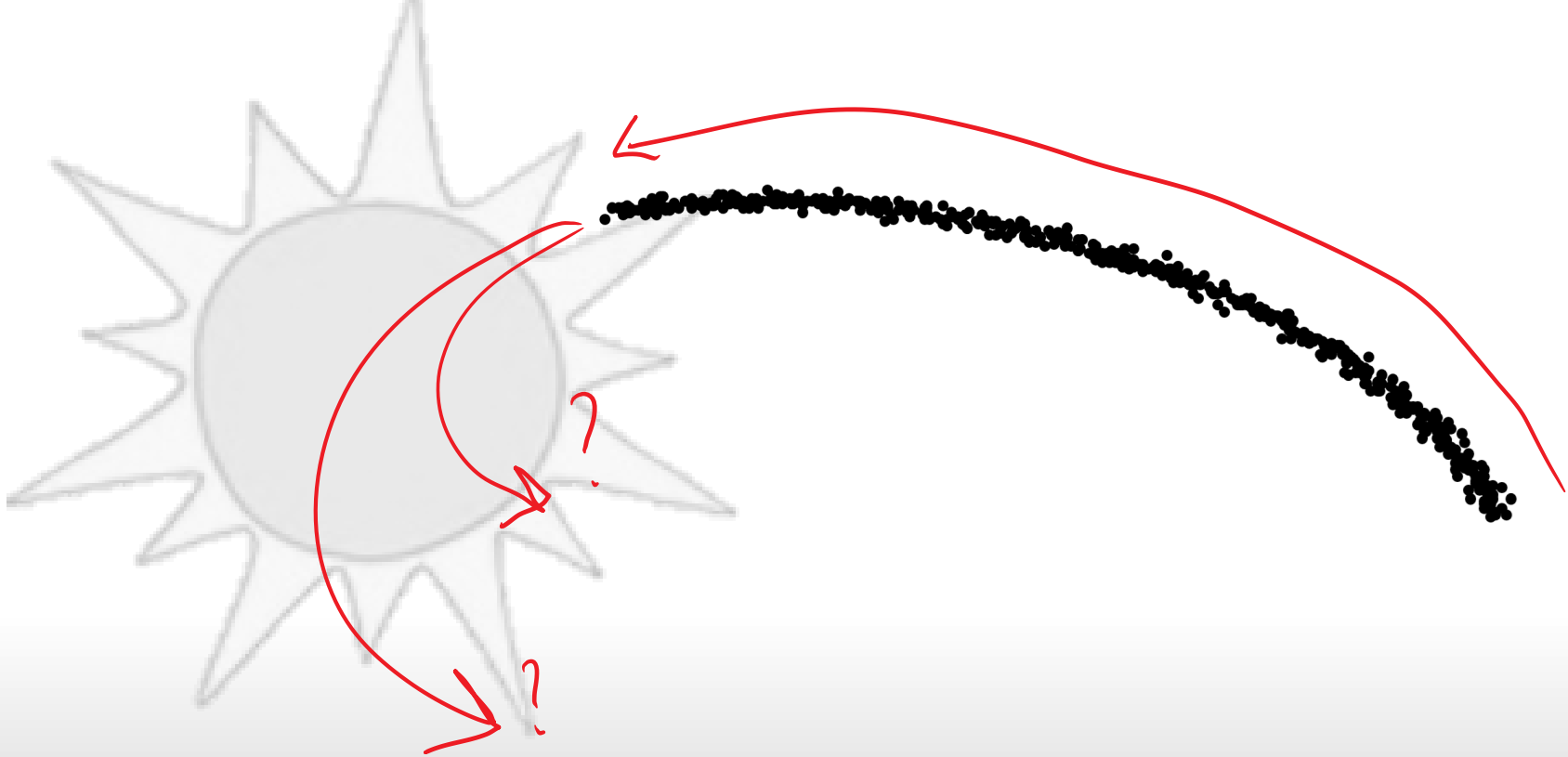
1. Stereo camera calibration
2. Multiple-camera bundle adjustment
3. Surface fitting, e.g. subdivision surfaces to range data, realtime hand tracking
4. Matrix completion
5. Image denoising.



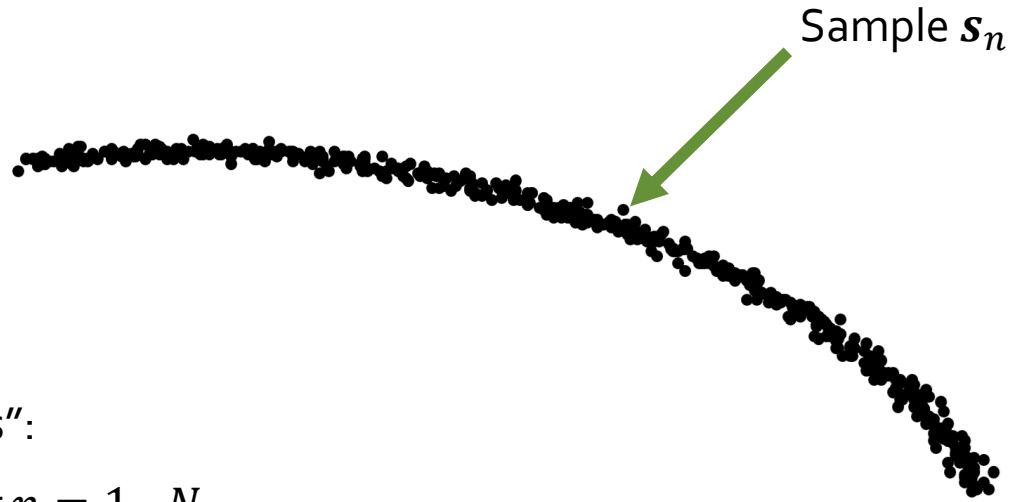
The year: 1801

The hot topic: A “guest planet”, named Ceres

The big question: Where will it reappear?



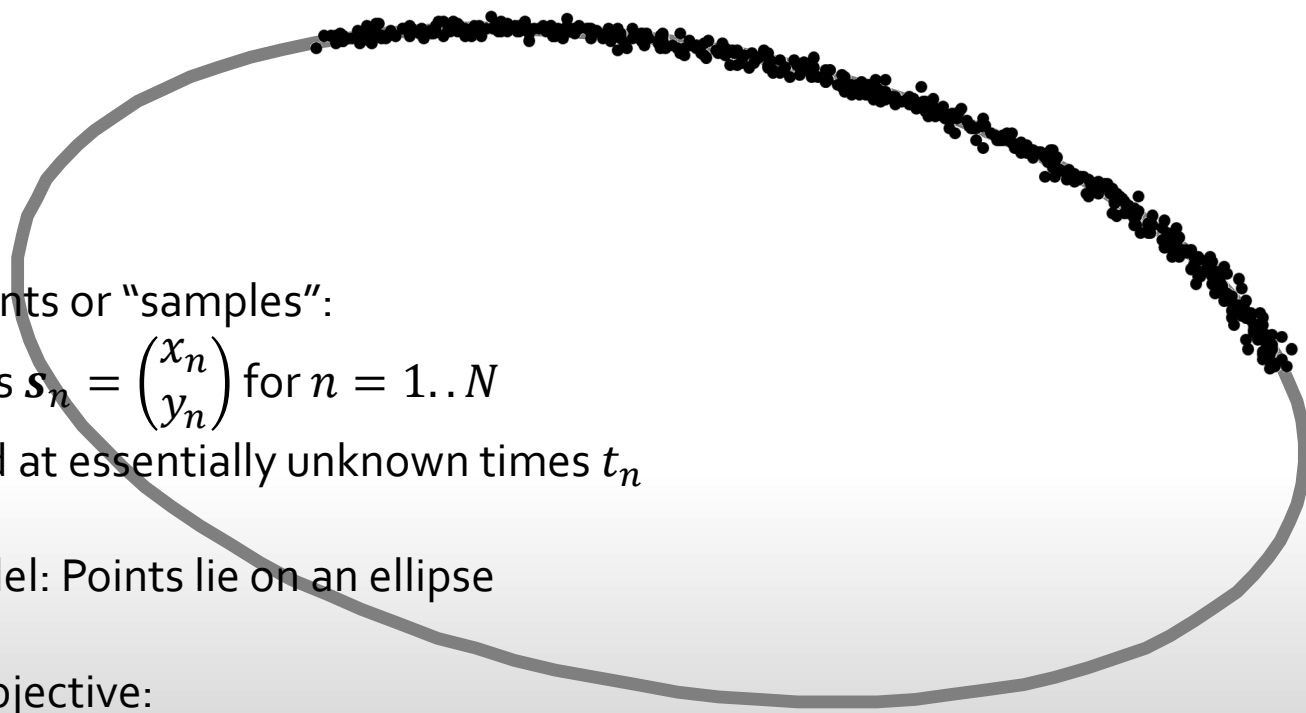
AN EXEMPLARY PROBLEM



Measurements or “samples”:

- 2D points  $s_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$  for  $n = 1..N$
- Captured at essentially unknown times  $t_n$





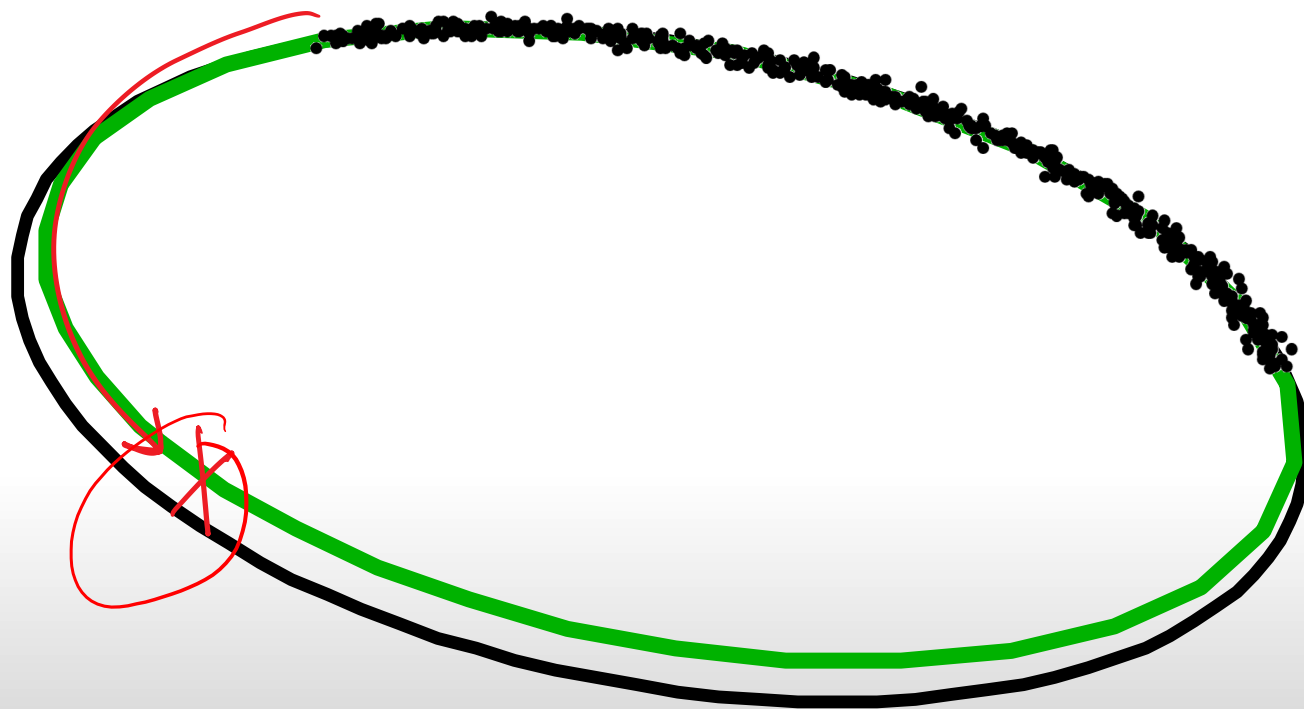
Measurements or “samples”:

- 2D points  $\mathbf{s}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$  for  $n = 1..N$
- Captured at essentially unknown times  $t_n$

Known model: Points lie on an ellipse

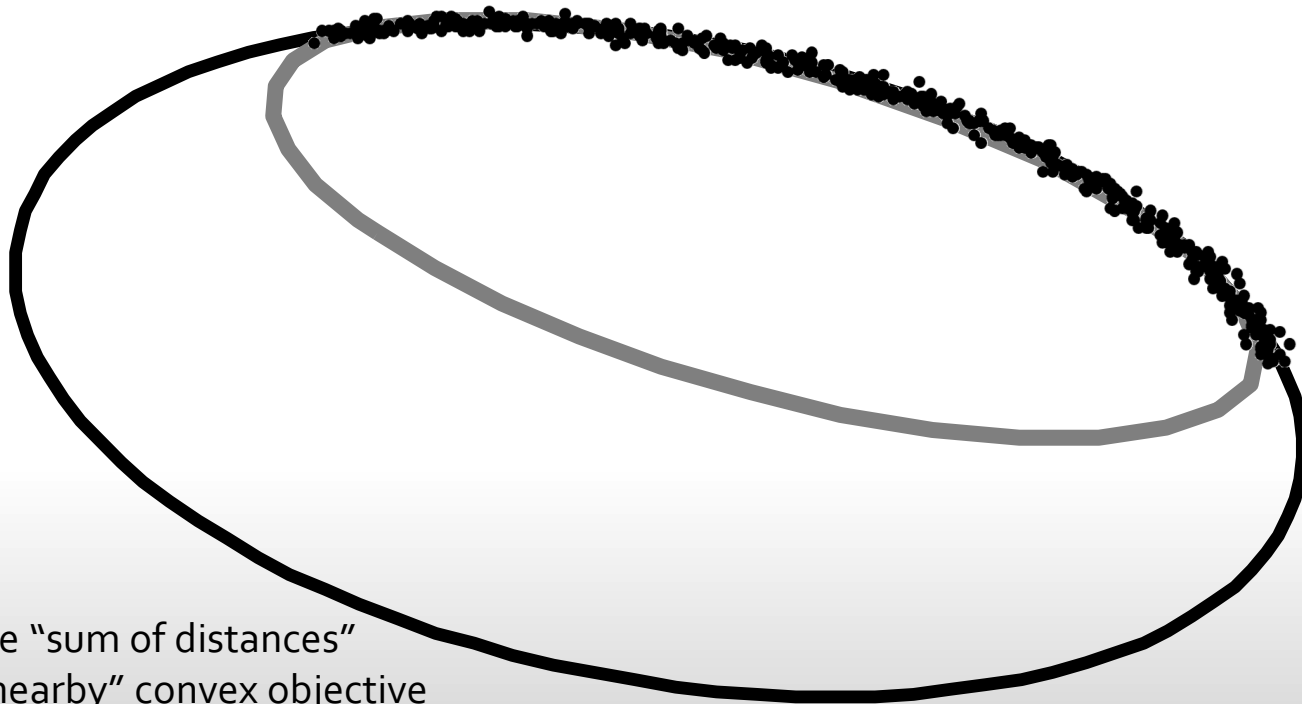
Clear(ish) objective:

Estimate the ellipse parameters, intersect with circle of sun, achieve fame



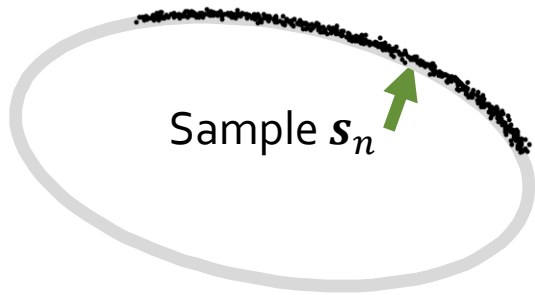
AND ESTIMATING IT WELL GETS US CLOSE...

“Direct least squares fitting of ellipses”  
[Fitzgibbon et al, 1999]



Does not minimize “sum of distances”  
objective, but a “nearby” convex objective

RUNNING AN OFF-THE-SHELF FITTER DOES NOT.



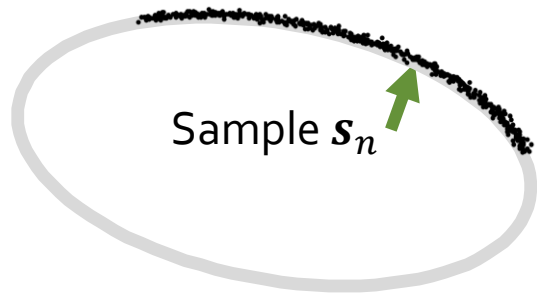
Measurements or “samples”:

- 2D points  $\mathbf{s}_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$  for  $n = 1..N$
- Captured at unknown times  $t_n$

Known model: Points lie on an ellipse

$$\mathbf{s}_n = \mathbf{c}(t_n; \boldsymbol{\theta}) + \text{Noise}$$

$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$



$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\mathbf{s}_n = \mathbf{c}(t_n; \boldsymbol{\theta}) + \text{Noise}$$

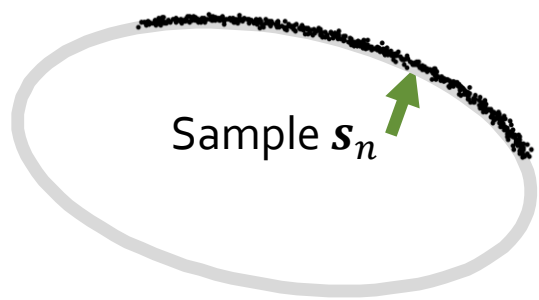
A parametric description

$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

Defines a **curve** (a set of points in  $\mathbb{R}^2$ )

$$\mathcal{C}(\boldsymbol{\theta}) = \{\mathbf{c}(t; \boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

Potential confusion: curve parameter  $t$  and shape parameter vector  $\boldsymbol{\theta}$ . This should be ok for this talk.



Sample  $s_n$

$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\mathbf{s}_n = \mathbf{c}(t_n; \boldsymbol{\theta}) + \text{Noise}$$

$$C(\boldsymbol{\theta}) = \{\mathbf{c}(t; \boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

All our algorithms will start with a guess of  $\boldsymbol{\theta}$  and refine it.

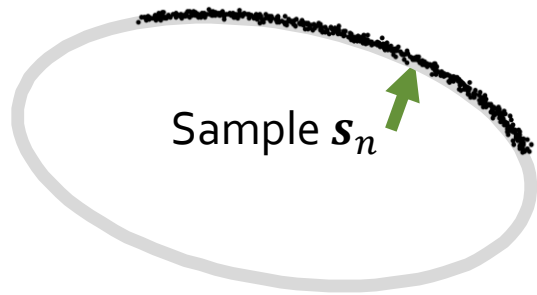
We will often want to think about the *distance* of a sample  $\mathbf{s}$  from the curve  $C(\boldsymbol{\theta})$ .

Often, *closest point* is appropriate.

[Others easily handled too.]

$$D(\mathbf{s}, \boldsymbol{\theta})^2 := \min_{\mathbf{x} \in C(\boldsymbol{\theta})} \|\mathbf{s} - \mathbf{x}\|^2$$

$$D(\mathbf{s}, \boldsymbol{\theta})^2 := \min_t \|\mathbf{s} - \mathbf{c}(t; \boldsymbol{\theta})\|^2$$



$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\mathbf{s}_n = \mathbf{c}(t_n; \boldsymbol{\theta}) + \text{Noise}$$

$$\mathcal{C}(\boldsymbol{\theta}) = \{\mathbf{c}(t; \boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

$$D(\mathbf{s}, \boldsymbol{\theta}) := \min_t \|\mathbf{s} - \mathbf{c}(t; \boldsymbol{\theta})\|^2$$

Minimize over all ellipses  $\boldsymbol{\theta}$

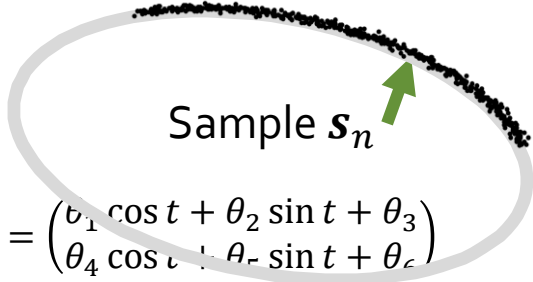
$$\boldsymbol{\theta}^* := \operatorname{argmin}_{\boldsymbol{\theta}} \sum_n D(\mathbf{s}_n, \boldsymbol{\theta})$$

Just using an off-the-shelf optimizer.

```
% Objective function for fminunc
% Distance of N data samples 'S' to
% curve 'theta'
function err = objective(theta, S)
    err = 0;
    for n=1:size(S,2)
        err = err + D(S(:,n), theta);
    end
end
```

```
% initial estimate 'theta_0'
theta_star = fminunc(@ (theta) objective(theta, S), theta_0);
```

```
% Sample from curve 'theta' at 't'
function out = c(t, theta)
    out = [
        theta(1)*cos(t) + theta(2)*sin(t) + theta(3)
        theta(4)*cos(t) + theta(5)*sin(t) + theta(6)
    ];
end
```



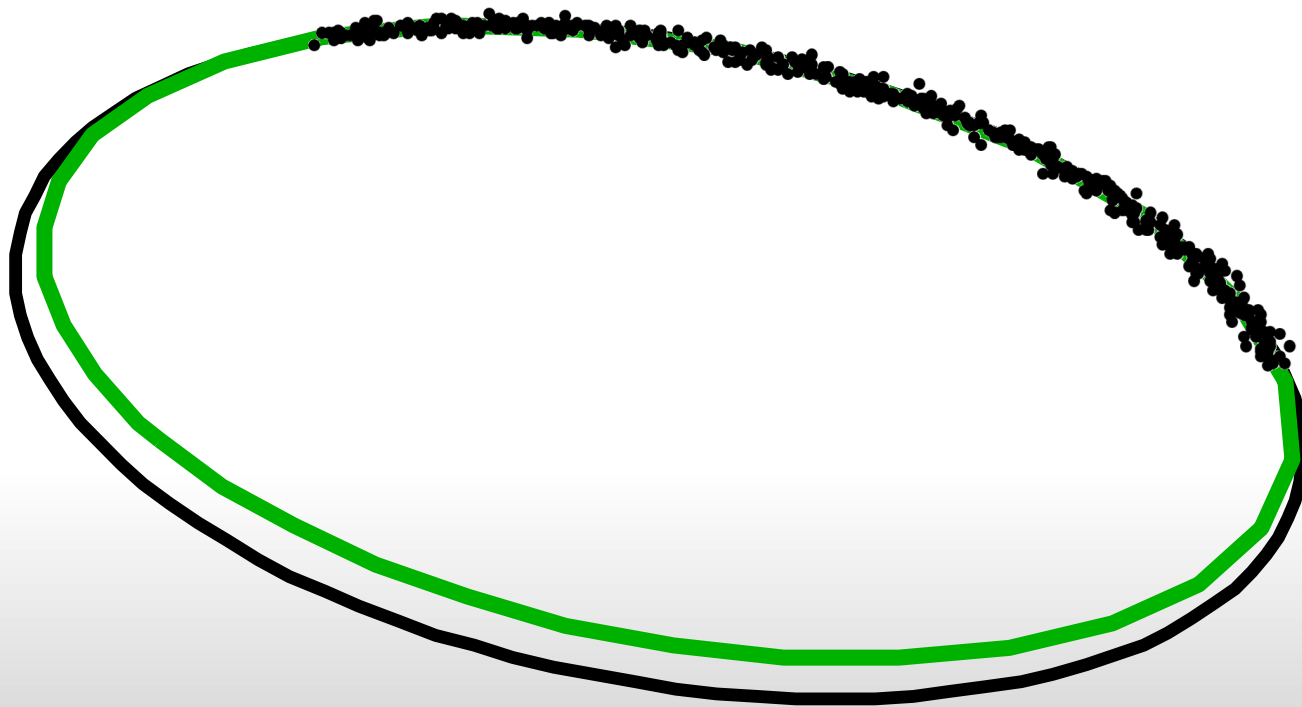
$$c(t; \theta) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

```
% Closest point to 's' on curve 'theta'
% Algorithm: discretize t and search.
function d_min = D(s, theta)
    d_min = Inf;
    for t_test = 0:0.01:2*pi
        d = norm(c(t_test, theta) - s);
        d_min = min(d, d_min);
    end
end
```

```
% Objective function for fminunc
% Distance of N data samples 'S' to
% curve 'theta'
function err = objective(theta, S)
    err = 0;
    for n=1:size(S,2)
        err = err + D(S(:,n), theta);
    end
end
```

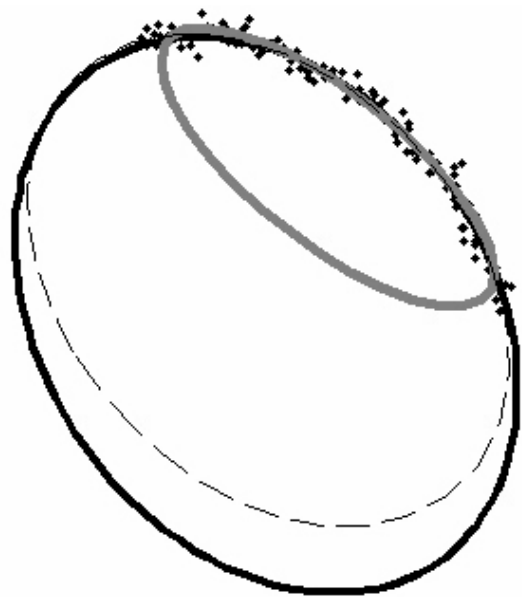
```
% initial estimate 'theta_0'
theta_star = fminunc(@objective(theta, S), theta_0);
```



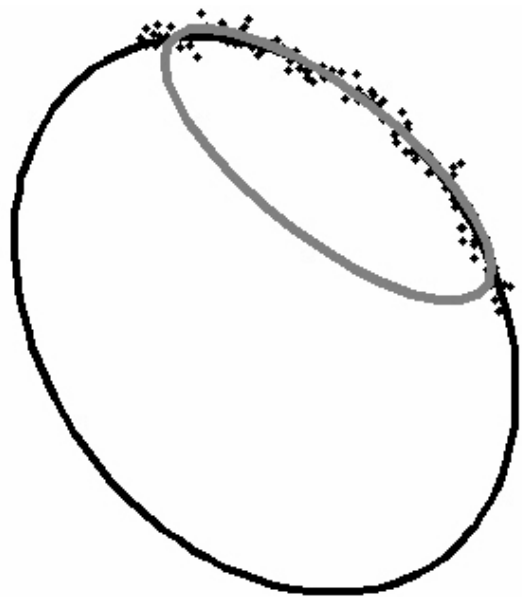


AND ESTIMATING IT WELL GETS US CLOSE...

- We have an accurate solution
  - Certainly better than the “closed form” algorithm, which minimized a “nearby” convex objective.
- All we need to worry about now is speed...
  - If you take 3 weeks to make a prediction, someone else will get the fame.
  - Speed *is* everything. If speed didn't matter, you would just use random search.
- Strategies to speed it up
  - Attack the inner loop
    - Remove discrete minimization in  $D(\mathbf{s}, \boldsymbol{\theta})$
  - Analyse the problem again
  - Understand our tools: 'fminunc', or whatever we're using
  - Compute analytic derivatives



A slow method



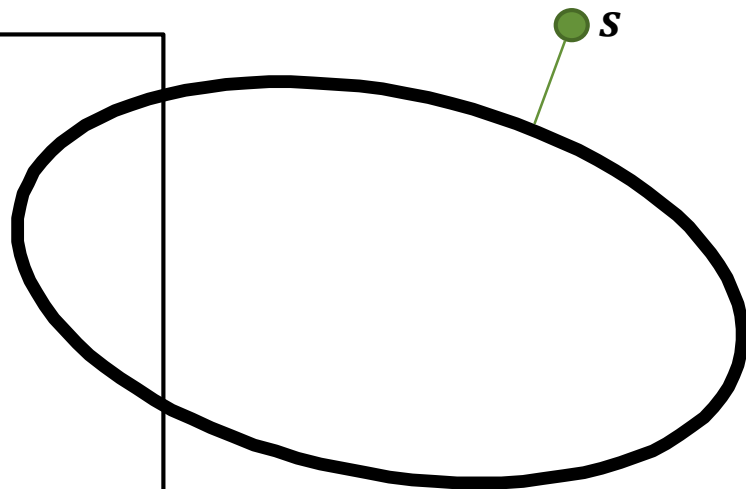
A fast method, slowed down 10x

# SPEEDUP <sub>1</sub>: ATTACK THE INNER LOOP

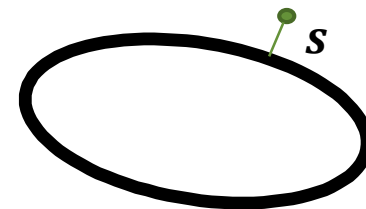
```
% Sample from curve 'theta' at 't'
function out = c(t, theta)
    out = [
        theta(1)*cos(t) + theta(2)*sin(t) + theta(3)
        theta(4)*cos(t) + theta(5)*sin(t) + theta(6)
    ]
end
```

```
% Closest point to 's' on curve 'theta'
% Algorithm: discretize t and search.
function d_min = D(s, theta)
    d_min = Inf;
    for t_test = 0:0.01:2*pi
        d = norm(c(t_test, theta) - s);
        d_min = min(d, d_min);
    end
end
```

```
theta_star = fminunc(@(theta) objective(theta, S), theta_0);
```



```
% Sample from curve 'theta' at 't'
function out = c(t, theta)
    out = [
        theta(1)*cos(t) + theta(2)*sin(t) + theta(3)
        theta(4)*cos(t) + theta(5)*sin(t) + theta(6)
    ];
end
```



$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

Define  $E(t) = \|\mathbf{s} - \mathbf{c}(t; \boldsymbol{\theta})\|^2$

Set  $\frac{dE}{dt} = 0$

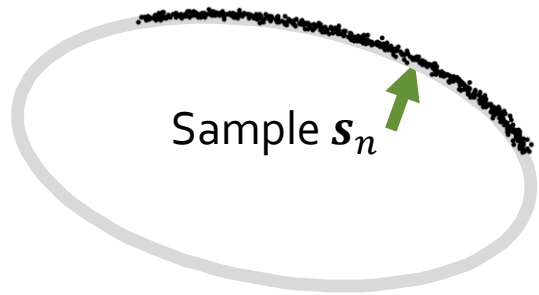
Yields 4<sup>th</sup> order polynomial, extract 4 roots.

Much cheaper than previous implementation.

```
% Closest point to 's' on curve 'theta'
% Algorithm: discretize t and search.
function d_min = D(s, theta)
    d_min = Inf;
    for t_test = 0:0.01:2*pi
        d = norm(c(t_test, theta) - s);
        d_min = min(d, d_min);
    end
end
```

$$D(\mathbf{s}, \boldsymbol{\theta}) = \min_t \|\mathbf{s} - \mathbf{c}(t; \boldsymbol{\theta})\|^2$$

# SPEEDUP 2: ANALYSE THE PROBLEM



$$\mathbf{c}(t; \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \cos t + \theta_2 \sin t + \theta_3 \\ \theta_4 \cos t + \theta_5 \sin t + \theta_6 \end{pmatrix}$$

$$\mathbf{s}_n = \mathbf{c}(t_n; \boldsymbol{\theta}) + \text{Noise}$$

$$\mathcal{C}(\boldsymbol{\theta}) = \{\mathbf{c}(t; \boldsymbol{\theta}) \mid 0 < t \leq 2\pi\}$$

$$D(\mathbf{s}, \boldsymbol{\theta}) := \min_t \|\mathbf{s} - \mathbf{c}(t; \boldsymbol{\theta})\|^2$$

$$\boldsymbol{\theta}^* := \operatorname{argmin}_{\boldsymbol{\theta}} \sum_n D(\mathbf{s}_n, \boldsymbol{\theta})$$

Minimize over all ellipses  $\boldsymbol{\theta}$

$$\sum_n D(\mathbf{s}_n, \boldsymbol{\theta}) = \sum_n \min_t \|\mathbf{s}_n - \mathbf{c}(t; \boldsymbol{\theta})\|^2$$

Notice  $\mathbf{c}(t; \boldsymbol{\theta})$  is linear in  $\boldsymbol{\theta}$ , so function is

$$= \sum_n \min_{t_n} \|\mathbf{s}_n - A(t_n)\boldsymbol{\theta}\|^2$$

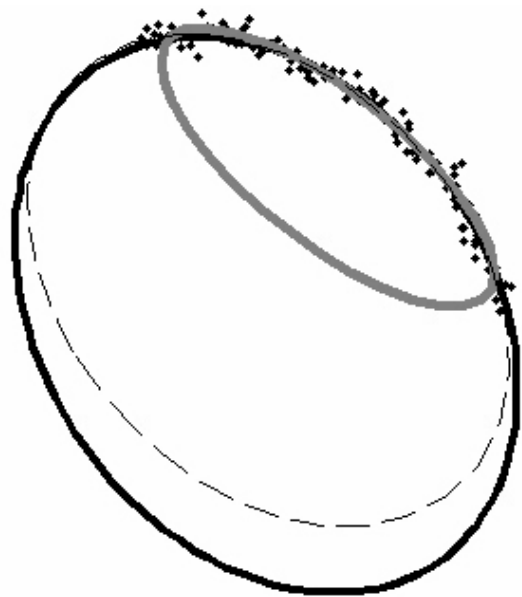
And we can solve in closed form:

- for  $T = \{t_n\}_{n=1}^N$  given  $\boldsymbol{\theta}$ . Cost  $N$  RootOfs.
- and  $\boldsymbol{\theta}$  given  $T$ . Cost one linear solve.

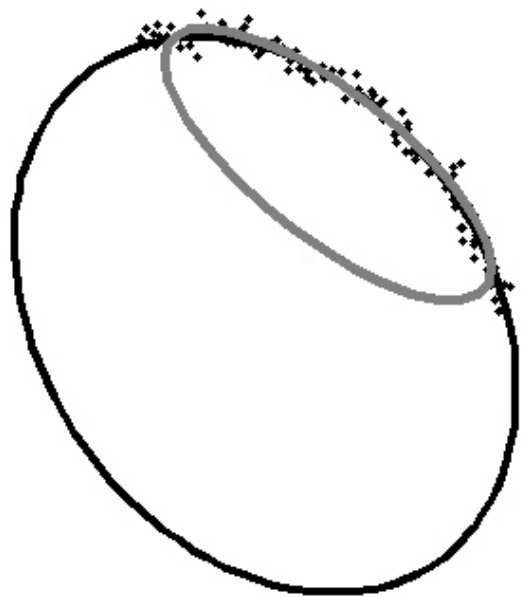
So alternate—"ICP", "EM", "Block Coordinate Descent"



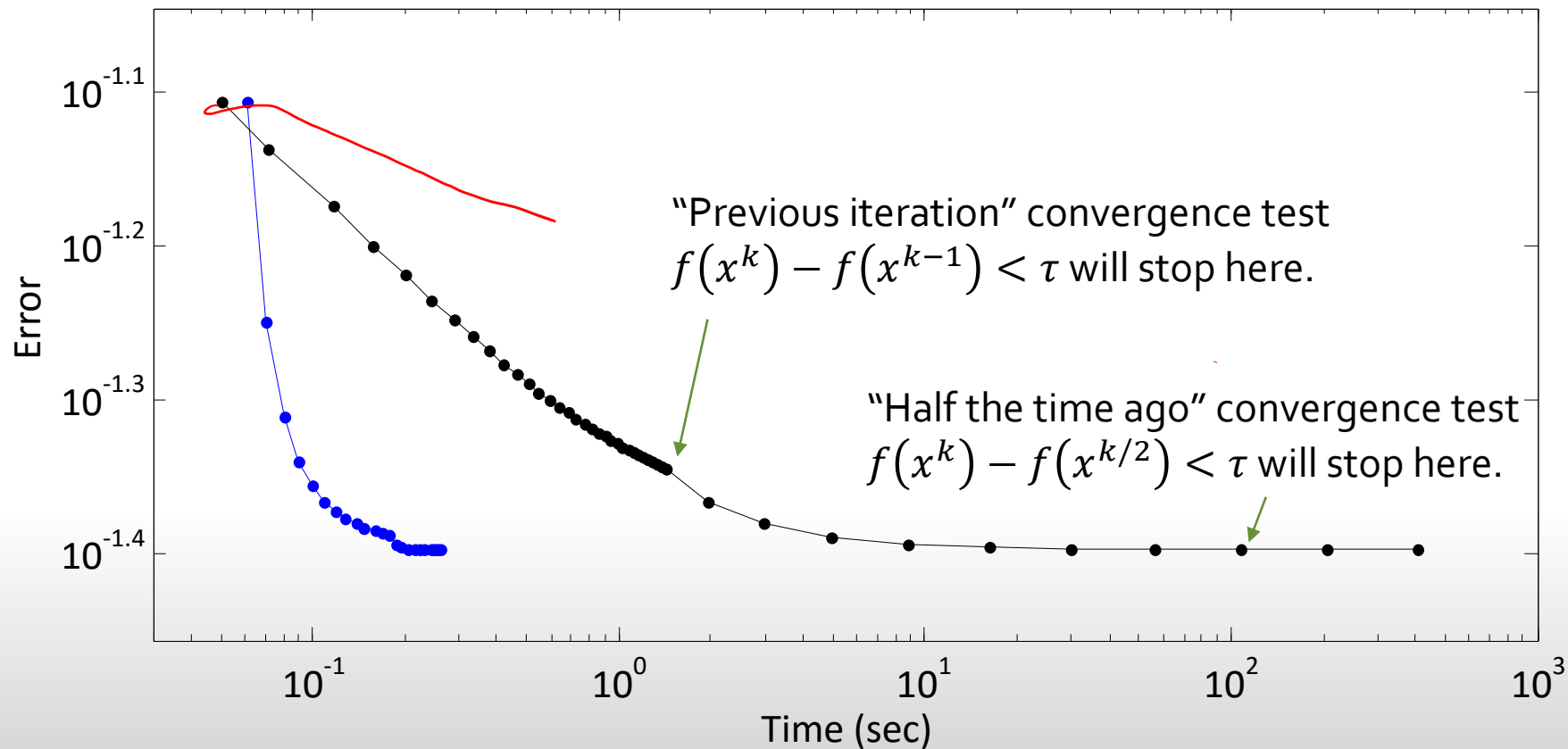
bad decision...

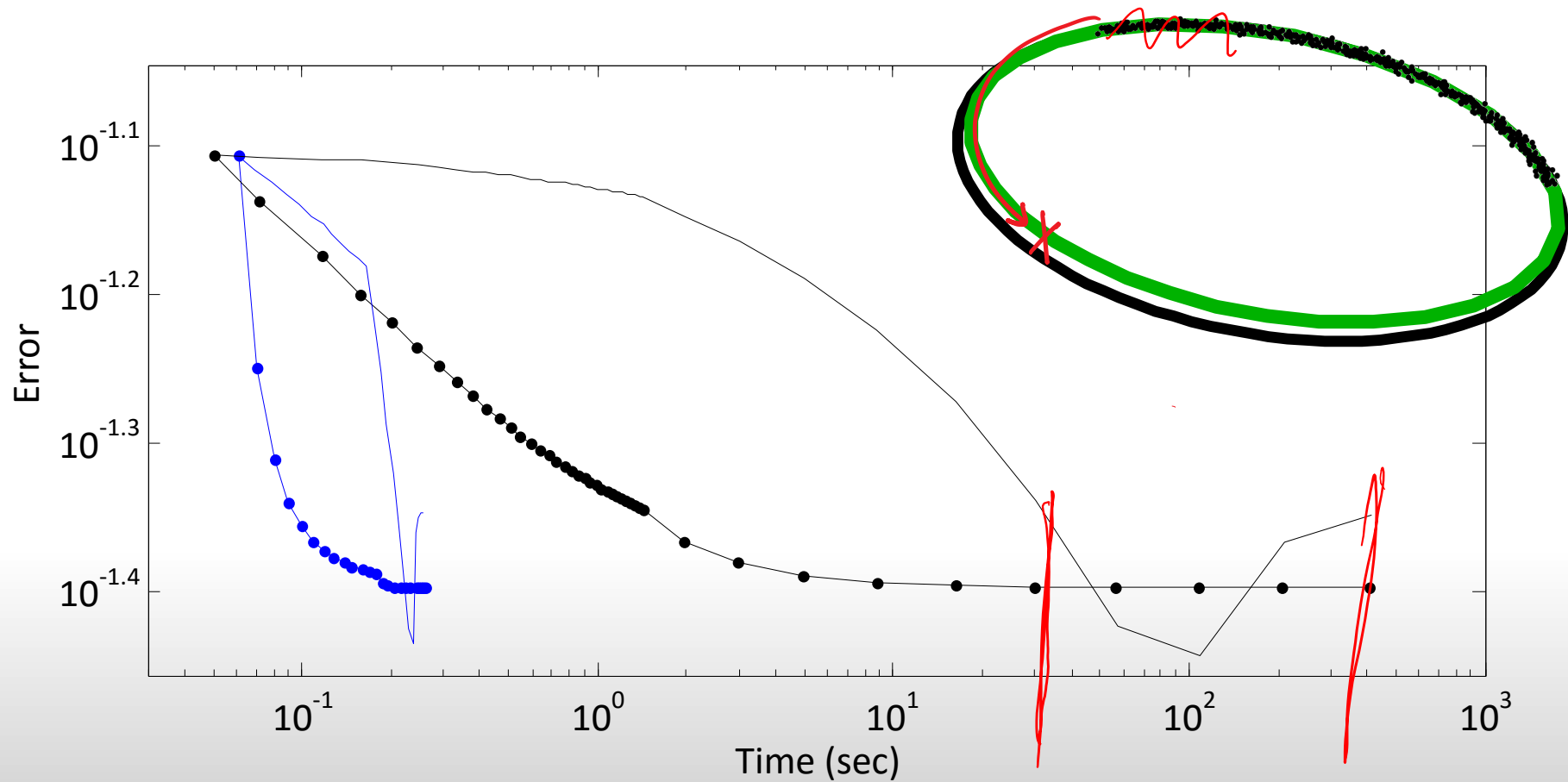


ICP, a bad 1<sup>st</sup>-order method



A second order method, slowed down 10x





AH, BUT WHAT ABOUT TEST ERROR?

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{n=1}^N \min_u f_n(u, \theta)$$

$$\begin{aligned}\hat{\theta} &= \operatorname{argmin}_{\theta} \sum_{n=1}^N \min_u f_n(u, \theta) \\ &= \operatorname{argmin}_{\theta} \sum_n \min_{u_n} f_n(u_n, \theta)\end{aligned}$$

$$\min_{u_i} f(x) + g(y)$$

$$\begin{aligned}\hat{\theta} &= \operatorname{argmin}_{\theta} \sum_{n=1}^N \min_t f_n(u, \theta) \\ &= \operatorname{argmin}_{\theta} \sum_n \min_{u_n} f_n(u_n, \theta) \\ &= \operatorname{argmin}_{\theta} \min_{u_{1..N}} \sum_n f_n(u_n, \theta)\end{aligned}$$

$$= \min_{u_1, u_2} f_1(u_1, x) + f_2(u_2, x)$$

[Recall that:  $\min_x (f(x) + \min_y g(y)) = \min_{x,y} f(x) + g(y)$ ]

$$\hat{\theta} = \operatorname{argmin}_{\theta} \sum_{n=1}^N \min_u f_n(u, \theta)$$

- Nasty objective
- $M$  parameters
- Cost per iteration  $O(N)$

*Slow*

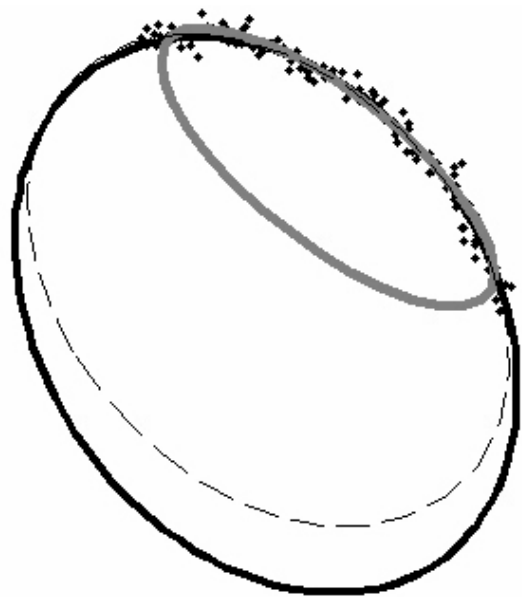
$$\hat{\theta} = \operatorname{argmin}_{\theta} \min_{u_{1..N}} \sum_n f_n(u_n, \theta)$$

- Simple objective (no “min”)
- $M + N$  parameters
- Cost per iteration  $O(NM^r)$

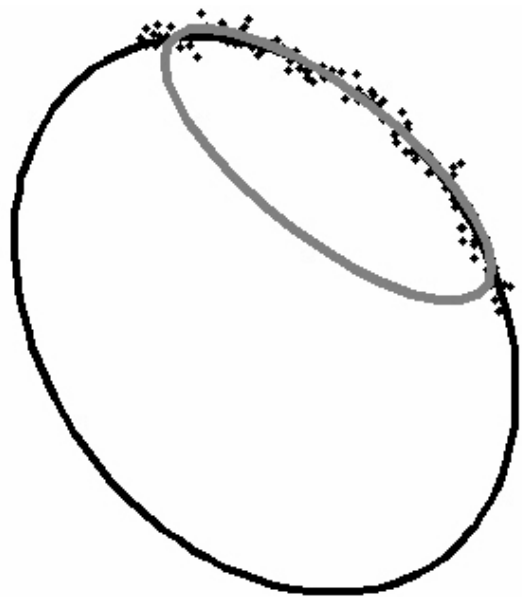
*Fast*

(in actual real-world wall clock time, even for very large  $N$ )





ICP, a bad 1<sup>st</sup>-order method



A second order method, slowed down 10x

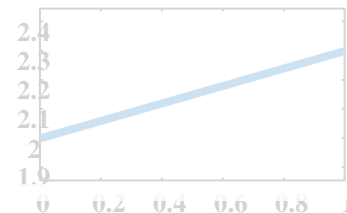
$$\begin{bmatrix} A_t & A_r \end{bmatrix}$$

Sparse  $Q_N(J)$

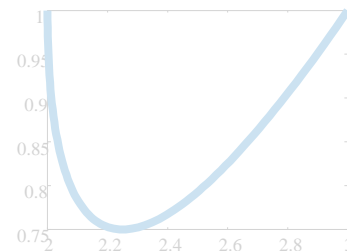
Block  $Q_N \left( \text{Block Diag } Q_N(2,1), \text{Dense } Q_N(J) \right)$

WHAT IS A SURFACE?

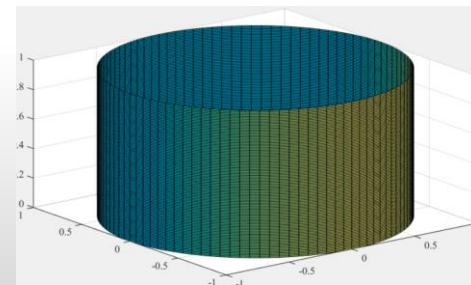
```
function y(x::Interval)::Real = .3*x + 2
```



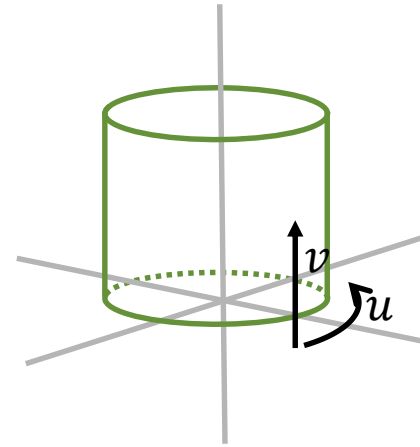
```
function C(t::Interval)::Point2D =  
    Point2D(t^2 + 2, t^2 - t + 1)
```



```
function S(u::Interval, v::Real)::Point3D =  
    Point3D(cos(u), sin(u), v)
```

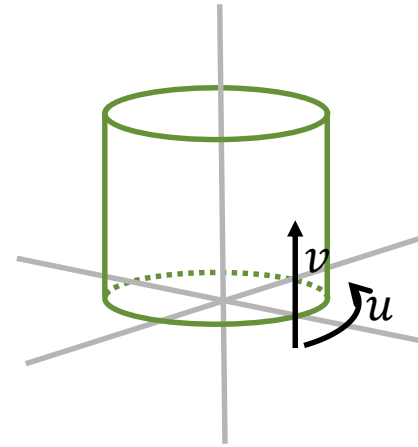


- Surface: mapping  $S(\mathbf{u})$  from  $\mathbb{R}^2 \mapsto \mathbb{R}^3$ 
  - E.g. cylinder  $S(u, v) = (\cos u, \sin u, v)$



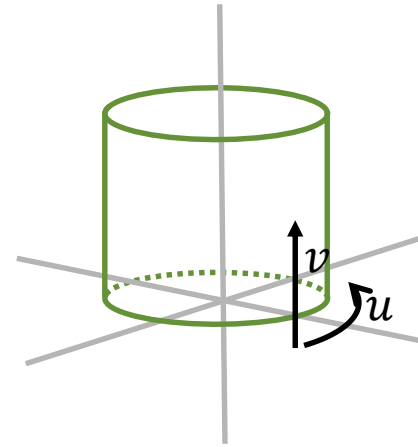
\*the surface is actually the set  $\{M(u; \theta) | u \in \Omega\}$

- Surface: mapping  $S(\mathbf{u})$  from  $\mathbb{R}^2 \mapsto \mathbb{R}^3$ 
  - E.g. cylinder  $S(u, v) = (\cos u, \sin u, v)$
- Probably not all of  $\mathbb{R}^2$ , but a subset  $\Omega$ 
  - E.g. square  $\Omega = [0, 2\pi) \times [0, H]$
  - But also any union of **patch domains**  $\Omega = \bigcup_p \Omega_p$



\*the surface is actually the set  $\{M(u; \theta) | u \in \Omega\}$

- Surface: mapping  $S(\mathbf{u})$  from  $\mathbb{R}^2 \mapsto \mathbb{R}^3$ 
  - E.g. cylinder  $S(u, v) = (\cos u, \sin u, v)$
- Probably not all of  $\mathbb{R}^2$ , but a subset  $\Omega$ 
  - E.g. square  $\Omega = [0, 2\pi) \times [0, H]$
  - But also any union of **patch domains**  $\Omega = \bigcup_p \Omega_p$
- And we'll look at **parameterised** surfaces  $S(\mathbf{u}; \Theta)$ 
  - E.g. Cylinder  $S(u, v; R, H) = (R \cos u, R \sin u, Hv)$  with  $\Omega = [0, 2\pi) \times [0, 1]$
  - E.g. subdivision surface  $S(\mathbf{u}; X)$  where  $\Theta = X \in \mathbb{R}^{3 \times n}$  is matrix of **control vertices**

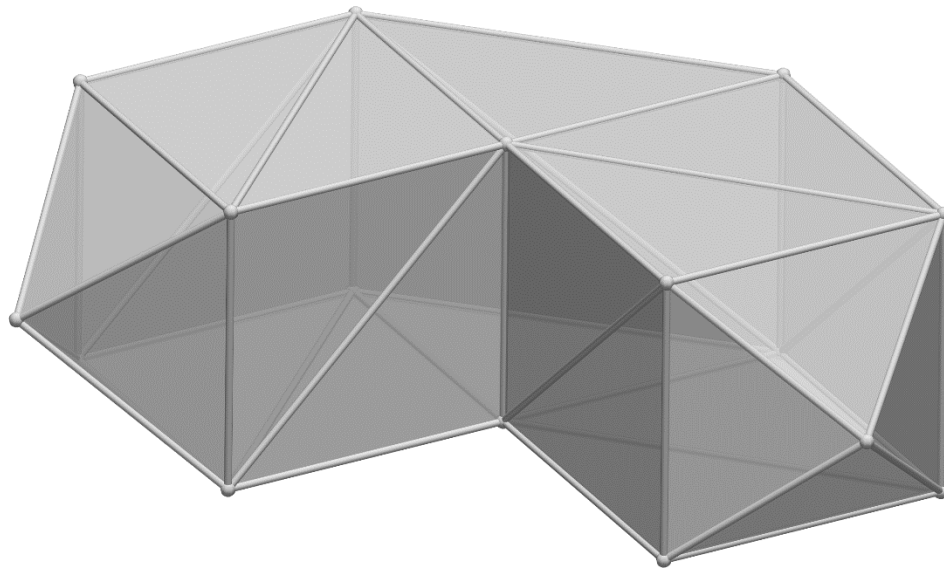


\*the surface is actually the set  $\{M(u; \Theta) | u \in \Omega\}$

TOOL: SUBDIVISION SURFACES

Control mesh vertices  $X \in \mathbb{R}^{3 \times m}$

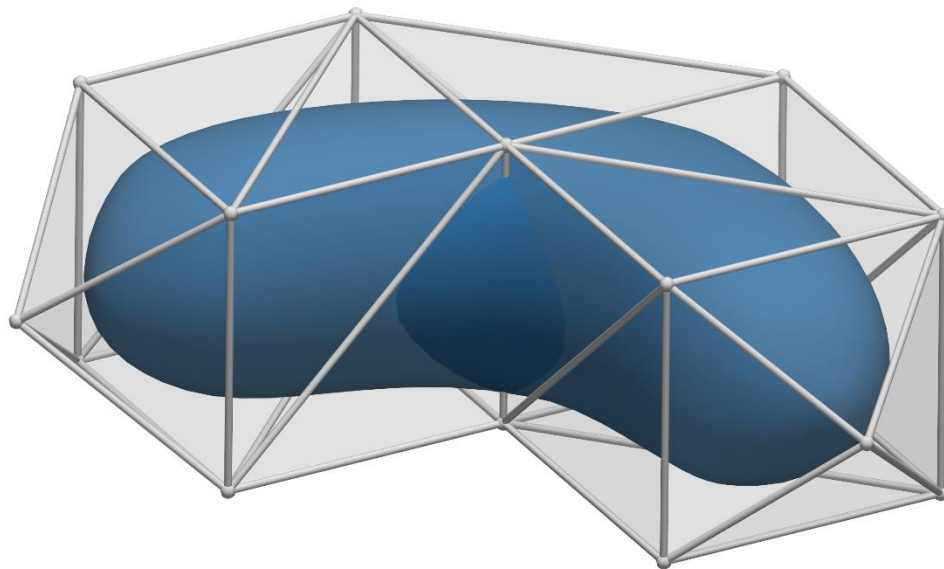
Here  $m = 16$

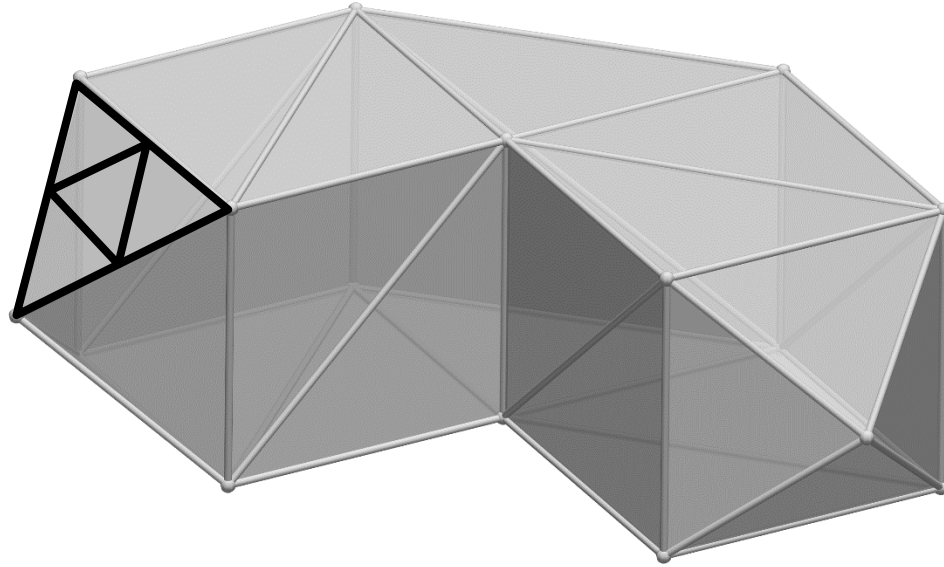




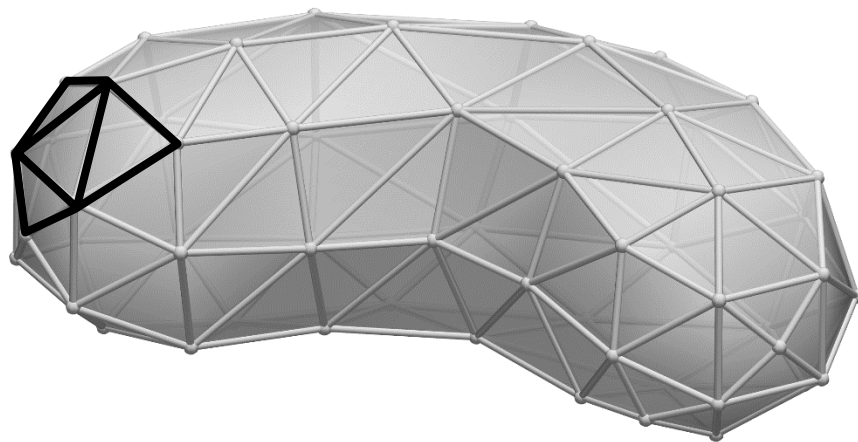
Control mesh vertices  $X \in \mathbb{R}^{3 \times m}$

Here  $m = 16$

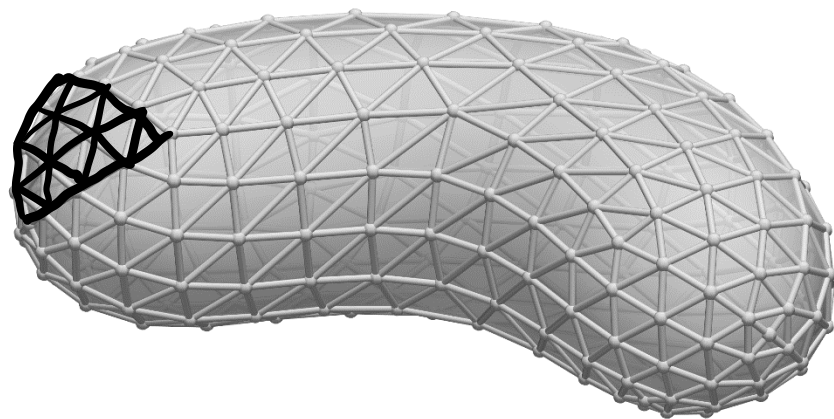


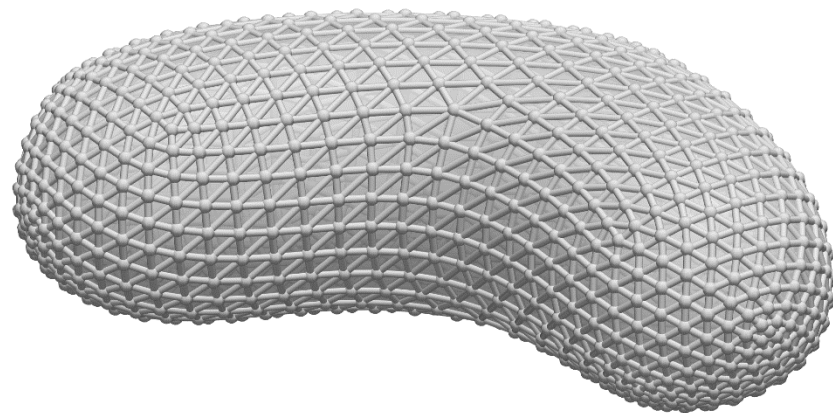


SUBDIV RULE: STEP 1. ADD NEW VERTICES



SUBDIV RULE: STEP 2. AVERAGE NEIGHBOURS



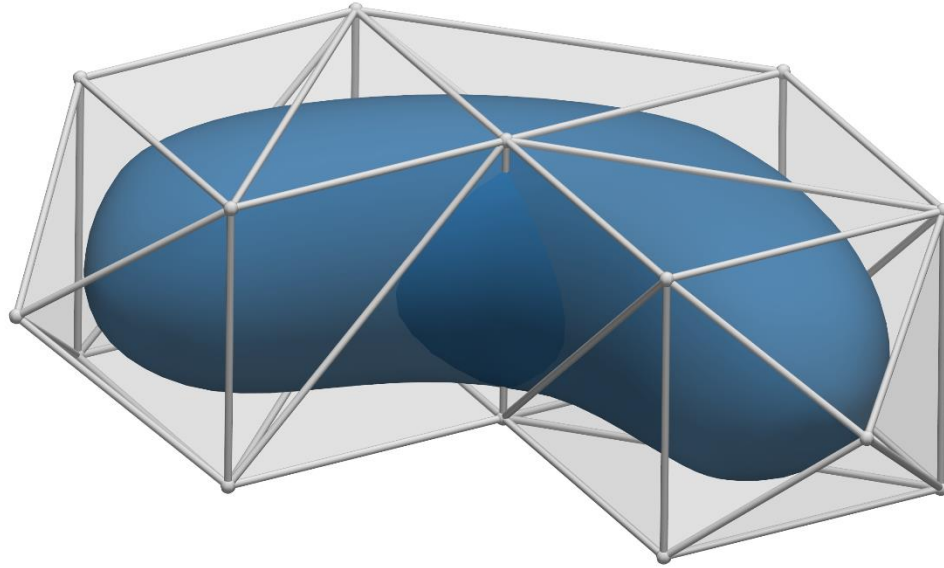


Control mesh vertices  $V \in \mathbb{R}^{3 \times m}$

Here  $m = 16$

Blue surface is  $\{M(\mathbf{u}; V) \mid \mathbf{u} \in \Omega\}$

$\Omega$  is the grey surface

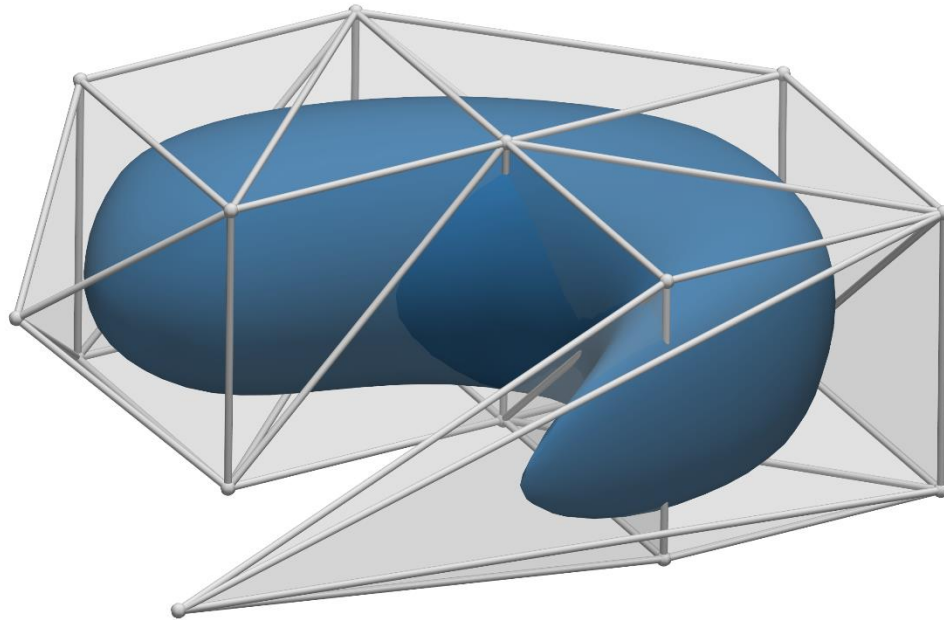


Control mesh vertices  $V \in \mathbb{R}^{3 \times n}$

Here  $n = 16$

Blue surface is  $\{M(\mathbf{u}; V) \mid \mathbf{u} \in \Omega\}$

$\Omega$  is the grey surface

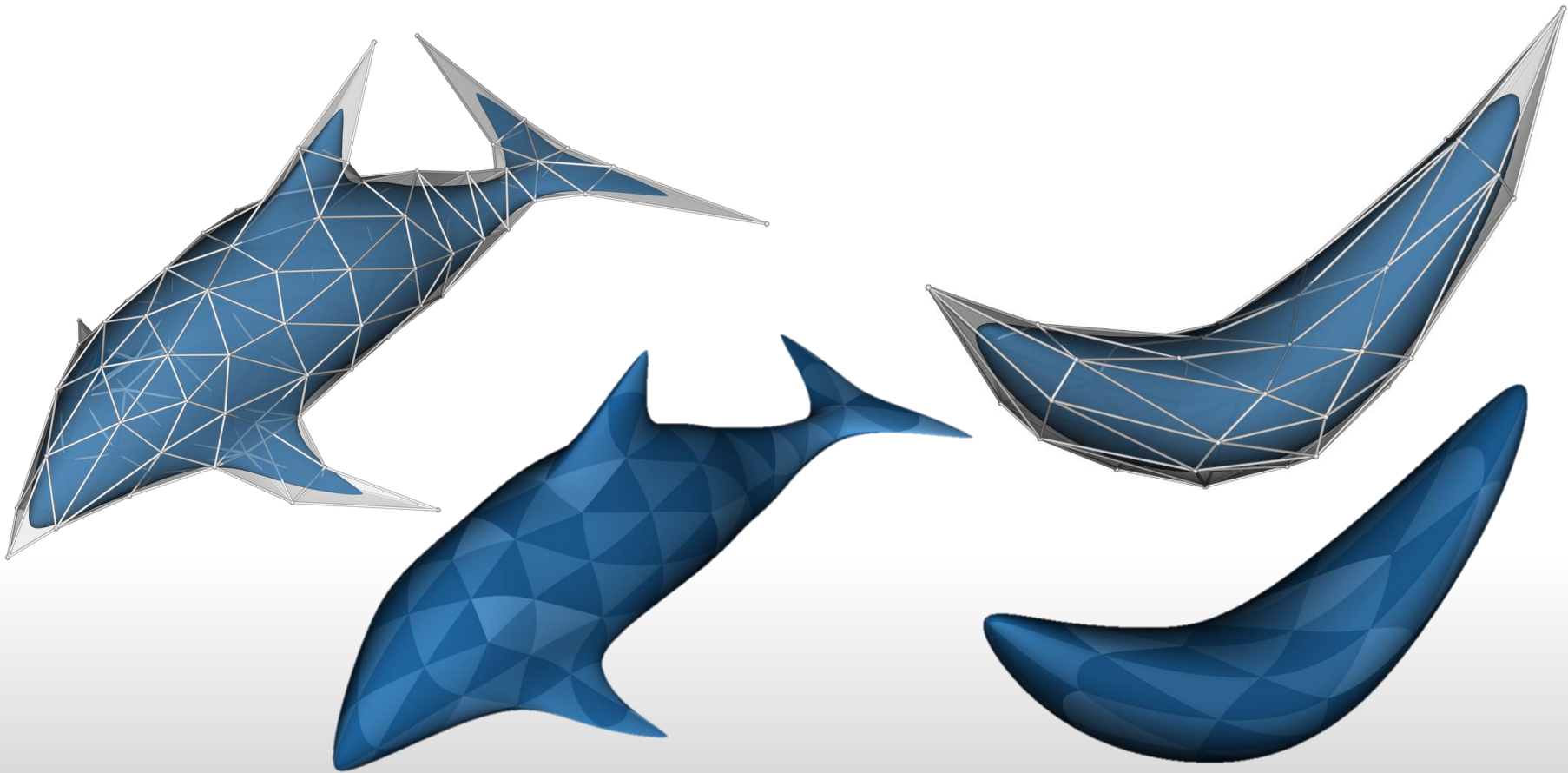


- Mostly,  $M$  is quite simple:

$$M(\mathbf{u}; X) = M(t, u, v; \mathbf{x}_1, \dots, \mathbf{x}_n) = \sum_{\substack{i+j \leq 4 \\ k=1..n}} A_{ijk}^t u^i v^j \mathbf{x}_k$$

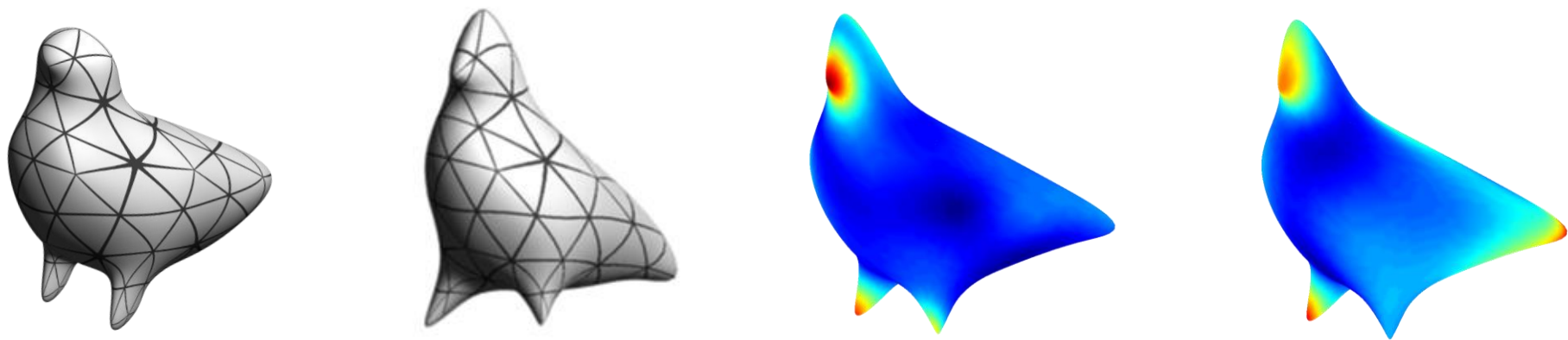
- Integer triangle id  $t$
- Quartic in  $u, v$
- Linear in  $X$
- Easy derivatives
- But...
  - 2<sup>nd</sup> Derivatives unbounded although normals well defined
  - Piecewise parameter domain





# EXAMPLES

# BACK TO DOLPHINS



$$X_i = \mathcal{B}_0 + \alpha_{i1} \mathcal{B}_1 + \alpha_{i2} \mathcal{B}_2$$

$$X_n = \sum_{k=0}^K \alpha_{ik} \mathcal{B}_k$$

Linear blend shapes:  
Image  $i$  represented by coefficient  
vector  $\alpha_i = [\alpha_{i1}, \dots, \alpha_{iK}]$



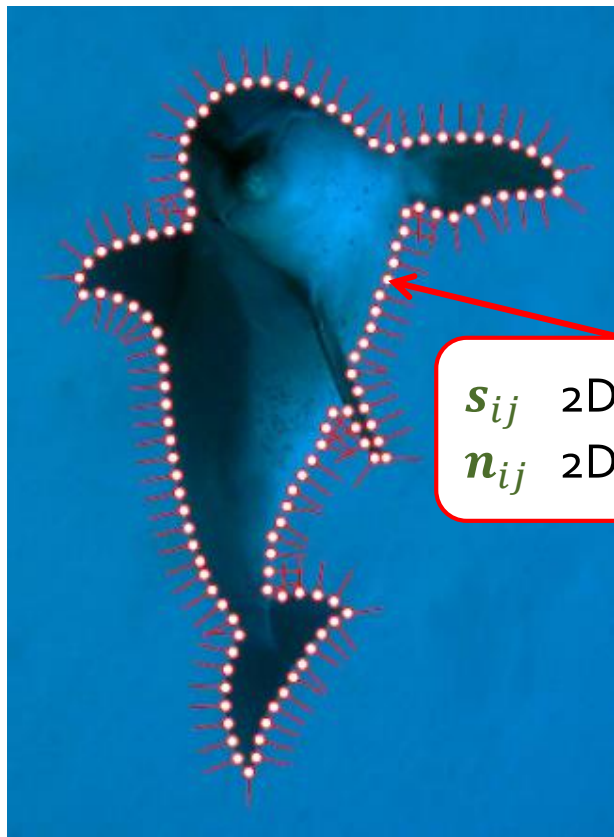




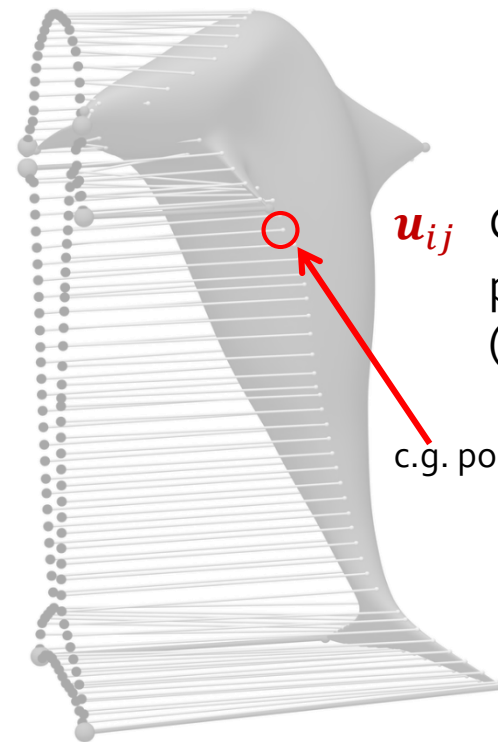




Image  $i$



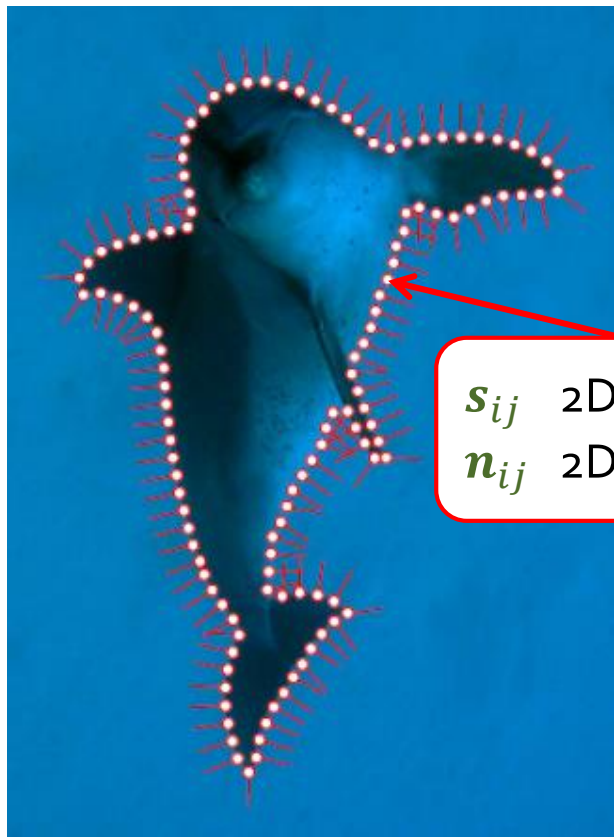
$s_{ij}$  2D point  
 $n_{ij}$  2D normal



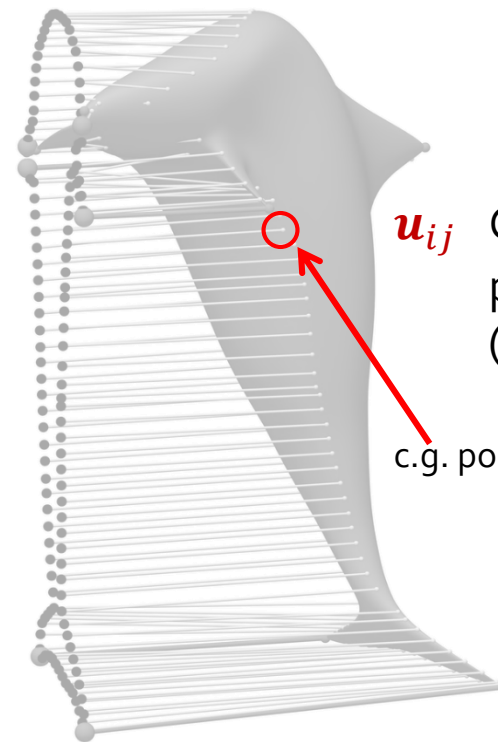
$u_{ij}$  Contour generator  
preimage in  $\Omega$   
(unknown)

c.g. point in 3D is  $M(u_{ij}; X_i)$

Image  $i$



$s_{ij}$  2D point  
 $n_{ij}$  2D normal

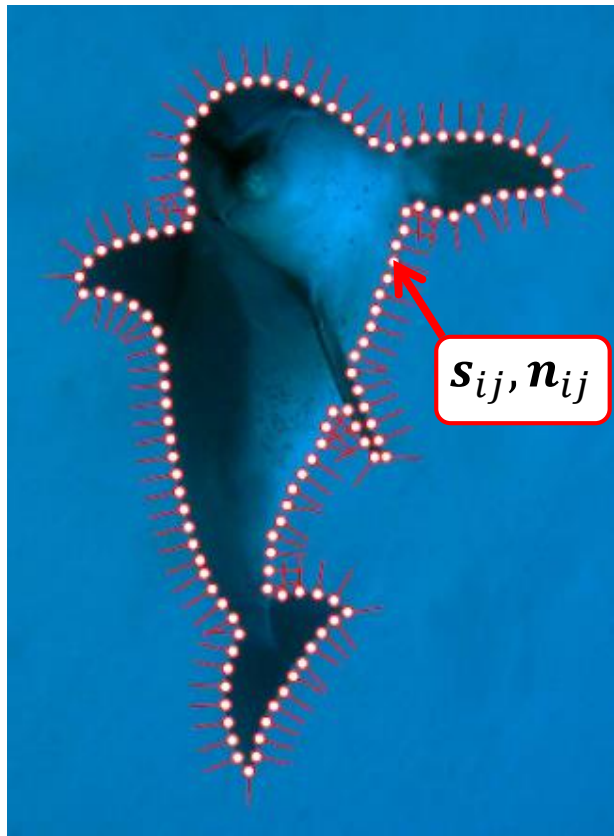


$u_{ij}$  Contour generator  
preimage in  $\Omega$   
(unknown)

c.g. point in 3D is  $M(u_{ij}; X_i)$



Image  $i$



$s_{ij}, n_{ij}$

Projection  
e.g. Perspective

Camera  
position

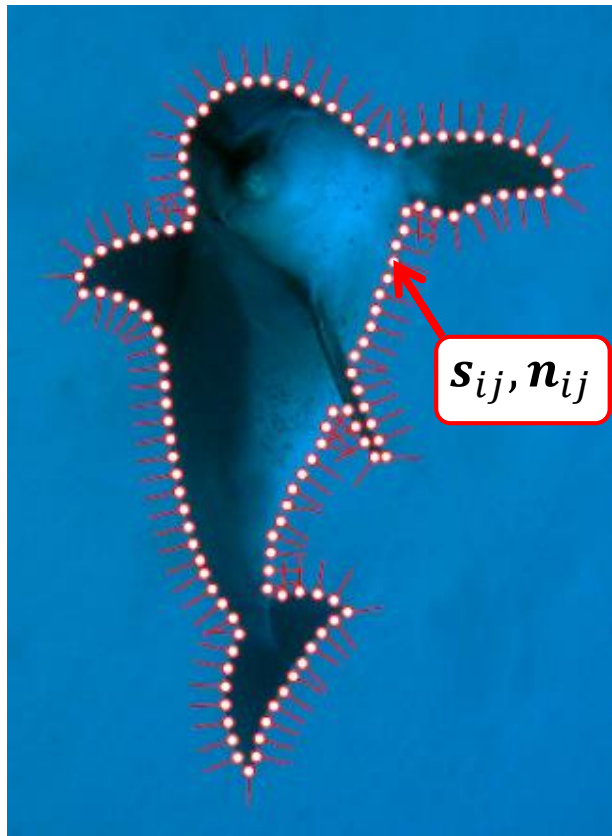
**Silhouette:**

$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| s_{ij} - \pi \left( \theta_i, M(u_{ij}, \mathbf{X}_i) \right) \right\|^2$$

**Normal:**

$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| \begin{bmatrix} n_{ij} \\ 0 \end{bmatrix} - R(\theta_i) N(u_{ij}, \mathbf{X}_i) \right\|^2$$

Image  $i$



## Linear Blend Shapes (PCA) Model:

$$\mathbf{X}_i = \sum_k \alpha_{ik} \mathbf{B}_k$$

## Silhouette:

$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| \mathbf{s}_{ij} - \pi \left( \theta_i, M(u_{ij}, \mathbf{X}_i) \right) \right\|^2$$

## Normal:

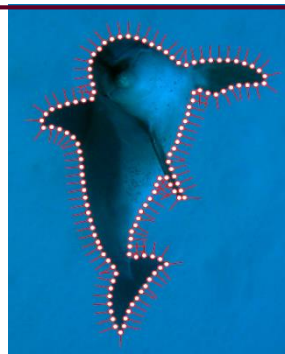
$$E_i^{sil} = \sum_{j=1}^{S_i} \left\| \begin{bmatrix} \mathbf{n}_{ij} \\ 0 \end{bmatrix} - R(\theta_i) N(u_{ij}, \mathbf{X}_i) \right\|^2$$

Data fidelity  
terms

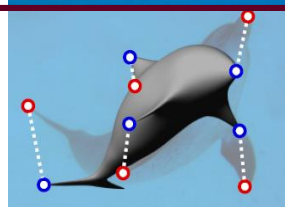
$p(I|X_i; U)$

$$E_i^{\text{sil}} = \frac{1}{2} \sigma_{\text{sil}}^{-2} \sum_{j=1}^{S_i} \|s_{ij} - \pi_i(M(\hat{u}_{ij}|X_i))\|^2$$

$$E_i^{\text{norm}} = \frac{1}{2} \sigma_{\text{norm}}^{-2} \sum_{j=1}^{S_i} \left\| \begin{bmatrix} n_{ij} \\ 0 \end{bmatrix} - \nu(R_i N(\hat{u}_{ij}|X_i)) \right\|^2$$



$$E_i^{\text{con}} = \frac{1}{2} \sigma_{\text{con}}^{-2} \sum_{k=1}^{K_i} \|c_{ik} - \pi_i(M(\hat{\mu}_{ik}|X_i))\|^2$$



Smooth Basis  
 $p(\Theta)$

$$E_m^{\text{tp}} = \frac{\bar{\lambda}^2}{2} \int_{\Omega} \|M_{xx}(\hat{u}|B_m)\|^2 + 2 \|M_{xy}(\hat{u}|B_m)\|^2 + \|M_{yy}(\hat{u}|B_m)\|^2 \, d\hat{u}$$

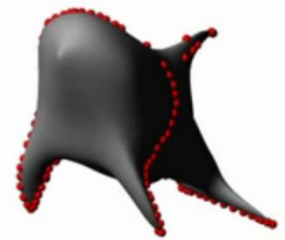
Gaussian shape  
weights

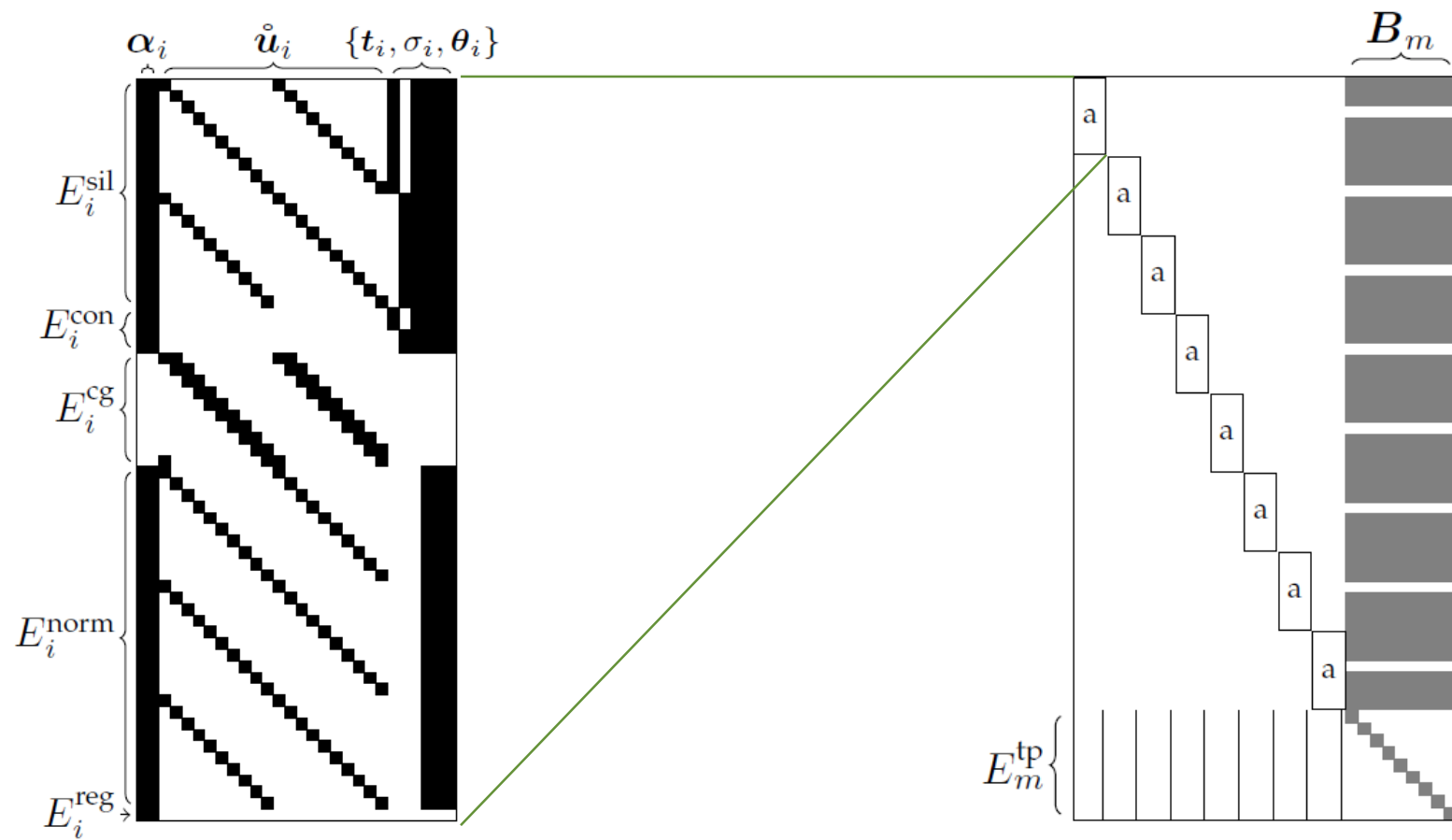
$$E_i^{\text{reg}} = \beta \sum_{m=1}^D \alpha_{im}^2$$

$$X_i = \sum_{m=0}^D \alpha_{im} B_m$$

Smooth  
contour

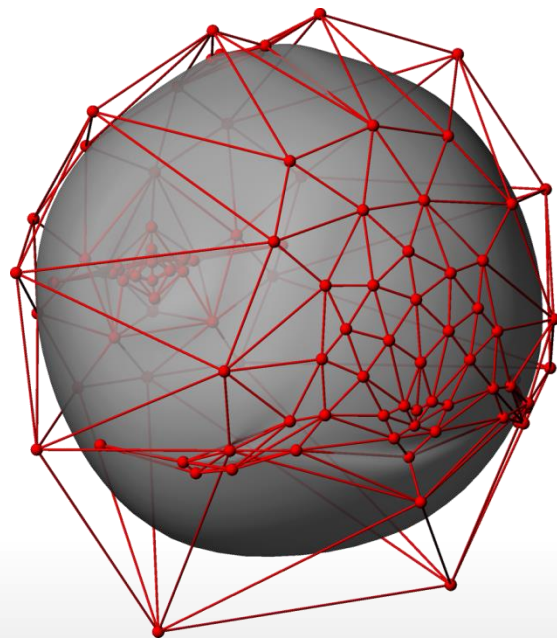
$$E_i^{\text{cg}} = \gamma \sum_{j=1}^{S_i} \tau(d(\hat{u}_{ij}, \hat{u}_{i,j+1}))$$



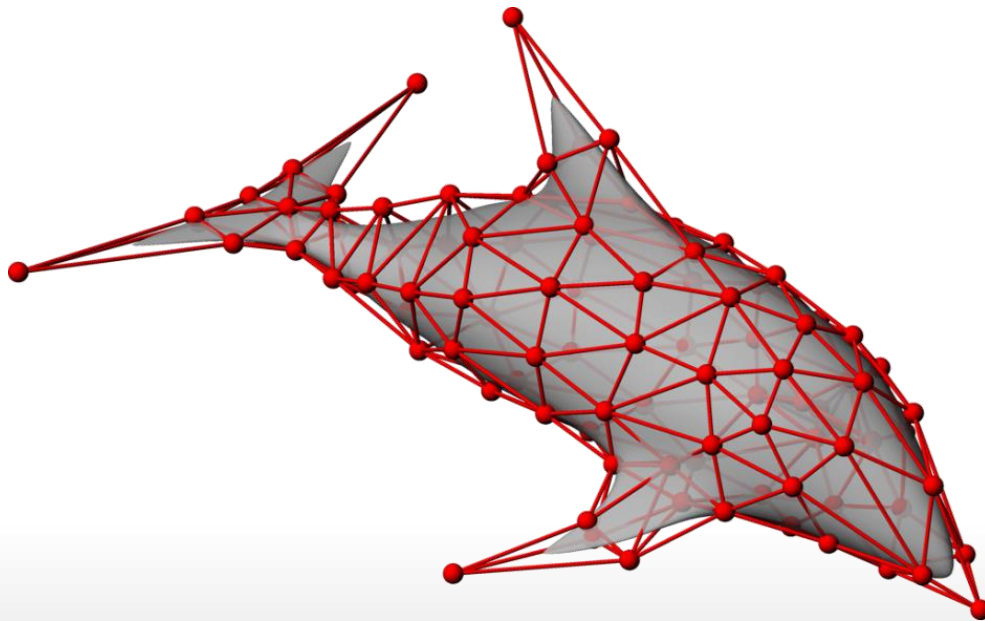


$$E_i^{\text{sil}} = \frac{1}{2} \sigma_{\text{sil}}^{-2} \sum_{j=1}^{S_i} \|s_{ij} - \pi_i(M(\overset{\text{red arc}}{u}_{ij} | X_i))\|^2 \quad \text{red } \sum \alpha_{ik} B_k$$

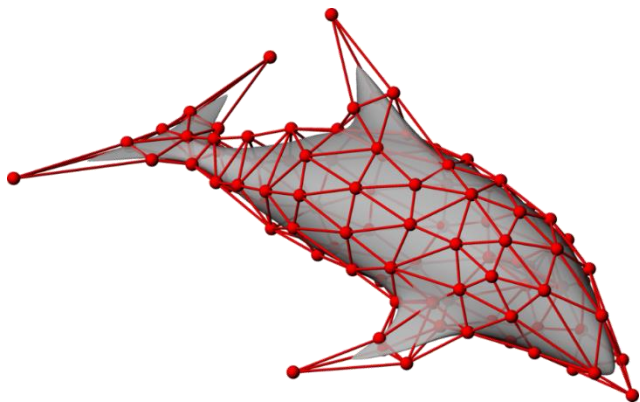
- Can focus on this term to understand entire optimization.
  - Total number of residuals  $n$  = number of silhouette points.  
Say  $300N$  ( $N$  = number of images)  $\approx 10,000$
  - Total number of unknowns  $2n + KN + m$  where  
 $m \approx 3K \times \text{number of vertices} \approx 3,000$



This is true, but misleading

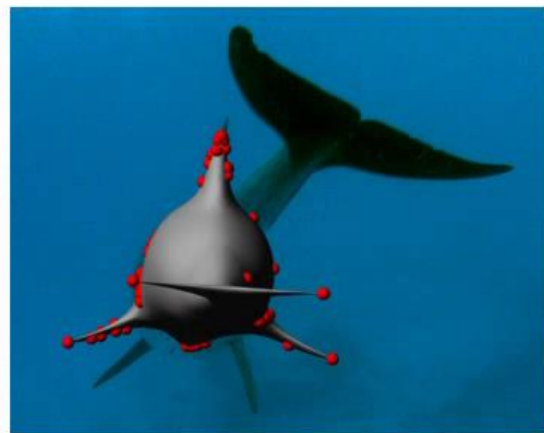
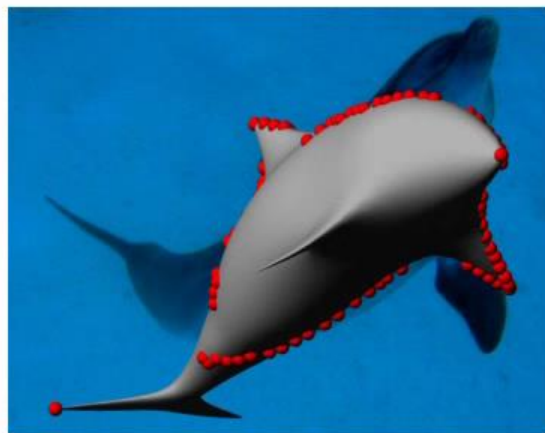
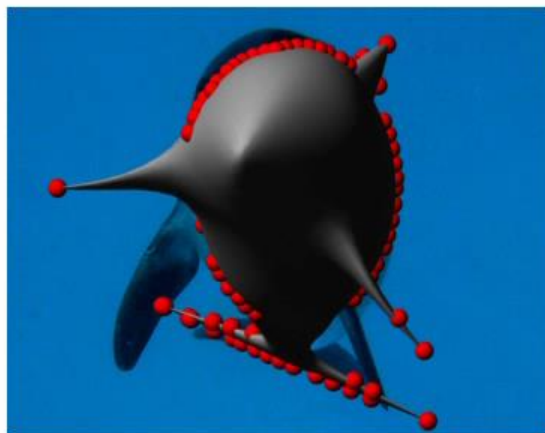
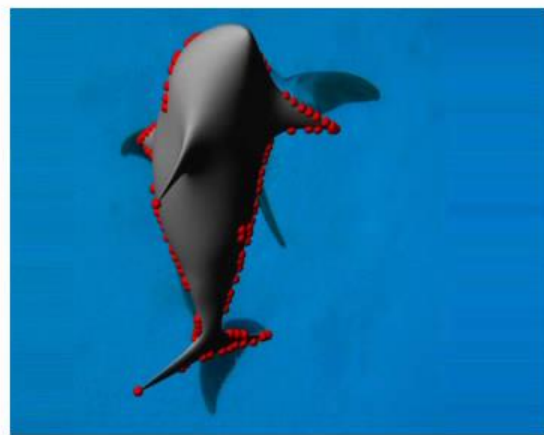
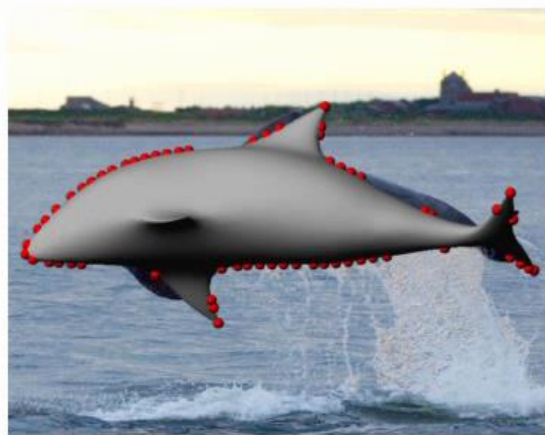


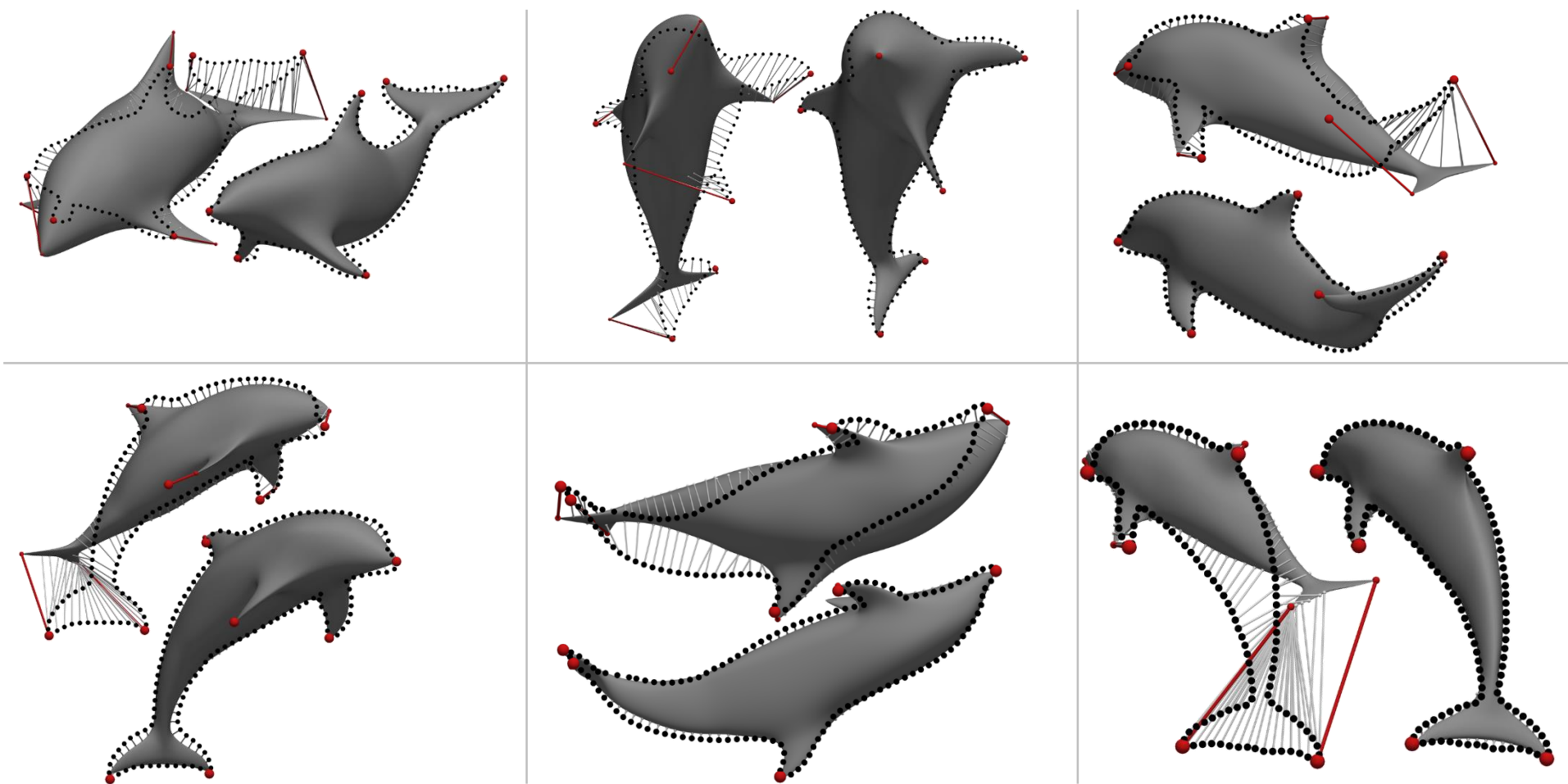
True initial estimate: only the *topology* is really important.  
But the easiest way to get the topology is to build a rough template.



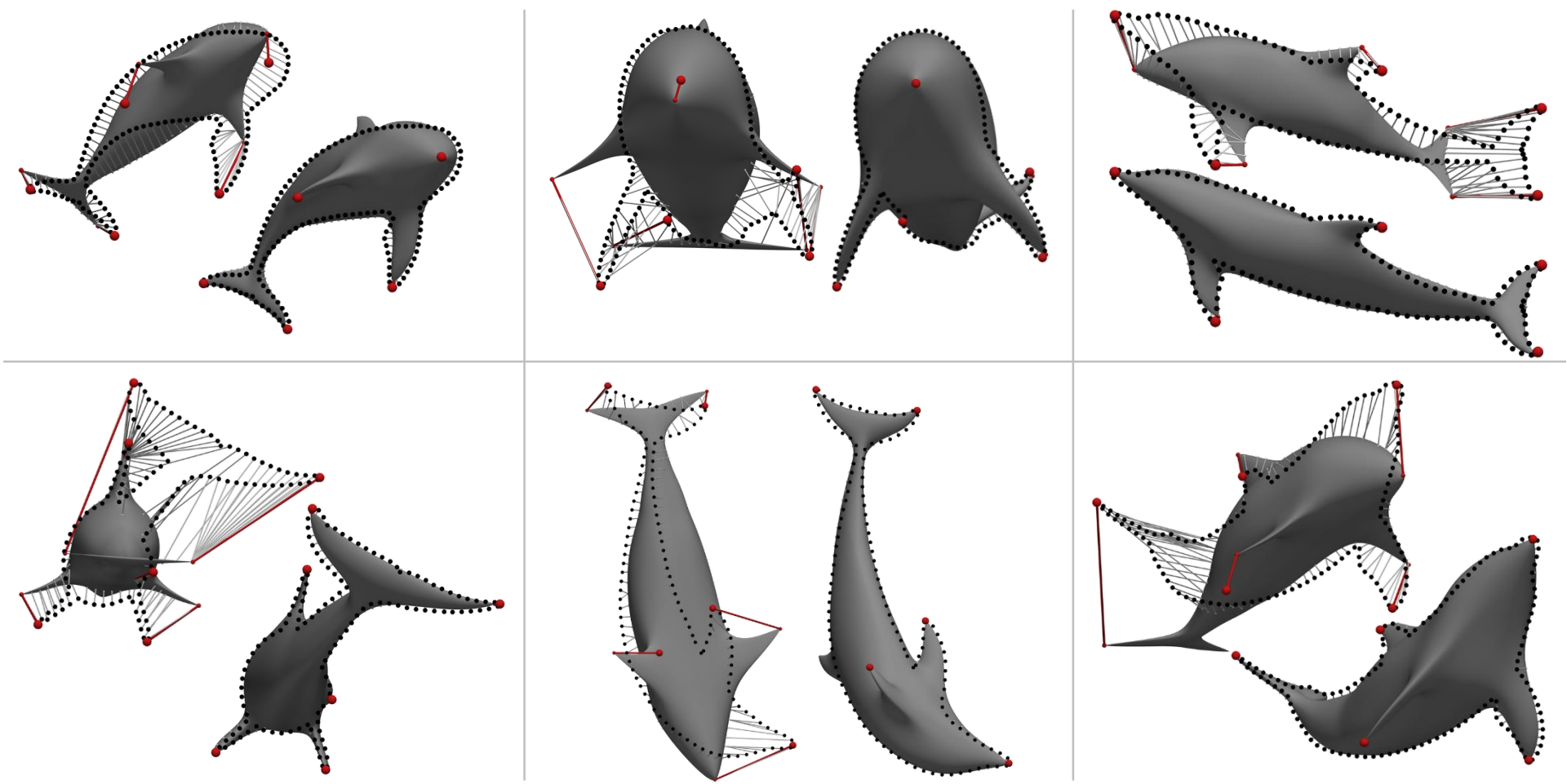
True initial estimate: only the *topology* is really important.  
But the easiest way to get the topology is to build a rough template.



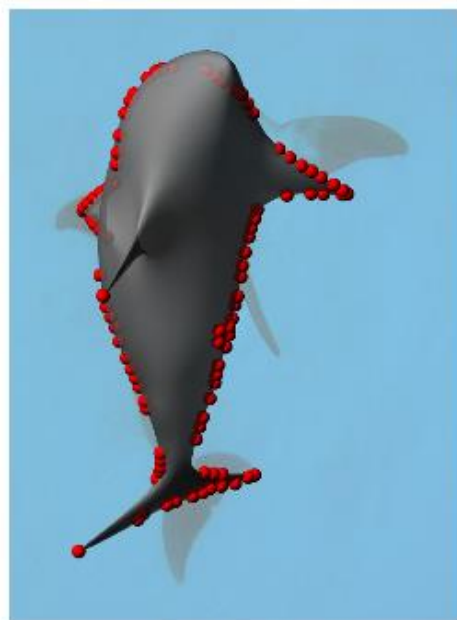




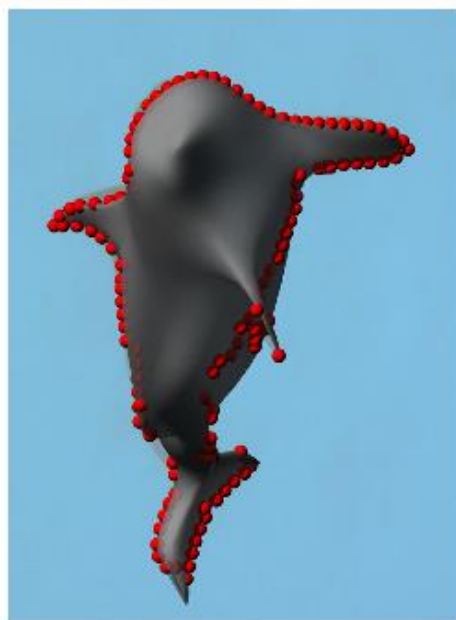
EXAMPLE RESULTS



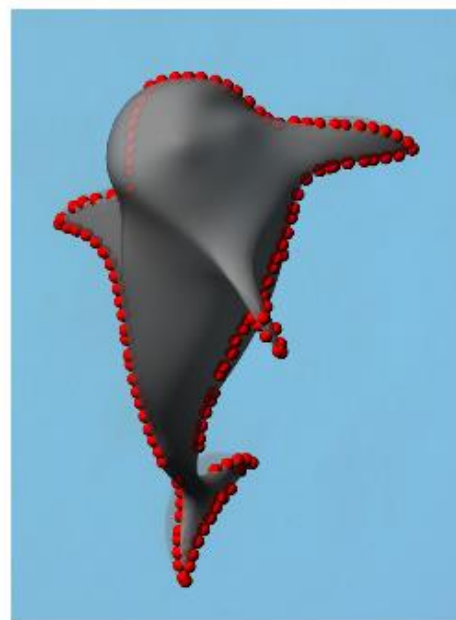
EXAMPLE RESULTS



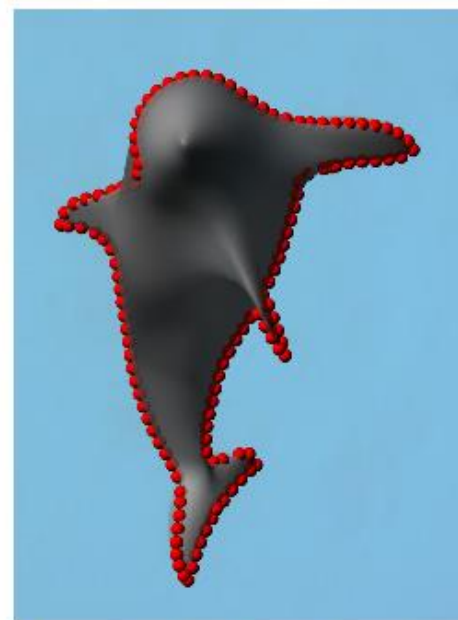
(a) Initial estimate.



(b) Only continuous local optimization, as described in Sec. 4.1.



(c) As (b), but including iterations of our global search (Sec. 4.2).



(d) As (c), but with reparametrization around extraordinary vertices.



8



16



32

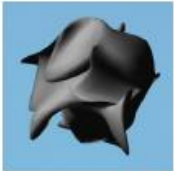
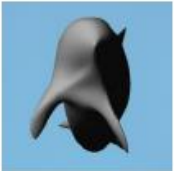







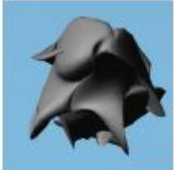
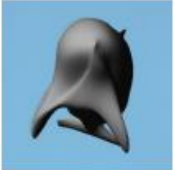







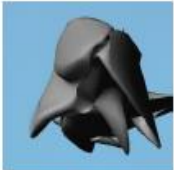
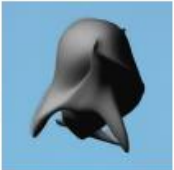
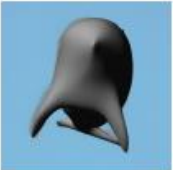








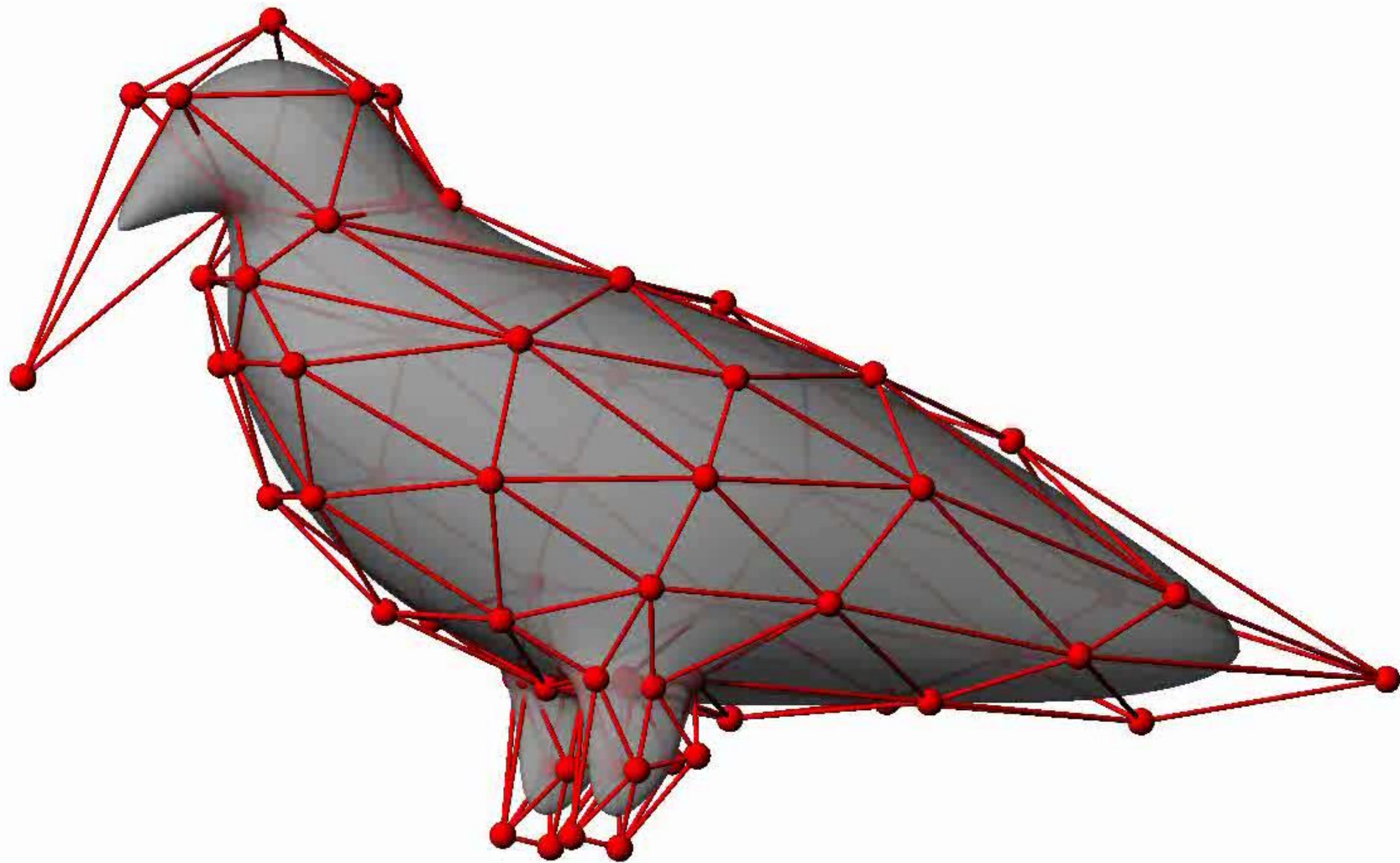
"Pixel" terms: noise level params

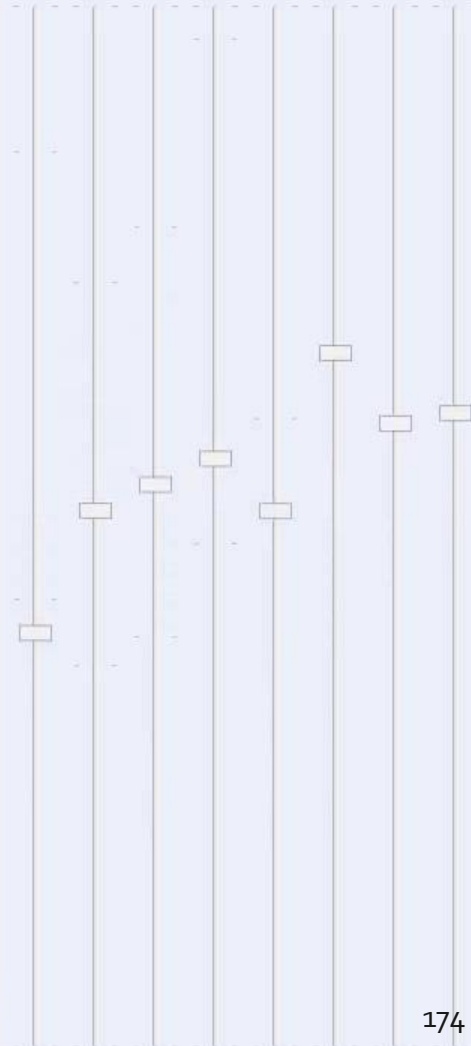
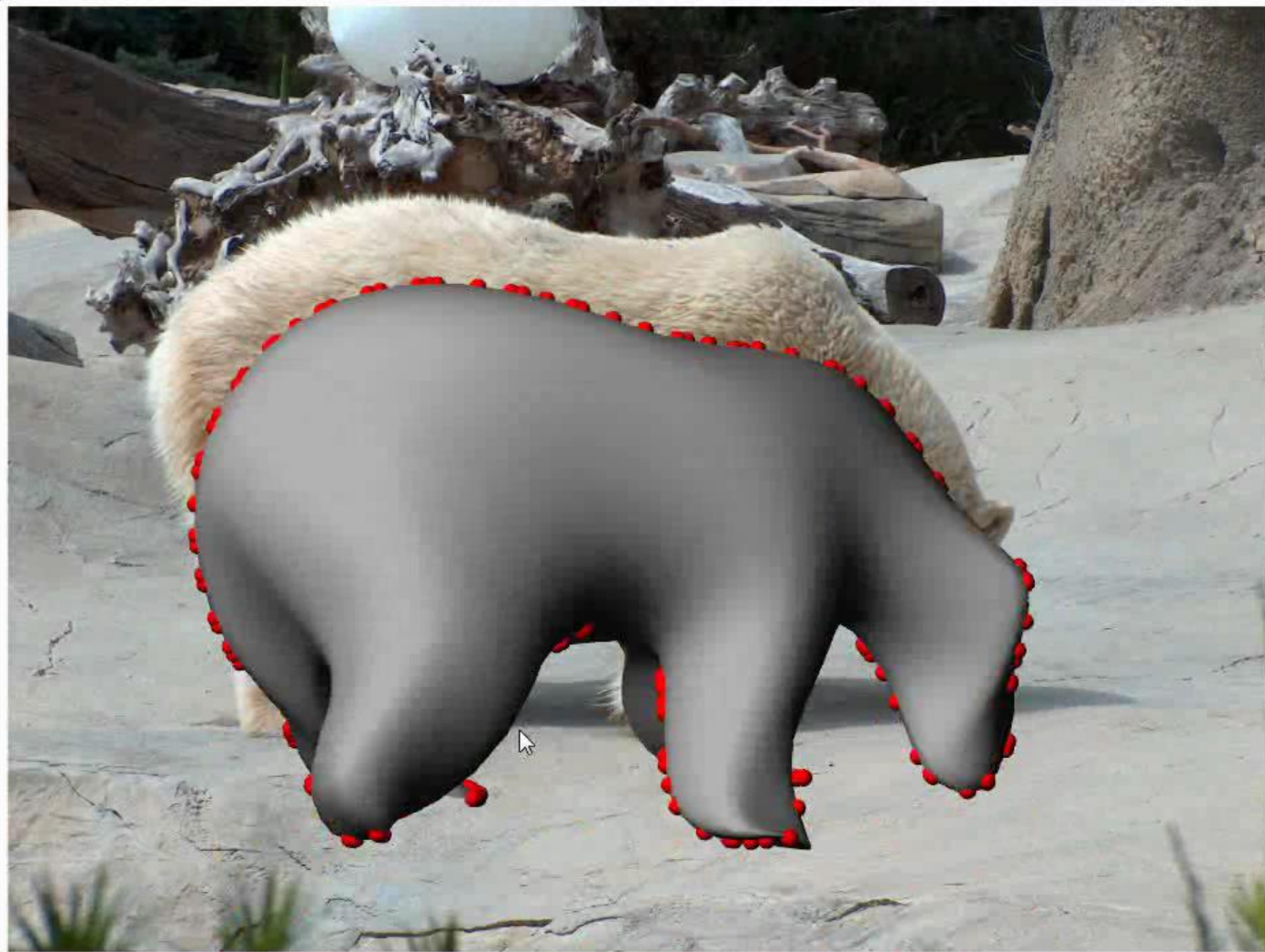
"Dimensionless" terms

"Smoothness" terms

$$E = \sum_{i=1}^n (E_i^{\text{sil}} + E_i^{\text{norm}} + E_i^{\text{con}}) + \sum_{i=1}^n (E_i^{\text{cg}} + E_i^{\text{reg}}) + \xi_0^2 E_0^{\text{tp}} + \xi_{\text{def}}^2 \sum_{i=1}^n E_m^{\text{tp}}$$

$\xi_0 \backslash \xi_{\text{def}}$	0.05			0.25			0.5		
0.05									
0.25									
0.5									



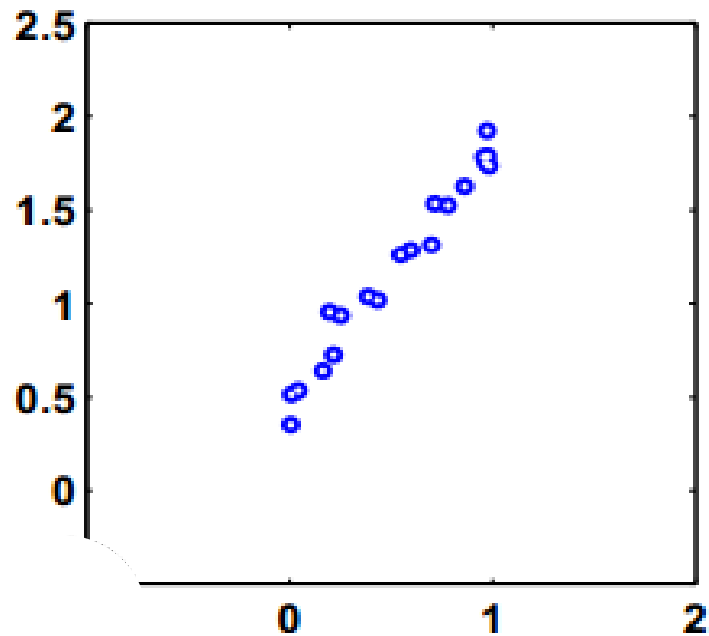




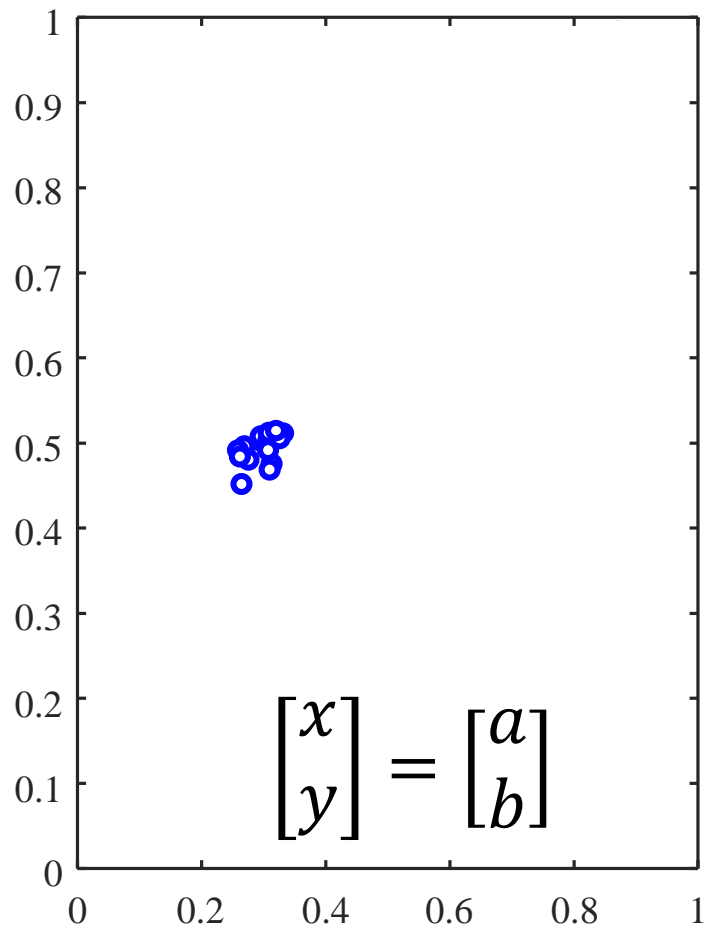


# Robust estimation

[BLACK AND RANGARANJAN, CVPR 91] – NEARLY  
[LI, PAULY, SUMNER, SIGGRAPH 08] – NEARLY  
[ZOLLHÖFER, SIGGRAPH 14] — BASICALLY  
[ZACH, ECCV 14] — DEFINITELY

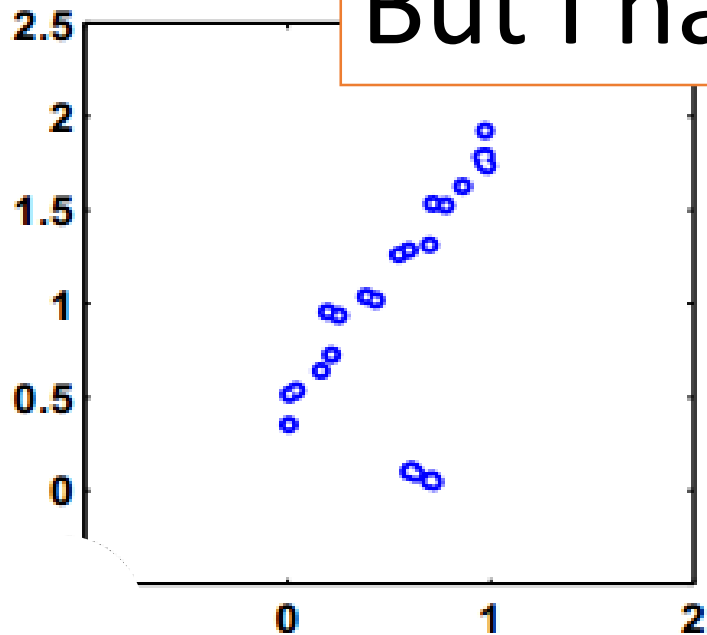


$$y = ax + b$$

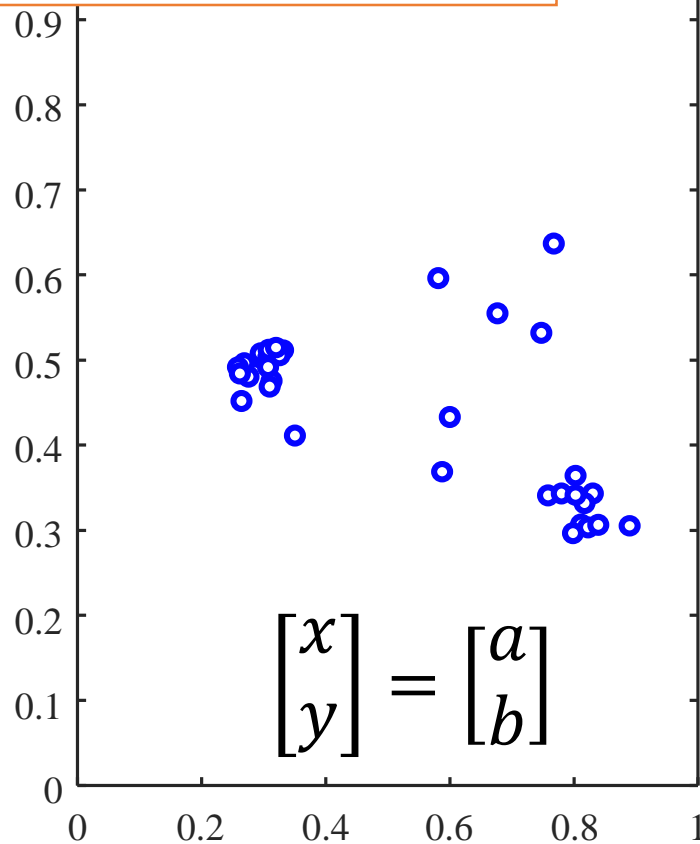


$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

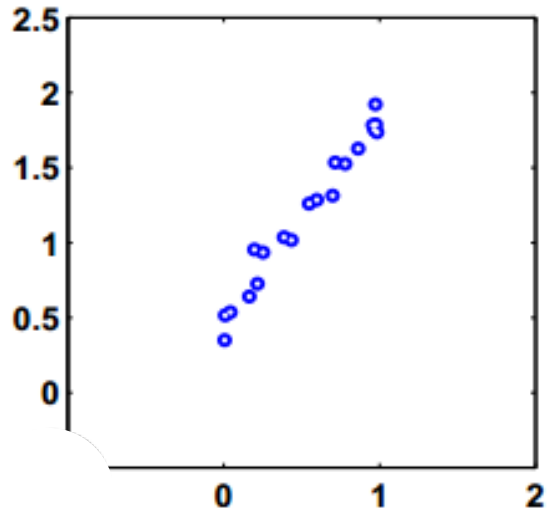
But I have “outliers” ☹️



$$y = ax + b$$



$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

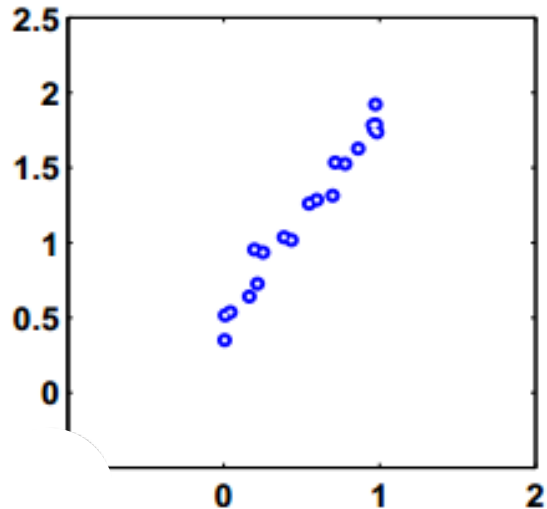


How do I fit a line to data samples  $\mathbf{s}_i = (x_i, y_i)$ ?

For this example, let us suppose true inlier model is  
 $y = a_1x + a_2 + \mathcal{N}(0, \sigma)$

Alg. 1:  $\mathbf{a} = [\mathbf{x} \text{ ones}(\mathbf{x})] \backslash \mathbf{y}$

Alg. 2:  $\mathbf{a} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_i (y_i - a_1x_i - a_2)^2$



How do I fit a line to data samples  $\mathbf{s}_i = (x_i, y_i)$ ?

For this example, let us suppose true inlier model is  
 $y = a_1x + a_2 + \mathcal{N}(0, \sigma)$

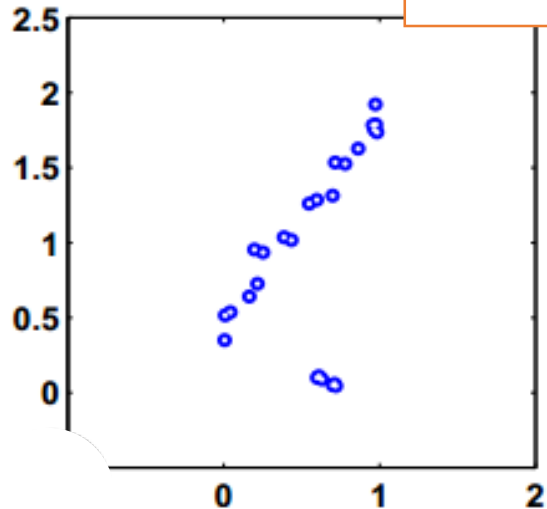
Alg. 1:  $\mathbf{a} = [\mathbf{x} \text{ ones}(\mathbf{x})] \backslash \mathbf{y}$

Alg. 2:  $\mathbf{a} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_i (y_i - a_1x_i - a_2)^2$

```
>> a = lsqnonlin(@(a) y - a(1)*x - a(2), [1 1]);
```

Works really well because objective is sum-of-squares

# But I have “outliers” 😞

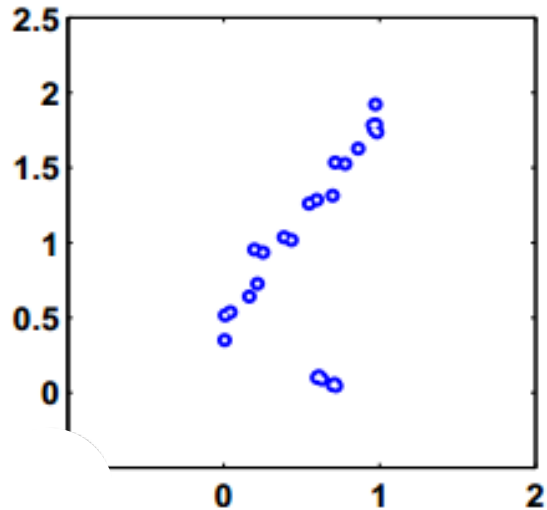


How do I fit a line to data samples  $s_i = (x_i, y_i)$ ?

For this example, let us suppose true inlier model is  
 $y = ax + b + \mathcal{N}(0, \sigma)$

Alg. 1:  ~~$a = [x \text{ ones}(x)] \backslash y$~~

Alg. 2:  $a = ?$



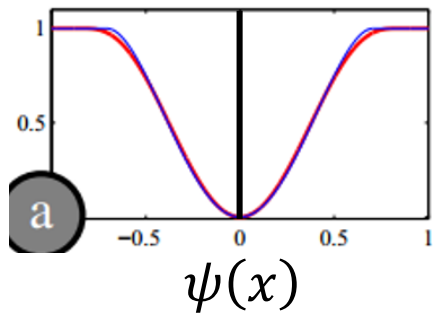
How do I fit a line to data samples  $\mathbf{s}_i = (x_i, y_i)$ ?

For this example, let us suppose true inlier model is  
 $y = ax + b + \mathcal{N}(0, \sigma)$

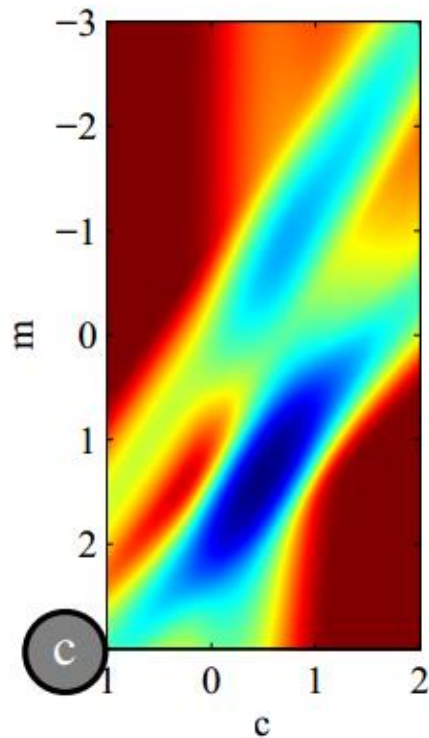
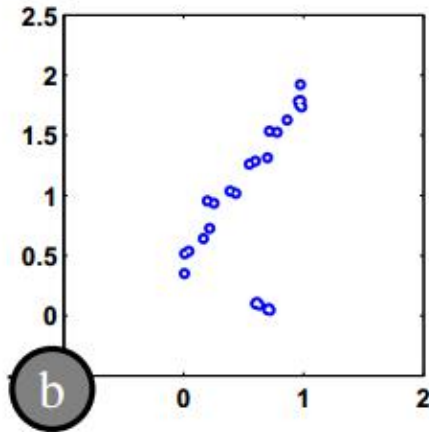
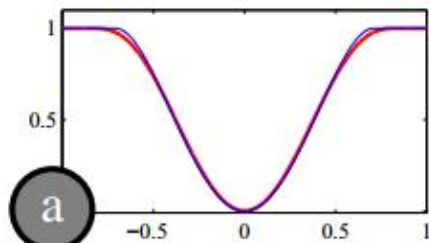
Alg. 1:  $\mathbf{a} = [\mathbf{x} \text{ ones}(\mathbf{x})] \backslash \mathbf{y}$

Alg. 2:  $\mathbf{a} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_i \psi(y_i - a_1 x_i - a_2)$

```
>> a = fminunc(@(a) sum(psi(y - a(1)*x - a(2))), [1  
1]);
```







$$\min_a \sum_i \psi(y_i - a_1 x_i - a_2)$$

Global minimum in a good place

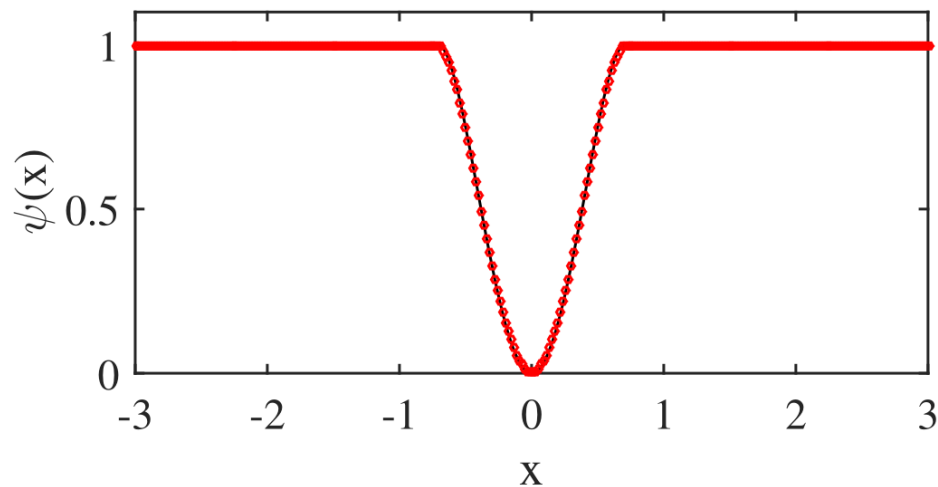
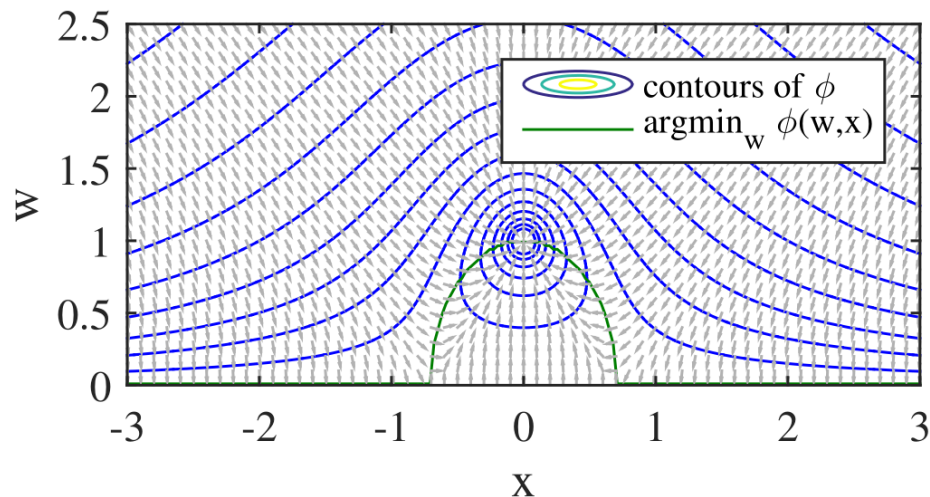
But hard to optimize:

- Multiple optima
- Huge flat spots

Robust kernels can be expressed as minimization over “outlier process” variables [e.g. Geman & Reynolds '92, Black & Rangarajan '95]

$$\phi(x, w) = w^2 x^2 + (1 - w^2)^2$$

$$\psi(x) = \min_w \phi(x, w)$$



Data residual for  $i^{\text{th}}$  data point:

$$f_i(\mathbf{a}) = y_i - a_1 x_i - a_2$$

“Lifted” robust kernel:

$$\phi(x, w) = w^2 x^2 + (1 - w^2)^2$$

Gives kernel:

$$\psi(x) = \min_w \phi(x, w)$$

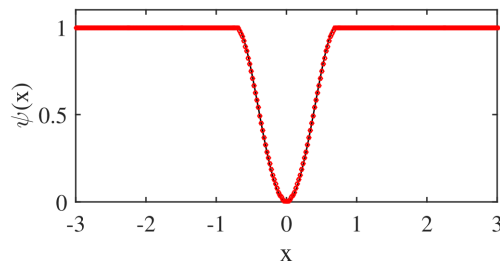
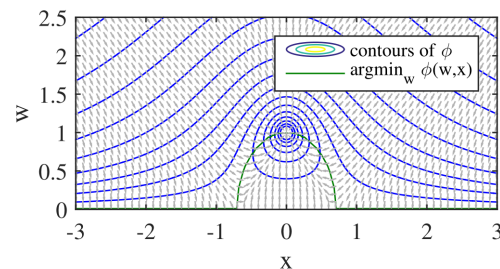
And original nasty problem:

$$\min_{\mathbf{a}} \sum_i \psi(f_i(\mathbf{a}))$$

Becomes:  $\min_{\mathbf{a}} \sum_i \min_w w^2 f_i^2(\mathbf{a}) + (1 - w^2)^2$

$$\min_{\mathbf{a}} \sum_i \min_{w_i} w_i^2 f_i^2(\mathbf{a}) + (1 - w_i^2)^2$$

$$\min_{\mathbf{a}} \min_{w_i} \sum_i w_i^2 f_i^2(\mathbf{a}) + (1 - w_i^2)^2$$



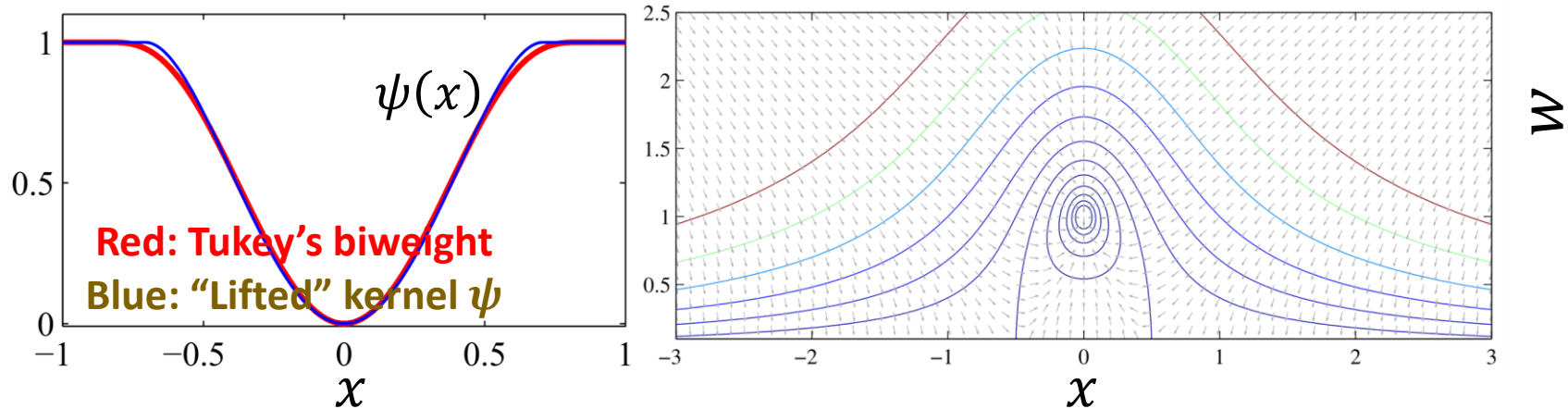
Which is in the Gauss-Newton form...

$$\psi(x) = \min_w w^2 x^2 + (1 - w^2)^2 = f(x) = \begin{cases} \frac{r^2}{2} \left( 2 - \frac{r^2}{2} \right), & x < 0 \\ 1, & x \geq 0 \end{cases}$$

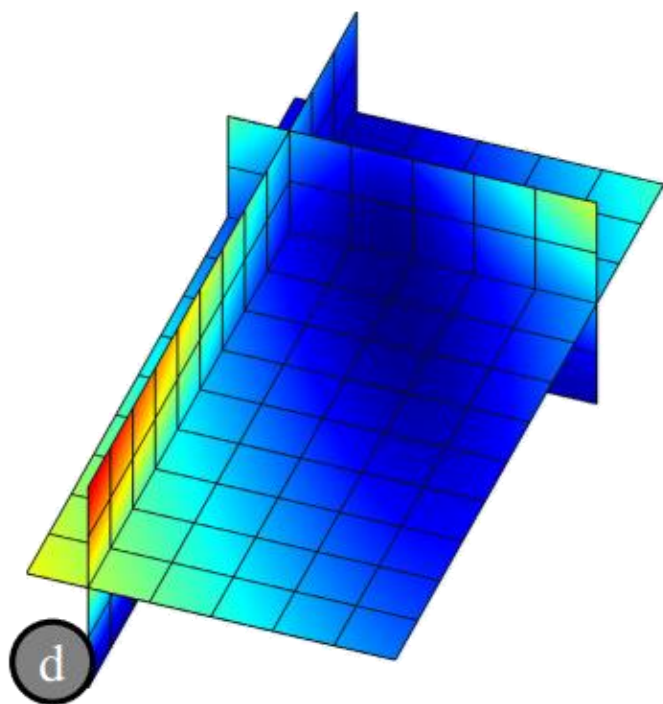
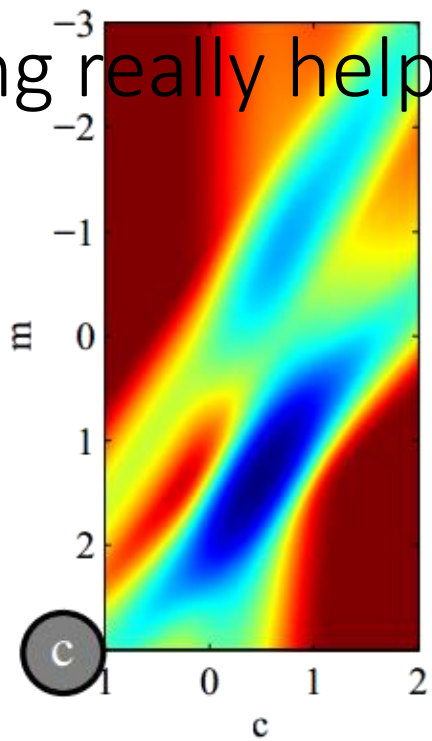
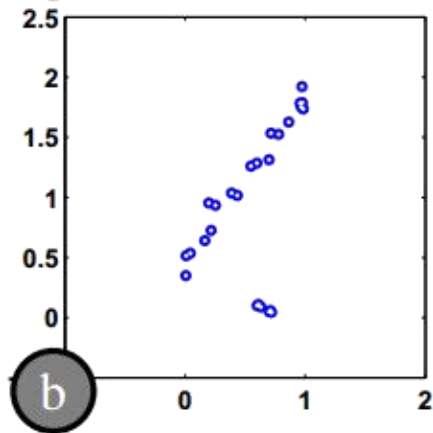
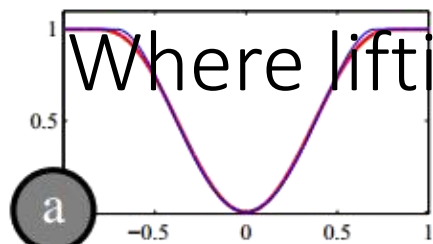
$$\psi(x) = \min_w \phi(x, w) \quad [\text{Zöllhofer et al '14}]$$

$$\phi(x, w) = w^2 x^2 + (1 - w^2)^2$$

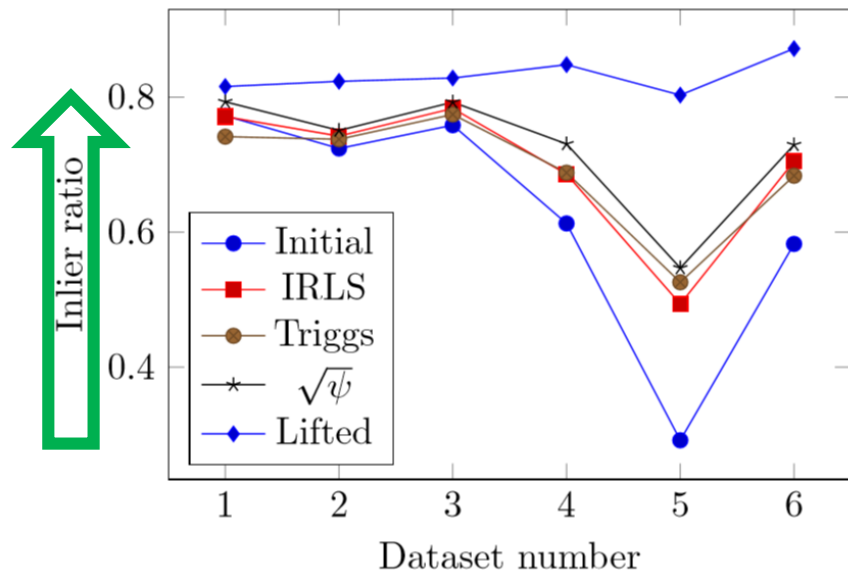
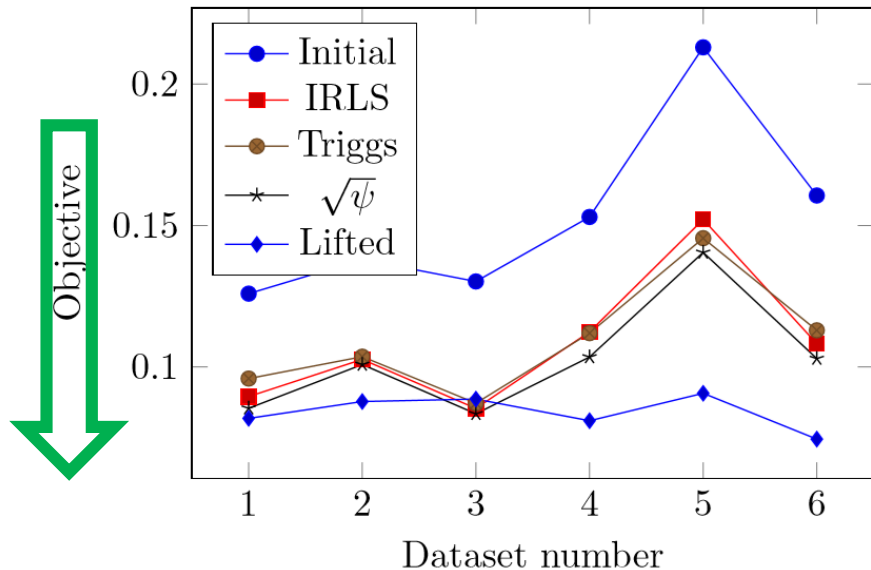
[Li, Sumner, Pauly '08]



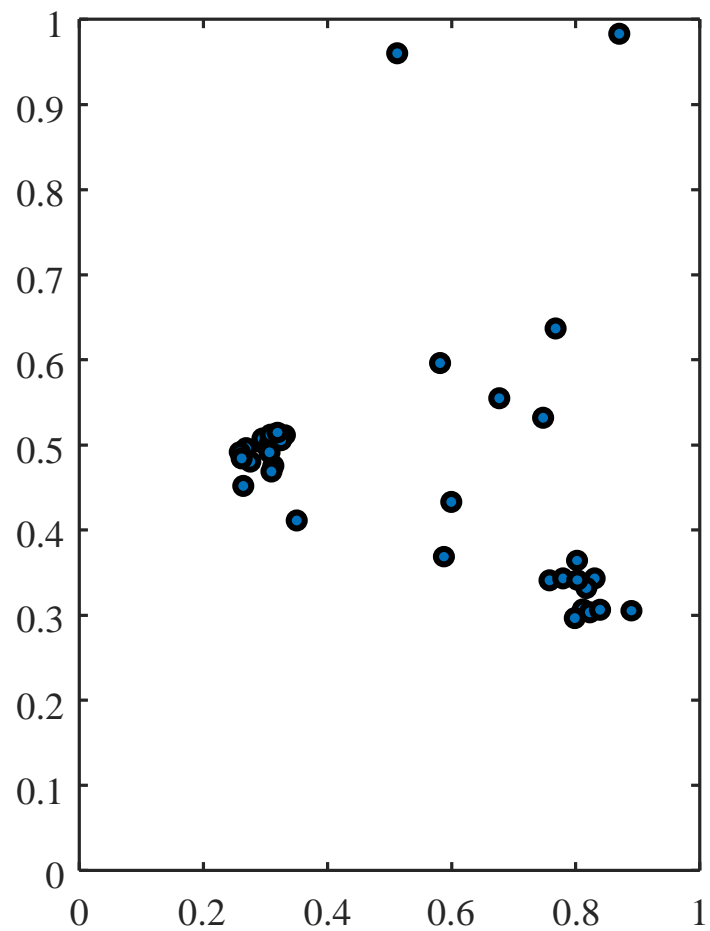
Where lifting really helps

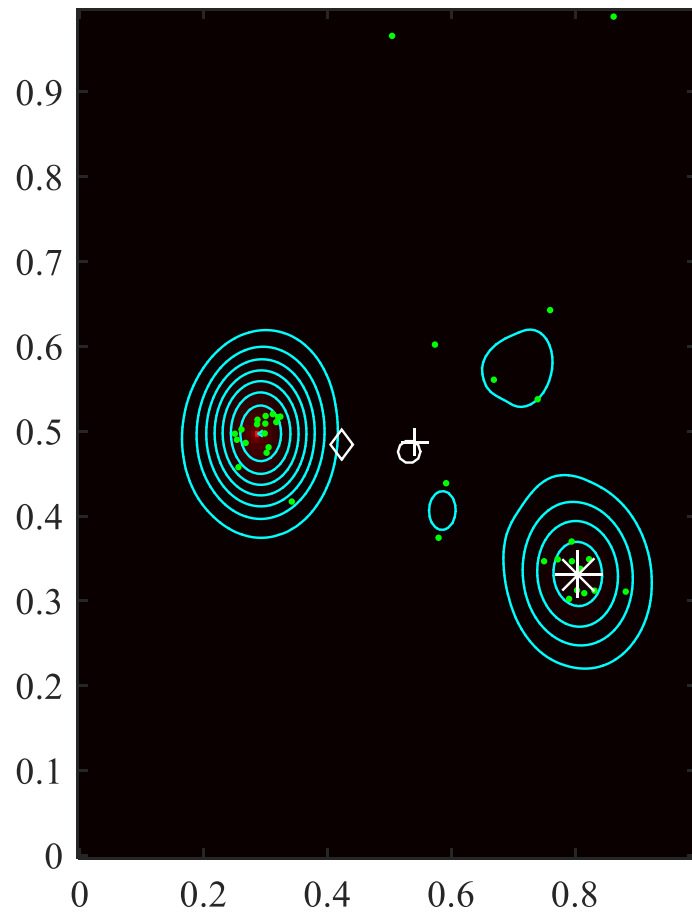
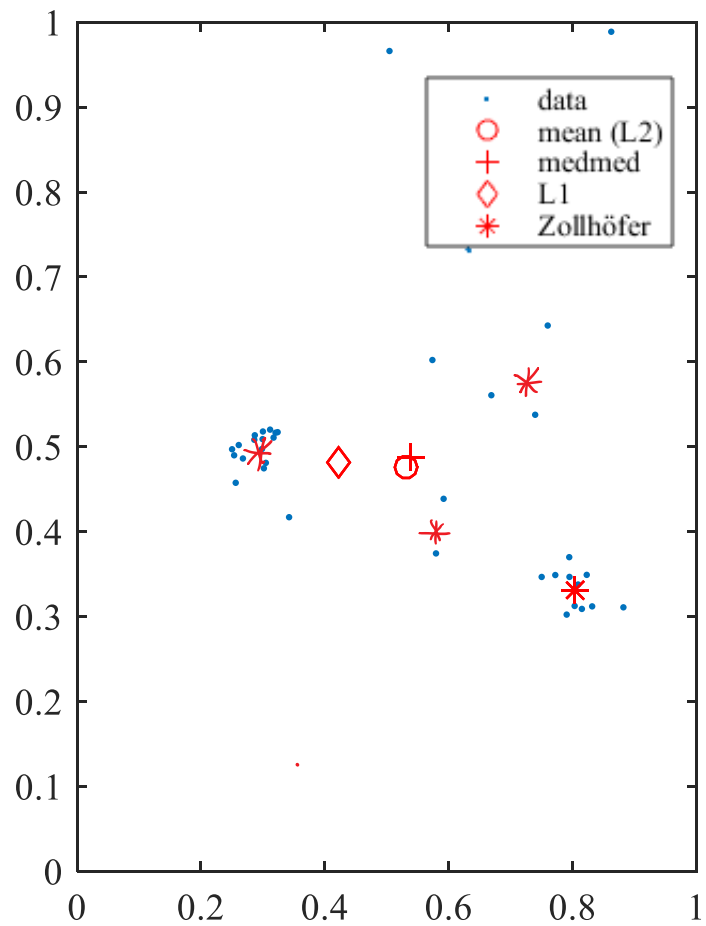


## 3D reconstruction datasets: up to $10^6$ parameters, $10^6$ measurements



Before [Zach '14], no-one used the Gauss-Newton structure, so never beat IRLS (iterated reweighted least squares), with its ICP-like convergence.







SOFTWARE

SEARCH

- Introduction
  - License
  - Getting Started
  - Contributing
  - Building OpenSubdiv
  - Code Examples
  - Roadmap
  - References
- Release 3.0
  - Overview
  - Porting Guide: 2.0 to 3.0
  - Subdivision Compatibility
- Subdivision Surfaces
  - Introduction
  - Topology
  - Uniform
  - Feature Adaptive
  - Boundary Interpolation
  - Face-Varying Interpolation
  - Semi-Sharp Creases
  - Modeling Tips
- OpenSubdiv User Guide
  - API Overview
  - Sdc
  - Vtr
  - Far
    - Topology Refiner
    - Topology Refiner Factory
    - Primvar Refiner
    - Patch Table
    - Stencil Table
  - Osd
    - Shader Interface
- Tutorials
- Historical But Relevant
  - Hbr
    - Using Hbr
  - Hierarchical Edits

# Introduction



©Disney/Pixar

# Ceres Solver

Ceres Solver <sup>[1]</sup> is an open source C++ library for modeling and solving large, complicated optimization problems. It is a feature rich, mature and performant library which has been used in production at Google since 2010. Ceres Solver can solve two kinds of problems.

1. [Non-linear Least Squares](#) problems with bounds constraints.
2. General unconstrained optimization problems.

## Getting started

- Download the [latest stable release](#) or clone the Git repository for the latest development version.

```
git clone https://ceres-solver.googlesource.com/ceres-solver
```

*Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.*

#### Contents [\[hide\]](#)

- 1 Overview
- 2 Documentation
- 3 Requirements
- 4 License
- 5 Compiler support
- 6 Get support
- 7 Bug reports
- 8 Mailing list
- 9 IRC Channel
- 10 Contributing to Eigen
- 11 Projects using Eigen
- 12 Credits

## Announcements

**Eigen 3.2.8 released!** (16.02.2016) [\[i\]](#)

**Eigen3.3-beta1 released!** (16.12.2015) [\[i\]](#)

**Eigen 3.2.7 released!** (05.11.2015) [\[i\]](#)

**Eigen 3.2.6 released!** (01.10.2015) [\[i\]](#)

**Eigen 3.3-alpha1 released!** (04.09.2015) [\[i\]](#)

## Get it

The **latest stable release** is Eigen 3.2.8. Get it here: [tar.bz2](#), [tar.gz](#), [zip](#). [Changelog](#).

The **latest development release** is Eigen **3.3-beta1**. Get it here: [tar.bz2](#), [tar.gz](#), [zip](#). [Changelog](#).

The **unstable** source code from the **development branch** is there: [tar.bz2](#), [tar.gz](#), [zip](#).

To check out the Eigen repository using [Mercurial](#), also known as "hg", do:

```
hg clone https://bitbucket.org/eigen/eigen/
```

Looking for the outdated Eigen2 version? Check it [here](#).

[ [other downloads](#) ] [ [browse the source code](#) ]

## Overview

### • Eigen is versatile.

- It supports all matrix sizes, from small fixed-size matrices to arbitrarily large dense matrices, and even sparse matrices.
- It supports all standard numeric types, including `std::complex`, integers, and is easily extensible to [custom numeric types](#).
- It supports various [matrix decompositions](#) and [geometry features](#).
- Its ecosystem of [unsupported modules](#) provides many specialized features such as non-linear optimization, matrix functions, a polynomial solver, FFT, and much more.

### • Eigen is fast.

- Expression templates allow to intelligently remove temporaries and enable [lazy evaluation](#), when that is appropriate.
- [Explicit vectorization](#) is performed for SSE 2/3/4, ARM NEON (32-bit and 64-bit), PowerPC AltiVec/VSX (32-bit and 64-bit) instruction sets, and now S390x SIMD (ZVector) with graceful fallback to non-vectorized code.
- Fixed-size matrices are fully optimized: dynamic memory allocation is avoided, and the loops are unrolled when that makes sense.
- For large matrices, special attention is paid to cache-friendliness.



Main page

Forum

Bugs & feature requests

FAQ

Contributing

Benchmark

Publications

Recent changes

▼ Documentation

Eigen 3

Dev branch

► Tools



```

template <typename _Scalar>
struct EllipseFitting : LevenbergMarquardtFunctor<_Scalar>
{
    typedef ColPivHouseholderQR<Matrix<Scalar, Dynamic, Dynamic> > DenseSolver;
    typedef BlockDiagonalSparseQR<JacobianType, DenseSolver> LeftSuperBlockSolver;
    typedef BlockSparseQR<JacobianType, LeftSuperBlockSolver, DenseSolver> QRSolver;

    const Eigen::Matrix<double, 3, Eigen::Dynamic> ellipsePoints;

    static const int nParamsModel = 5;

    EllipseFitting(Eigen::Matrix<double, 3, Eigen::Dynamic>& points ) :
        Base(nParamsModel + points.cols(), points.cols()*2),
        ellipsePoints(points)
    {
    }

    // Functor functions
    int operator()(const InputType& uv, ValueType& fvec) {
        int npoints = ellipsePoints.cols();
        auto params = uv.m.tail(nParamsModel);
    }
}

```

```

template <typename _Scalar>
struct EllipseFitting : LevenbergMarquardtFunctor<_Scalar>
{
    typedef ColPivHouseholderQR<Matrix<Scalar, Dynamic, Dynamic> > DenseSolver;
    typedef BlockDiagonalSparseQR<JacobianType, DenseSolver> LeftSuperBlockSolver;
    typedef BlockSparseQR<JacobianType, LeftSuperBlockSolver, DenseSolver> QRSolver;

    const Eigen::Matrix<double, 3, Eigen::Dynamic> ellipsePoints;

    static const int nParamsModel = 5;

    EllipseFitting(Eigen::Matrix<double, 3, Eigen::Dynamic>& points ) :
        Base(nParamsModel + points.cols(), points.cols()*2),
        ellipsePoints(points)
    {
    }

    // Functor functions
    int operator()(const InputType& uv, ValueType& fvec) {
        int npoints = ellipsePoints.cols();
        auto params = uv.m.tail(nParamsModel);
        double a = params[0];
        double b = params[1];
        double x0 = params[2];
        double y0 = params[3];
        double r = params[4];
        for(int i=0; i < npoints; i++) {
            double t = uv.m[i];
            double x = a*cos(t)*cos(r) - b*sin(t)*sin(r) + x0;
            double y = a*cos(t)*sin(r) + b*sin(t)*cos(r) + y0;
            fvec(2*i + 0) = ellipsePoints(0,i) - x;
            fvec(2*i + 1) = ellipsePoints(1,i) - y;
        }
    }
}

```

```

int df(const InputType& uv, JacobianType& fjac) {
    // X_i - (a*cos(t_i) + x0)
    // Y_i - (b*sin(t_i) + y0)
    int npoints = ellipsePoints.cols();
    auto params = uv.m.tail(nParamsModel);
    double a = params[0];
    double b = params[1];
    double r = params[4];
    for(int i=0; i<npoints; i++) {
        double t = uv.m[i];
        fjac.coeffRef(2*i, npoints+0) = -cos(t)*cos(r);
        fjac.coeffRef(2*i, npoints+1) = +sin(t)*sin(r);
        fjac.coeffRef(2*i, npoints+2) = -1;
        fjac.coeffRef(2*i, npoints+4) = +a*cos(t)*sin(r) + b*sin(t)*cos(r);
        fjac.coeffRef(2*i, i) = +a*cos(r)*sin(t) + b*sin(r)*cos(t);

        fjac.coeffRef(2*i+1, npoints+0) = -cos(t)*sin(r) ;
        fjac.coeffRef(2*i+1, npoints+1) = -sin(t)*cos(r) ;
        fjac.coeffRef(2*i+1, npoints+3) = -1;
        fjac.coeffRef(2*i+1, npoints+4) = -a*cos(t)*cos(r) + b*sin(t)*sin(r);
        fjac.coeffRef(2*i+1, i) = +a*sin(r)*sin(t) - b*cos(r)*cos(t);
    }

    fjac.makeCompressed();
    return 0;
}

```

$$\log \left( \prod_{i=1}^N \sum_{k=1}^K w_k \det(2\pi \Sigma_k)^{-\frac{1}{2}} \exp \left( -\frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \right) \prod_{k=1}^K C(D, m) |\Sigma_k|^m \exp \left( -\frac{1}{2} \text{trace}(\Sigma_k) \right) \right) \quad (1)$$

$$\text{s.t. } \sum_{k=1}^K w_k = 1 \text{ and } \Sigma_k \text{ is positive-semidefinite } \forall k \in \{1, \dots, K\}$$

where  $\mathbf{x} \in \mathbb{R}^{D \times N}$  are data points,  $\mathbf{w} \in \mathbb{R}^K$  weights,  $\boldsymbol{\mu} \in \mathbb{R}^{D \times K}$  means,  $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D \times K}$  covariance matrices,  $m$  is a Wishart hyperparameter and  $C$  is a function not dependent on independent variables. To integrate the constraints on weights and covariances into the objective function, we reparametrize the GMM function (1). After simplification, the final function to be optimized looks like

$$\log(p(\mathbf{x}; \boldsymbol{\alpha}, \boldsymbol{\mu}, \mathbf{q}, \mathbf{l})) = \sum_{i=1}^n \log \text{sumexp} \left( \left[ \alpha_k + \text{sum}(\mathbf{q}_k) - \frac{1}{2} \|Q(\mathbf{q}_k, \mathbf{l}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)\|^2 \right]_{k=1}^K \right) - n \log \text{sumexp}(\boldsymbol{\alpha})$$

$$+ \frac{1}{2} \sum_{k=1}^K (\|\exp(\mathbf{q}_k)\|^2 + \|\mathbf{l}_k\|^2) - m \text{sum}(\mathbf{q}_k) + C'(D, m) \quad (2)$$

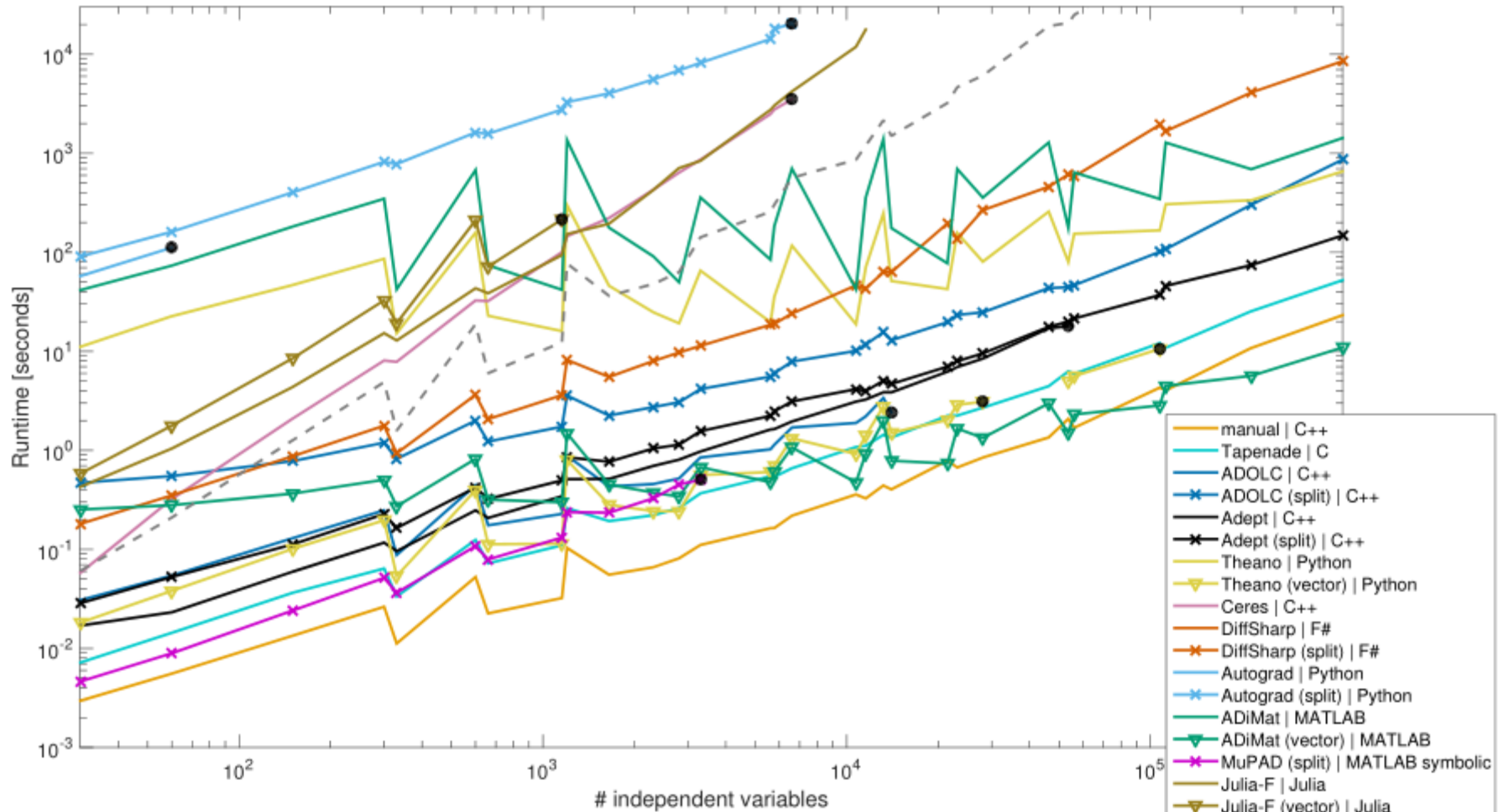
## A Benchmark of Selected Algorithmic Differentiation Tools on Some Problems in Machine Learning and Computer Vision

Filip Srajer, Zuzana Kukelova, Andrew Fitzgibbon

AD 2016



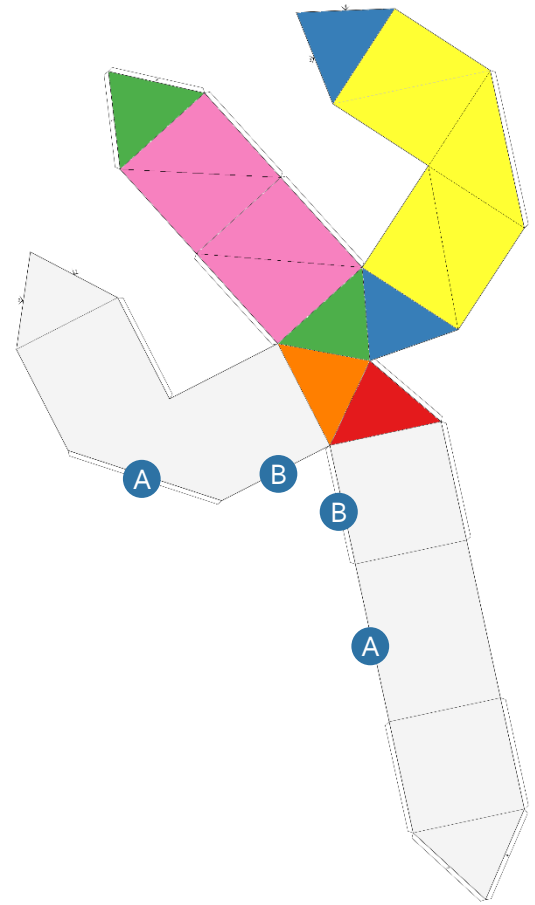
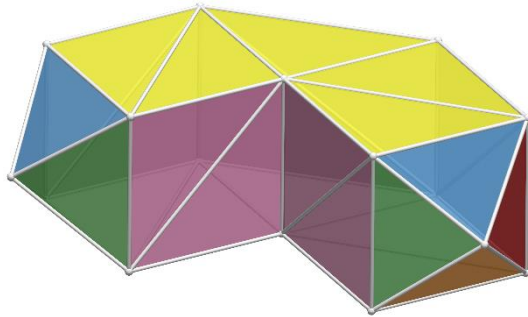
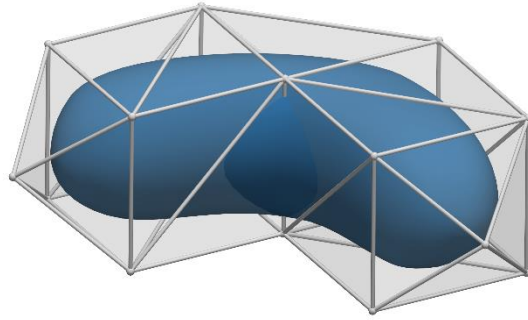
### GMM Gradient Absolute Runtimes - 10k Data Points



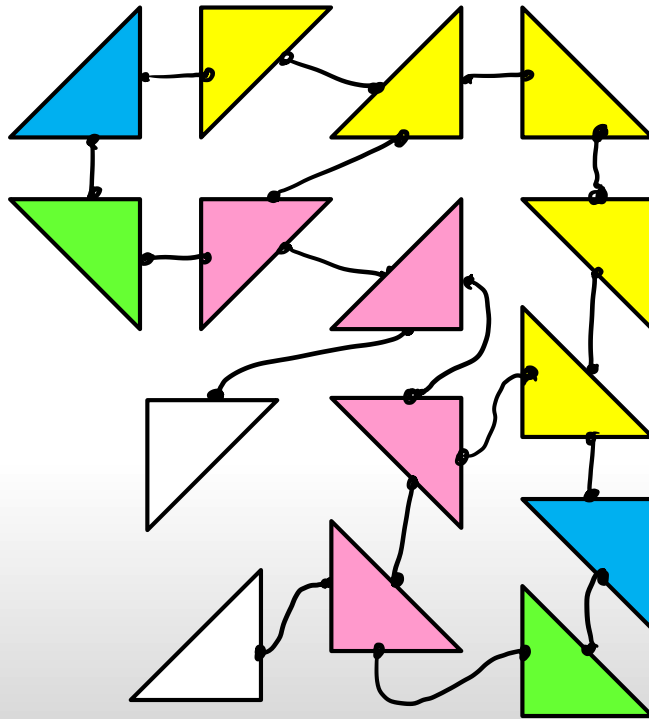
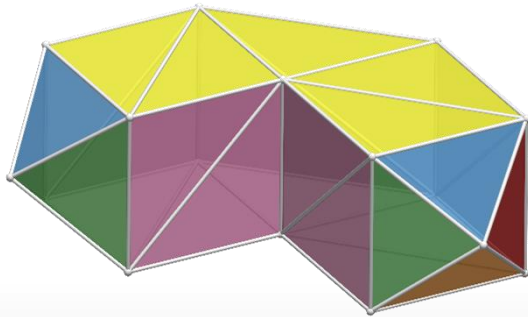


## SUBDIV PECULIARITIES 1: PIECEWISE DOMAIN

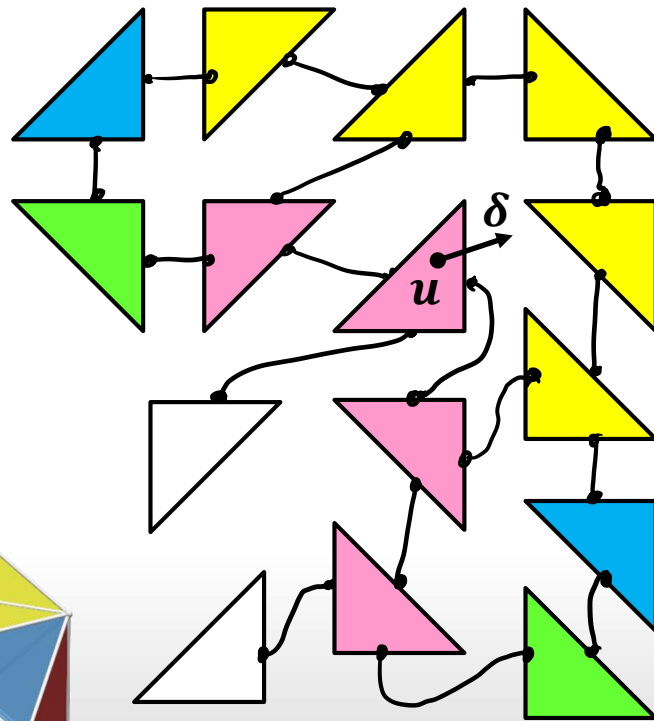
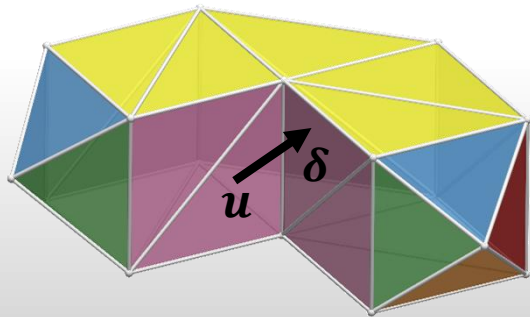
- Parameter domain  $\Omega$  is in pieces
  - Typically not unwrappable to a plane



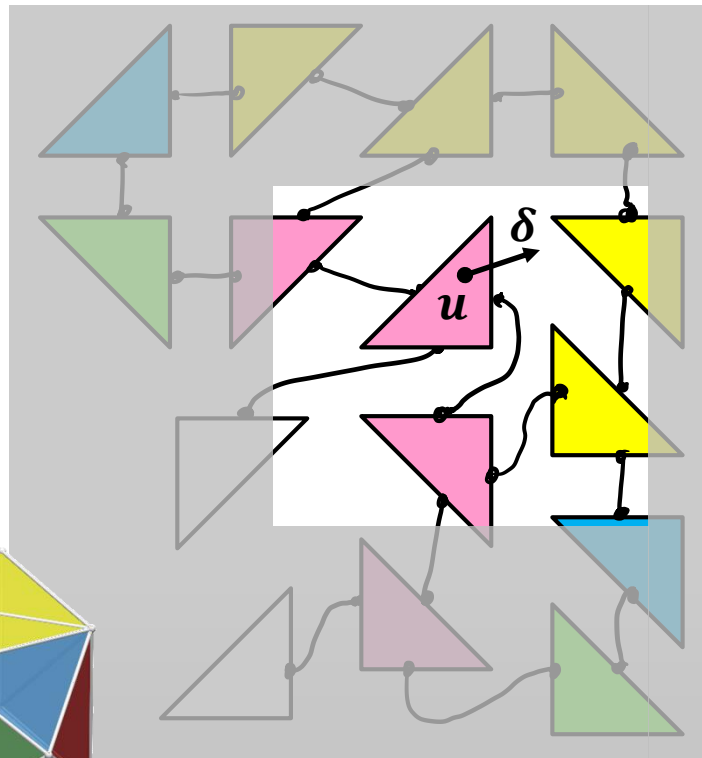
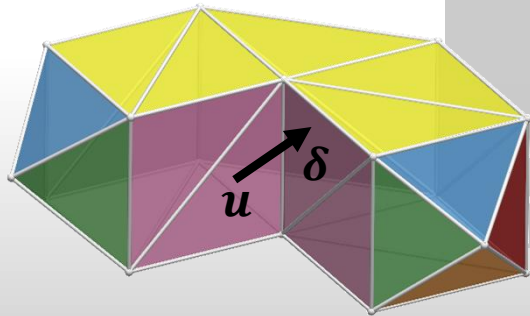
- Parameter domain  $\Omega$ : pieces with connectivity graph



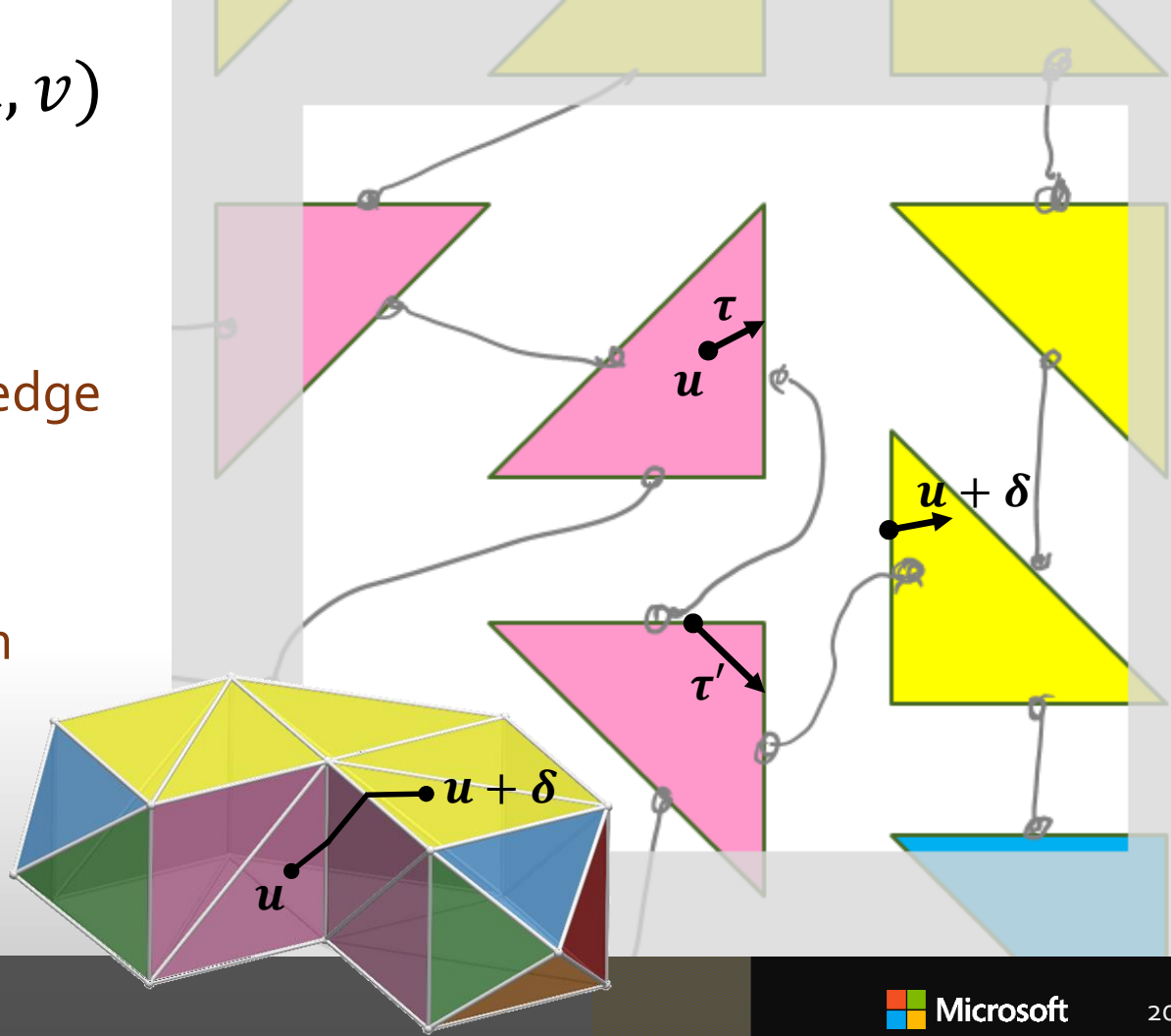
- At point  $\mathbf{u} = (p, u, v)$
- Easy to get direction  $\delta$  from  $M_u$  etc.
- But need  $\mathbf{u} + \lambda\delta$ 
  - **Override `ceres::Evaluator::Plus`**
- Easy *inside* patch
- Need *outside* too



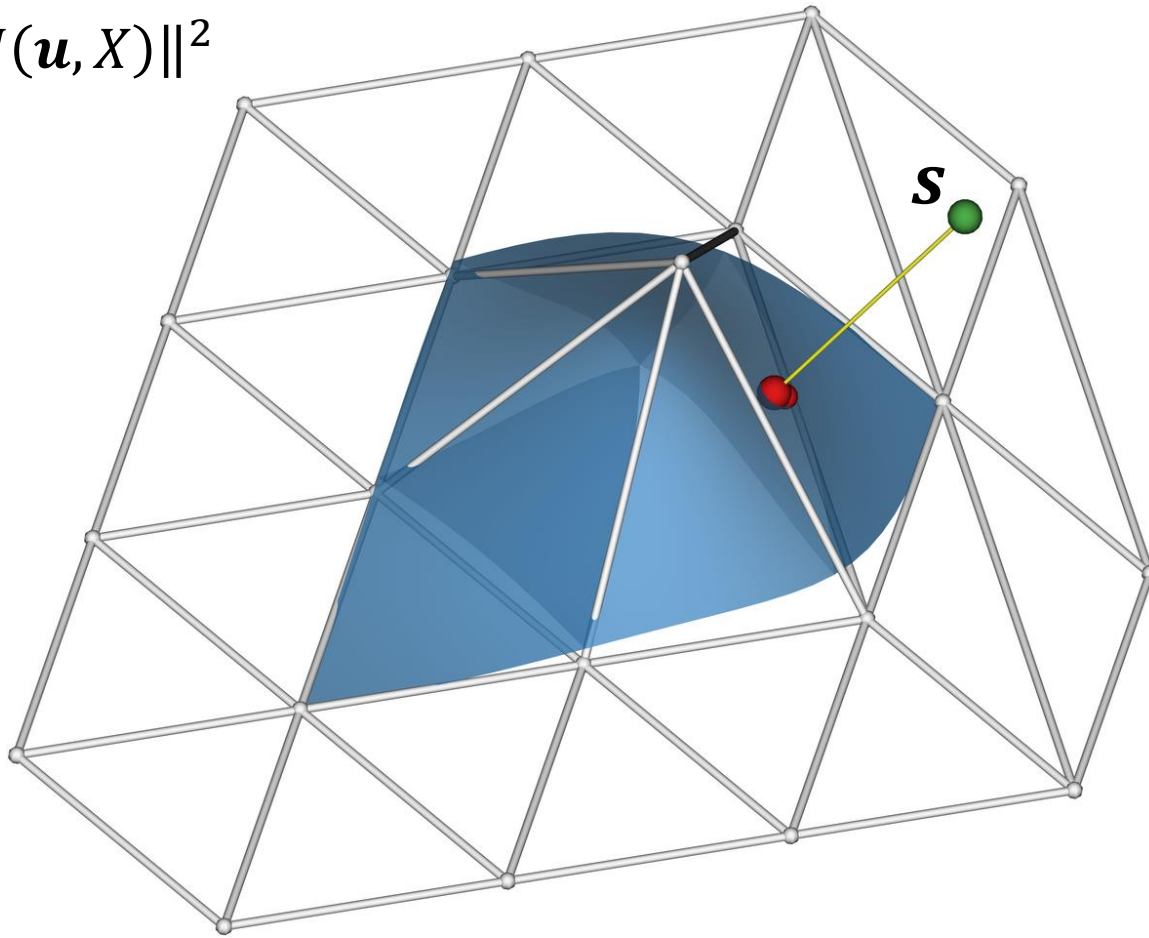
- At point  $\mathbf{u} = (p, u, v)$
- Need  $\mathbf{u} + \lambda \delta$
- *Outside* patch:
  - Move distance  $\tau$  to edge
  - Change direction
  - Move  $\delta - \tau$
  - Repeat in next patch



- At point  $\mathbf{u} = (p, u, v)$
- Need  $\mathbf{u} + \lambda \boldsymbol{\delta}$
- *Outside* patch:
  - Move distance  $\tau$  to edge
  - Change direction
  - Move  $\delta - \tau$
  - Repeat in next patch

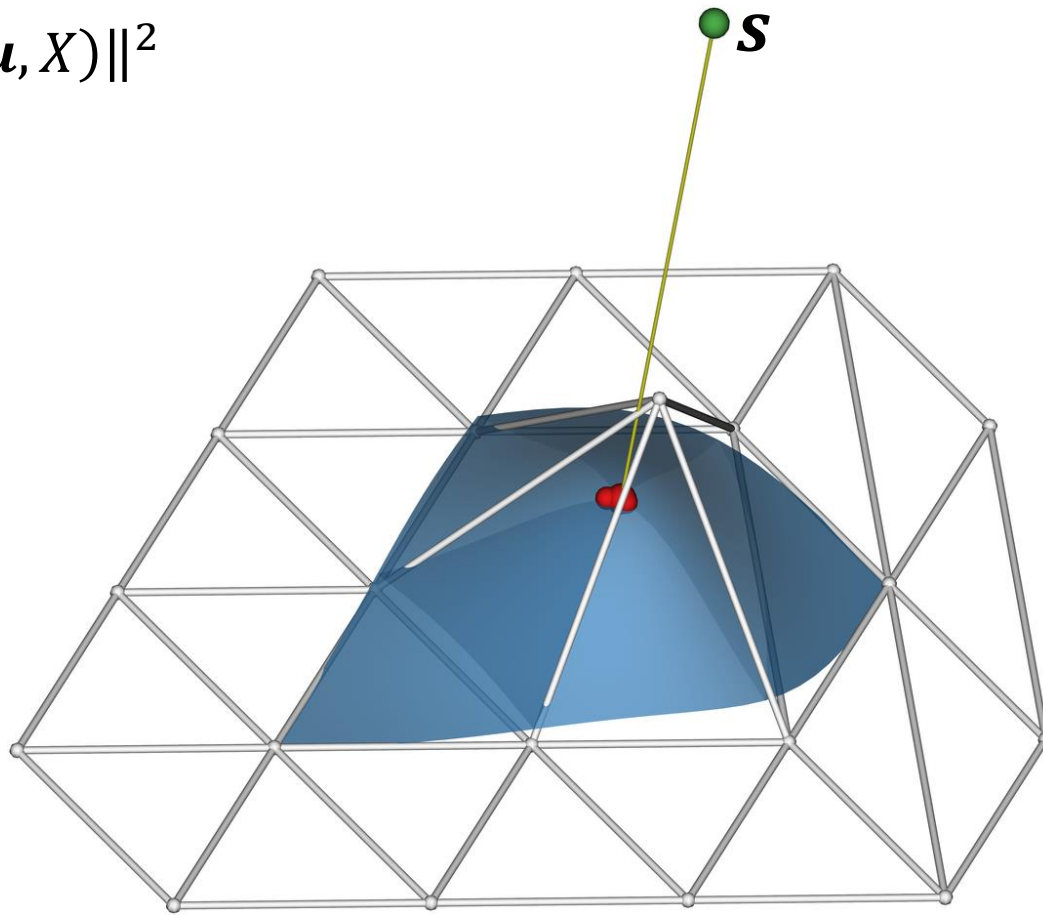


$$E(\mathbf{u}) = \|\mathbf{s} - M(\mathbf{u}, X)\|^2$$



EXAMPLE: SINGLE CLOSEST POINT PROBLEM

$$E(\mathbf{u}) = \|\mathbf{s} - M(\mathbf{u}, X)\|^2$$

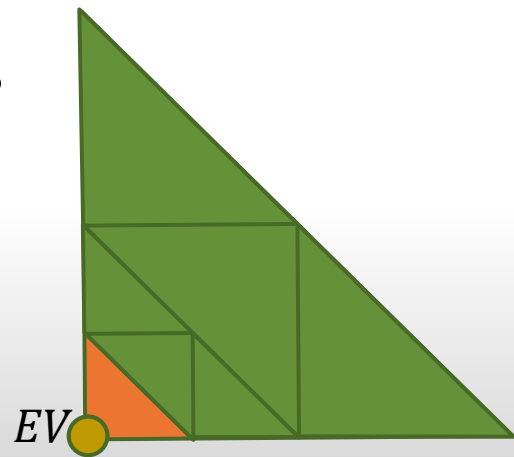


EXAMPLE: SINGLE CLOSEST POINT PROBLEM



## SUBDIV PECULIARITIES 2: EXTRAORDINARY VERTICES

- Any vertex of valency  $\neq 6$  is an “extraordinary vertex”
  - Call a triangle with an EV an “irregular triangle”
- Normals and surface at EVs well defined and well behaved
  - But spline evaluation rule is not...
- Solution: virtually subdivide irregular triangles
  - Each green element is still linear in  $X$ , quartic in  $u, v$
  - Need to generate different  $A_{ijk}$  for  $\sum A_{ijk} u^i v^j X_k$
  - All autogenerated C code using Sympy
    - Go to depth 5, and then handle “vestigial patch”
    - Initially just use spline coeffs from neighbour



```

LOOP_FUNCTION_SPECIFIER void M_7_4_7_4_0(double* m,
const double* u,
const double* x0,
const double* x1,
const double* x2,
const double* x3,
const double* x4,
const double* x5,
const double* x6,
const double* x7,
const double* x8,
const double* x9,
const double* x10,
const double* x11,
const double* x12,
const double* x13,
const double* x14) {

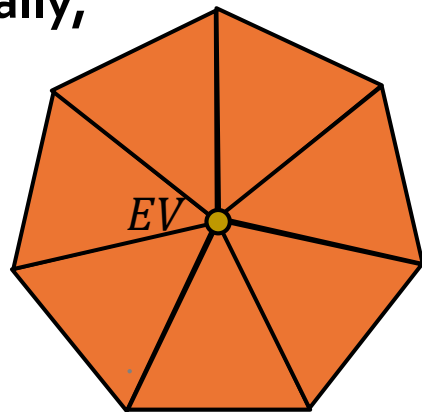
const double t26 = u[0]*u[0];
const double t0 = t26*u[0];
const double t24 = t0*u[1];
const double t21 = t0*u[0];
const double t20 = -1.00925423677687*t0 + 0.727289794784244*t21 + 1.45457958956849*t24 + 0.204156603646253*t26 + 0.189139176544278*u[0];
const double t12 = 0.0653665602547852*t21 + 0.13073312050957*t24;
const double t2 = u[0]*u[1];
const double t16 = t2*u[1];
const double t22 = -1.02456908770049*t0 + 0.731802091719096*t21 + 1.46360418343819*t24 + 0.40937272614469*t26 + 0.0675029905651878*u[0];
const double t25 = t16*u[0];
const double t29 = u[1]*u[1];
const double t30 = 0.783901952922574*t2 + 1.96044877193316*t25;
const double t23 = t2*u[0];
const double t31 = 0.117149481155709*t2 - 1.03909444846764*t25;
const double t15 = 0.166666666666667*t0 - 0.101300106411881*t21 - 0.202600212823763*t24;
const double t1 = t29*u[1];
const double t28 = t1*u[1];
const double t27 = t1*u[0];
const double t10 = 1.54426437363695*t0 + 1.54426437363695*t1 + 2.16228931204758*t16 - 1.27716982113273*t2 - 0.751683854410516*t21 + 2.16228931204758*t23 - 1.50336770882103*t24 - 1.21589933956426*t25 - 1.35422344
533665*t26 - 1.50336770882103*t27 - 0.751683854410516*t28 - 1.35422344533665*t29 + 0.522157262101323;
const double t7 = -2.05825738139938*t1 - 2.55751958917118*t16 + t20 - 2.00851644454867*t23 + 2.75378632355459*t27 + 1.3768931617773*t28 + 1.12375786149525*t29 + t30 + 0.303355685979819*u[1] + 0.0682632482712396;
const double t14 = -0.166666666666667*t1 + 0.202600212823763*t27 + 0.101300106411881*t28;
const double t11 = 0.166666666666667*t0 - t14 + 0.5*t16 - 0.101300106411881*t21 + 0.5*t23 - 0.202600212823763*t24;
const double t17 = 1.02456908770049*t1 - 1.46360418343819*t27 - 0.731802091719096*t28 - 0.40937272614469*t29 - 0.0675029905651878*u[1] + 0.0682632482712396;
const double t6 = 0.0750149862909235*t0 + 1.19205935169555*t16 + t17 - 0.272158810591444*t21 + 0.242505250285909*t23 - 0.54431762118289*t24 + 0.320448914419135*t26 + t31 - 0.273314028968999*u[0];
const double t19 = -1.00925423677687*t1 + 1.45457958956849*t27 + 0.727289794784244*t28 + 0.204156603646253*t29 + 0.189139176544278*u[1] + 0.0682632482712396;
const double t13 = 0.13073312050957*t27 + 0.0653665602547852*t28;
const double t9 = -2.05825738139938*t0 - 2.00851644454867*t16 + t19 + 1.3768931617773*t21 - 2.55751958917118*t23 + 2.75378632355459*t24 + 1.12375786149525*t26 + t30 + 0.303355685979819*u[0];
const double t8 = -0.491951150728699*t16 + t19 - 0.732663340897398*t2 - t22 + 1.54187217374866*t23 - 0.105812403346733*t25;
const double t18 = -0.0750149862909235*t1 + 0.54431762118289*t27 + 0.272158810591444*t28 - 0.320448914419135*t29 + 0.273314028968999*u[1] - 0.0682632482712396;
const double t3 = 0.242505250285909*t16 - t18 - t22 + 1.19205935169555*t23 + t31;
const double t4 = 0.0750149862909235*t0 - 0.580738903329257*t16 - t18 + 0.940393634770956*t2 - 0.272158810591444*t21 - 0.580738903329257*t23 - 0.54431762118289*t24 - 0.413584500673319*t25 + 0.320448914419135*t26
- 0.273314028968999*u[0];
const double t5 = 1.54187217374866*t16 + t17 - 0.732663340897398*t2 + t20 - 0.491951150728699*t23 - 0.105812403346733*t25;
m[0] = t10*x0[0] + t11*x11[0] + t12*x12[0] + t12*x13[0] + t13*x10[0] + t13*x9[0] - t14*x8[0] + t15*x14[0] + t3*x4[0] + t4*x5[0] + t5*x3[0] + t6*x6[0] + t7*x1[0] + t8*x7[0] + t9*x2[0];
m[1] = t10*x0[1] + t11*x11[1] + t12*x12[1] + t12*x13[1] + t13*x10[1] + t13*x9[1] - t14*x8[1] + t15*x14[1] + t3*x4[1] + t4*x5[1] + t5*x3[1] + t6*x6[1] + t7*x1[1] + t8*x7[1] + t9*x2[1];
m[2] = t10*x0[2] + t11*x11[2] + t12*x12[2] + t12*x13[2] + t13*x10[2] + t13*x9[2] - t14*x8[2] + t15*x14[2] + t3*x4[2] + t4*x5[2] + t5*x3[2] + t6*x6[2] + t7*x1[2] + t8*x7[2] + t9*x2[2];
}

```

## SUBDIV PECULIARITIES 2: VANISHING DERIVATIVES

“Neighbour extrapolation” for vestigial patch looks OK visually,  
but EVs have other issues:

- Vanishing first derivatives:  $\lim_{\mathbf{u} \rightarrow EV} M_{\mathbf{u}}(\mathbf{u}, X) = \mathbf{0}$ 
  - Saddle point for gradient-based optimization.
- Unbounded second derivatives
  - Infinite thin-plate energy (inconvenience).
  - Derivatives with respect to normal, although well defined, are unstable using chain-rule (inconvenience).
- Solutions
  - **Reparameterise** the function near the extraordinary vertex.
  - **Replace** the function near the extraordinary vertex.



Example bad parameterization:

$$\mathbf{m}(s) = (x, y) = (\sqrt{s}, \sin(\sqrt{s})) \quad s \in \mathbb{R}^+$$

$$\mathbf{m}'(s) = \frac{d\mathbf{m}}{ds}(s) = \left( \frac{1}{2\sqrt{s}}, \frac{\cos(\sqrt{s})}{2\sqrt{s}} \right)$$

$$\Rightarrow \lim_{s \rightarrow 0} \mathbf{m}'(s) \rightarrow (\infty, \infty)$$

Reparameterise  $s = t^2$

$$\mathbf{m}(t) = (x, y) = (t, \sin(t))$$

$$\mathbf{m}'(t) = \frac{d\mathbf{m}}{dt}(t) = (1, \cos(t))$$

$$\Rightarrow \lim_{t \rightarrow 0} \mathbf{m}_t(t) \rightarrow (1, 1)$$

WHERE LIFTING REALLY HURTS ...

- Bundle adjustment

$$\min_{U_{1..N}} \min_{V_{1..M}} \sum_{i,j \in S} \psi(\|\mathbf{m}_{ij} - \pi(U_i^\top V_j)\|)$$

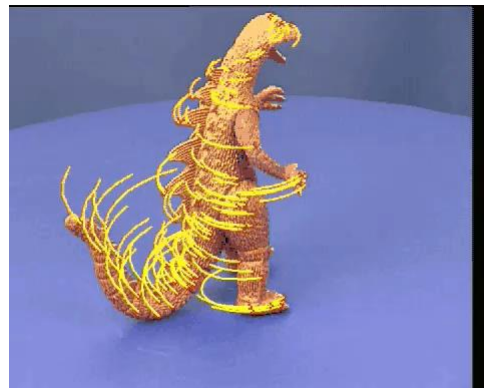
- Sum over set of camera-point observations  $S$
- Robust kernel  $\psi$
- Nonlinear projection  $\pi: \mathbb{R}^3 \mapsto \mathbb{R}$



- Bundle adjustment

$$\min_{U_{1..N}} \min_{V_{1..M}} \sum_{i,j \in S} \psi(\|\mathbf{m}_{ij} - \pi(U_i^\top V_j)\|)$$

- Sum over set of camera-point observations  $S$
- Robust kernel  $\psi$
- Nonlinear projection  $\pi: \mathbb{R}^3 \mapsto \mathbb{R}$

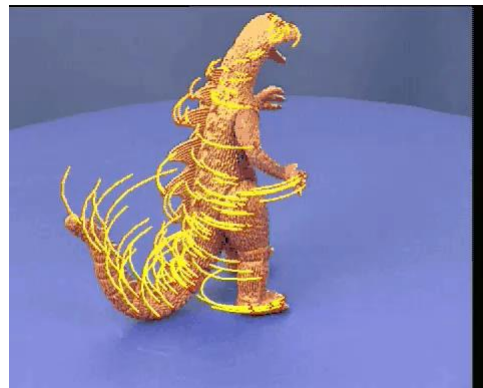


Success rate: 0.1%

- Bundle adjustment

$$\min_{U_{1..N}} \min_{V_{1..M}} \sum_{i,j \in S} \|\mathbf{m}_{ij} - \pi(U_i^\top V_j)\|^2$$

- Sum over set of camera-point observations  $S$
- ~~■ Robust kernel  $\psi$~~
- Nonlinear projection  $\pi: \mathbb{R}^3 \mapsto \mathbb{R}$



Success rate: 1%

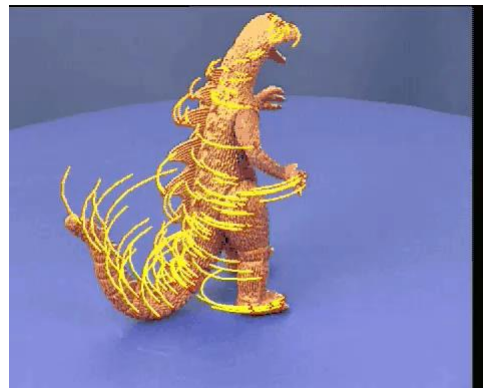
- Bundle adjustment

$$\min_{U_{1..N}} \min_{V_{1..M}} \sum_{i,j \in S} \|\mathbf{m}_{ij} - U_i^\top V_j\|^2$$

- Sum over set of camera-point observations  $S$

- ~~■ Robust kernel  $\psi$~~

- ~~■ Nonlinear projection  $\pi: \mathbb{R}^3 \mapsto \mathbb{R}$~~



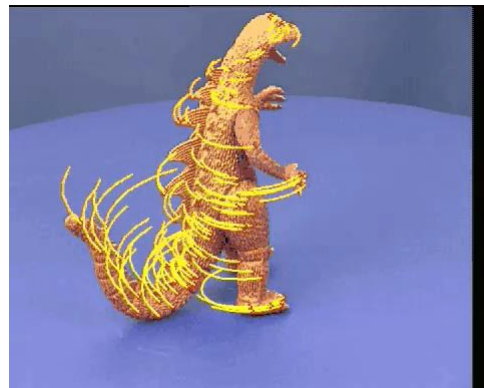
Success rate: 10%

[0-30% depending on  
some stuff]

- Which is just matrix completion:

$$\min_{U \in \mathbb{R}^{M \times K}} \min_{V \in \mathbb{R}^{N \times K}} \|M - UV^T\|^2$$

— I really want answer for  
 $K = 3, 4, 5, \dots$



Success rate: ? %

Algorithm	Framework	Manifold retraction
ALS [4]	RW3 (ALS)	None
PowerFactorization [4, 29]	RW3 (ALS)	$q$ -factor ( $U = \text{qorth}(U)$ )
LM-S [7]	Newton + $\langle \text{Damping} \rangle$	orth (replaced by $q$ -factor)
LM-S <sub>GN</sub> [8, 12]	RW1 (GN) + $\langle \text{Damping} \rangle$ (DRW1 equiv.)	orth (replaced by $q$ -factor)
LM-M [7]	Reduced <sub>r</sub> Newton + $\langle \text{Damping} \rangle$	orth (replaced by $q$ -factor)
LM-M <sub>GN</sub> [7]	Reduced <sub>r</sub> RW1 (GN) + $\langle \text{Damping} \rangle$	orth (replaced by $q$ -factor)
Wiberg [19]	RW2 (Approx. GN)	None
Damped Wiberg [20]	RW2 (Approx. GN) + $\langle \text{Projection const.} \rangle_P + \langle \text{Damping} \rangle$	None
CSF [12]	RW2 (Approx. GN) + $\langle \text{Damping} \rangle$ (DRW2 equiv.)	$q$ -factor ( $U = \text{qorth}(U)$ )
RTRMC [3]	Projected <sub>p</sub> Newton + {Regularization} + $\langle \text{Trust Region} \rangle$	$q$ -factor ( $U = \text{qorth}(U)$ )
LM-S <sub>RW2</sub>	RW2 (Approx. GN) + $\langle \text{Damping} \rangle$ (DRW2 equiv.)	$q$ -factor ( $U = \text{qorth}(U)$ )
LM-M <sub>RW2</sub>	Reduced <sub>r</sub> RW2 (Approx. GN) + $\langle \text{Damping} \rangle$	$q$ -factor ( $U = \text{qorth}(U)$ )
DRW1	RW1 (GN) + $\langle \text{Damping} \rangle$	$q$ -factor ( $U = \text{qorth}(U)$ )
DRW1P	RW1 (GN) + $\langle \text{Projection const.} \rangle_P + \langle \text{Damping} \rangle$	$q$ -factor ( $U = \text{qorth}(U)$ )
DRW2	RW2 (Approx. GN) + $\langle \text{Damping} \rangle$	$q$ -factor ( $U = \text{qorth}(U)$ )
DRW2P	RW2 (Approx. GN) + $\langle \text{Projection const.} \rangle_P + \langle \text{Damping} \rangle$	$q$ -factor ( $U = \text{qorth}(U)$ )

$$\begin{aligned}
\frac{1}{2}H^* = & \mathbf{P}_r^\top (\tilde{\mathbf{V}}^{*\top} (\mathbf{I}_p - [\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\dagger]_{RW2}) \tilde{\mathbf{V}}^* + [\mathbf{K}_{mr}^\top \mathbf{Z}^* (\tilde{\mathbf{U}}^\top \tilde{\mathbf{U}})^{-1} \mathbf{Z}^{*\top} \mathbf{K}_{mr}]_{RW1} \times [-1]_{FN} \\
& + [\mathbf{K}_{mr}^\top \mathbf{Z}^* \tilde{\mathbf{U}}^\dagger \tilde{\mathbf{V}}^* \mathbf{P}_p + \mathbf{P}_p \tilde{\mathbf{V}}^{*\top} \tilde{\mathbf{U}}^\dagger \mathbf{Z}^{*\top} \mathbf{K}_{mr}]_{FN} + \langle \alpha \mathbf{I}_r \otimes \mathbf{U} \mathbf{U}^\top \rangle_P + \langle \lambda \mathbf{I}_{mr} \rangle \mathbf{P}_r
\end{aligned}$$

# 1. Unified derivation of methods

•	ALS
•	VH_PF [8]
•	TW_WB [11]
•	TO_DW [9]
•	DRW1
•	DRW1P
•	DRW2
•	DRW2P
•	PG_CSF [7]
•	CH_LM_S [5]
•	CH_LM_S_GN [5]
•	CH_LM_S_RW2
•	CH_LM_M [5]
•	CH_LM_M_GN [5]
•	CH_LM_M_RW2
•	NB_RTRMC [2]

**inputs:** M, W, r, U,  $\lambda_0$

$\lambda \leftarrow \lambda_0$

$\tilde{W} \leftarrow \text{diag}(\text{vec}(W))$  with zero-rows removed.

$\tilde{\mathbf{m}} \leftarrow \tilde{W} \text{vec}(M)$

**repeat**

$\mathbf{g} \leftarrow V^*(U)^\top \text{vec}(UV^*(U)^\top - M)$  // for all algorithms listed here.

1:  $H \leftarrow V^{*\top} V^*$

2:  $H \leftarrow H - \tilde{V}^{*\top} \tilde{U} \tilde{U}^\top \tilde{V}^*$

3:  $H \leftarrow H + K_{mr}^\top Z^* (\tilde{U}^\top \tilde{U})^{-1} Z^{*\top} K_{mr}$

4:  $H \leftarrow H - K_{mr}^\top Z^* (\tilde{U}^\top \tilde{U})^{-1} Z^{*\top} K_{mr} + K_{mr}^\top Z^* \tilde{U}^\top \tilde{V}^* + \tilde{V}^{*\top} \tilde{U}^\top Z^{*\top} K_{mr}$

5:  $P \leftarrow I \otimes (I - UU^\top) \in \mathbb{R}^{mr \times mr}$  // (5) is also a no-operation if 7 and 8 are.

6:  $P \leftarrow I \otimes U_\perp^\top \in \mathbb{R}^{(m-r)r \times mr}$

7:  $\mathbf{g} \leftarrow P\mathbf{g}$  // (7) is a no-operation since  $\mathbf{g} = P\mathbf{g}$ .

8:  $H \leftarrow PHP^\top$  // (8) is a no-operation since  $H = PH$ .

9:  $H \leftarrow H + I \otimes UU^\top$  // relaxed constraint to promote  $U^\top \Delta U = 0$ .

10: **repeat**

11:  $\Delta U \leftarrow \text{unvec}(H^{-1} \mathbf{g})$

12:  $\Delta U \leftarrow \text{unvec}((H + \lambda I)^{-1} \mathbf{g})$

13:  $\lambda \leftarrow \lambda * 10$

14: **until**  $f(U + \Delta U, V^*(U + \Delta U)) < f(U, V^*(U))$

15:  $U \leftarrow U + \Delta U$

16:  $U \leftarrow qf(U)$  //  $[U, \sim] = \text{qr}(U, 0)$  in MATLAB.

17:  $V \leftarrow \text{unvec}(\tilde{U}^\top \tilde{\mathbf{m}})$

18:  $\lambda \leftarrow \lambda / 100$

**until** convergence

**outputs:** U, V

- Which is just matrix completion:

$$\min_{U \in \mathbb{R}^{M \times K}} \min_{V \in \mathbb{R}^{N \times K}} \|M - UV^T\|^2$$

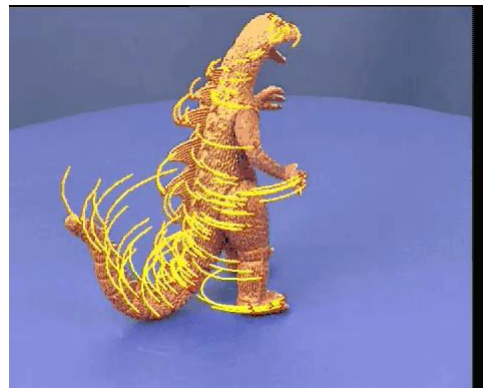
$\nwarrow$   $W \otimes (M - UV^T)$

- We all know that given  $U$ ,

$$V^*(U) = (U^\dagger M)^T$$

this is how you get alternation/ICP:

1.  $U = U^*(V);$
2.  $V = V^*(U);$



Success rate: 0%

$\Rightarrow$  ~~RTD~~

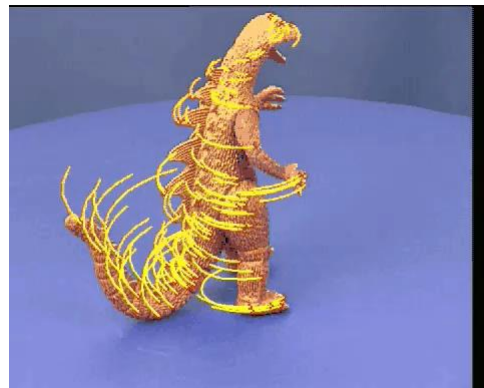
- Matrix completion:

$$\min_{U \in \mathbb{R}^{M \times K}} \min_{V \in \mathbb{R}^{N \times K}} \|M - UV^T\|^2$$

$$V^*(U) = (U^\dagger M)^T$$

- ... so “unlift”:

$$\min_{U \in \mathbb{R}^{M \times K}} \|M - UU^\dagger M\|^2$$



Success rate: ? %



- Matrix completion:

$$\min_{U \in \mathbb{R}^{M \times K}} \min_{V \in \mathbb{R}^{N \times K}} \|M - UV^T\|^2$$

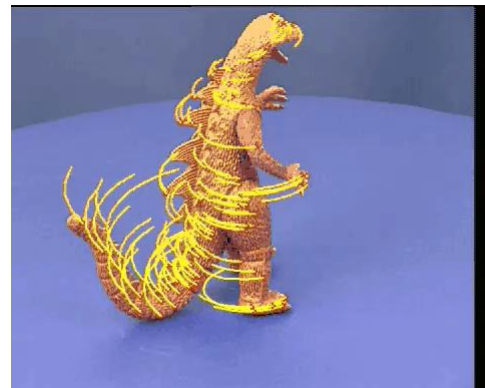
$$V^*(U) = (U^\dagger M)^T$$

- ... so “unlift”:

$$\min_{U \in \mathbb{R}^{M \times K}} \|M - UU^\dagger M\|^2$$

- Compute gradient:

$$\frac{\partial A^\dagger}{\partial x} = -A^\dagger \frac{\partial A}{\partial x} A^\dagger + \dots \text{ (see supmat)}$$



Success rate: 50%

This is [Wiberg '73]  
or “VarProGN” [Ruhe  
& Wedin '84]

- Matrix completion:

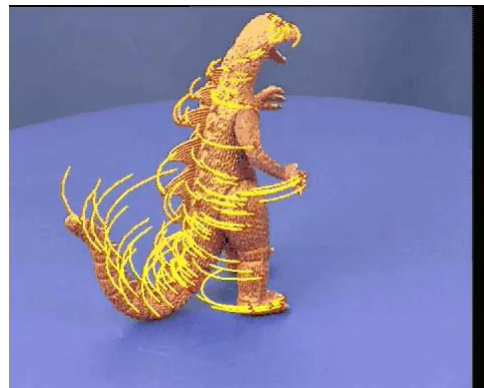
$$\min_{U \in \mathbb{R}^{M \times K}} \min_{V \in \mathbb{R}^{N \times K}} \|M - UV^T\|^2$$

$$V^*(U) = (U^\dagger M)^T$$

- ... so “unlift”:

$$\min_{U \in \mathbb{R}^{M \times K}} \|M - UU^\dagger M\|^2$$

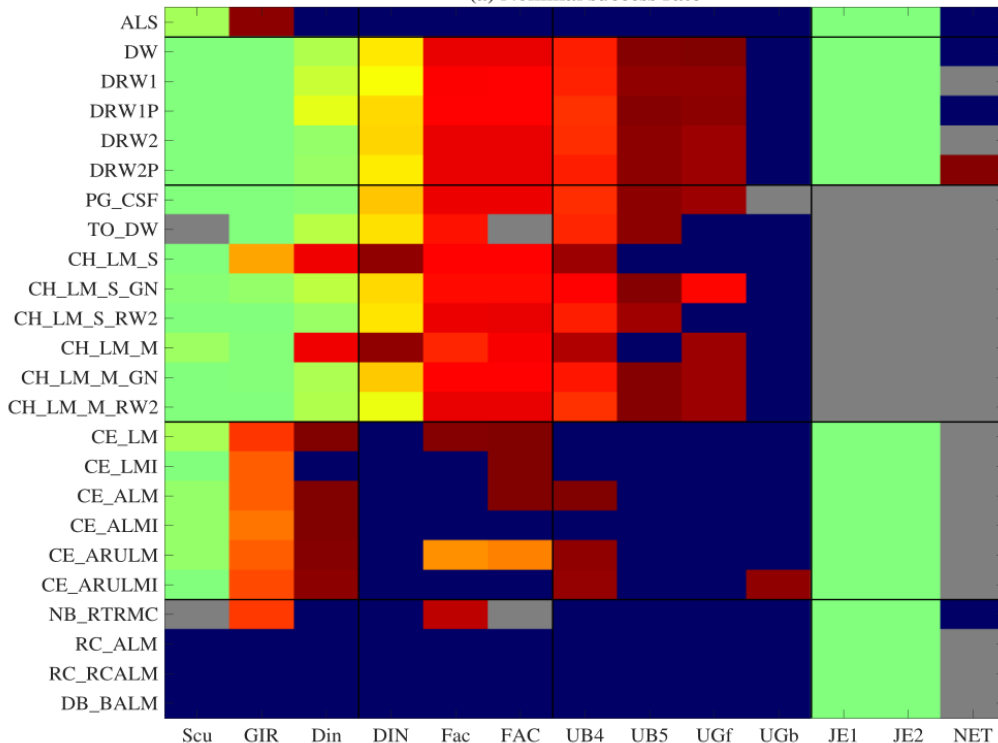
- And move to Levenberg-Marquardt instead of GN, and fix some other stuff  
(add priors etc)



Success rate: 97%

[Okatani '11, Hong & Fitzgibbon '15]

(a) Nominal success rate



13 standard datasets  
24 algorithms

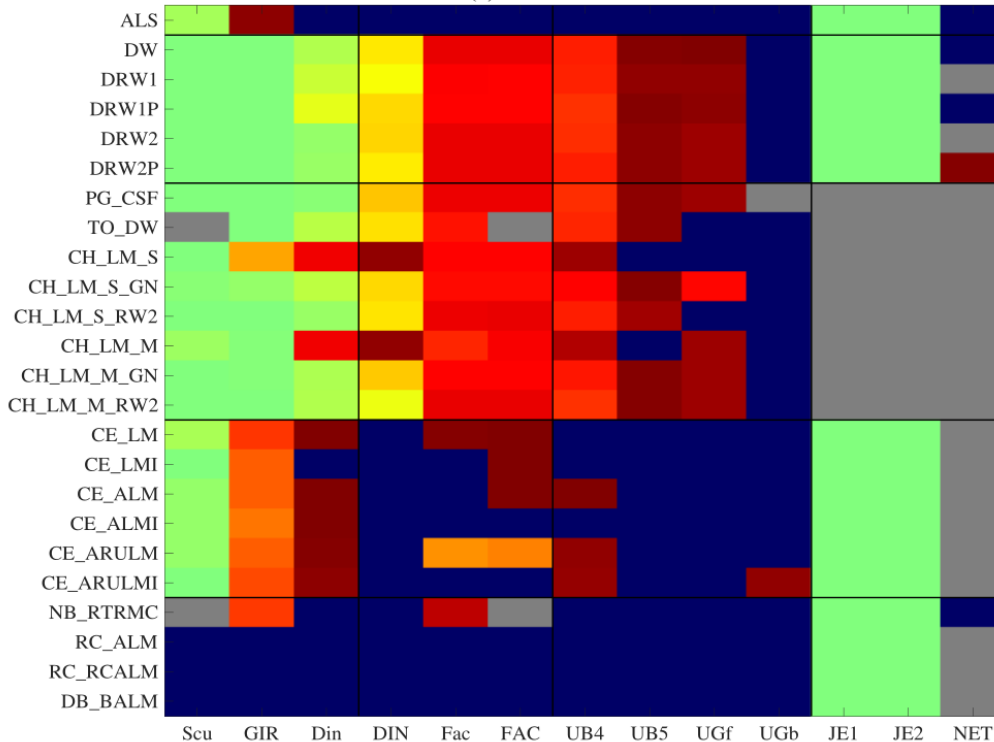
From how many of 100 random starting points does algorithm  $X$  reach the best known optimum?

Green=100

Blue =0

Grey=timeout

(a) Nominal success rate

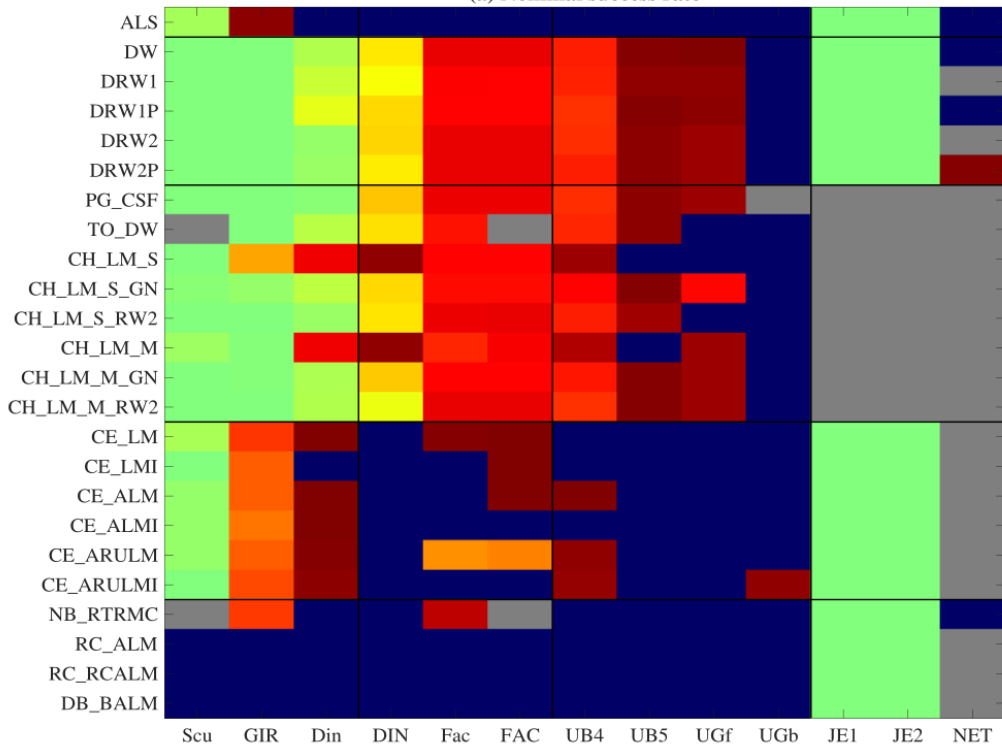


A “new” algorithm: RUSSO- $X$

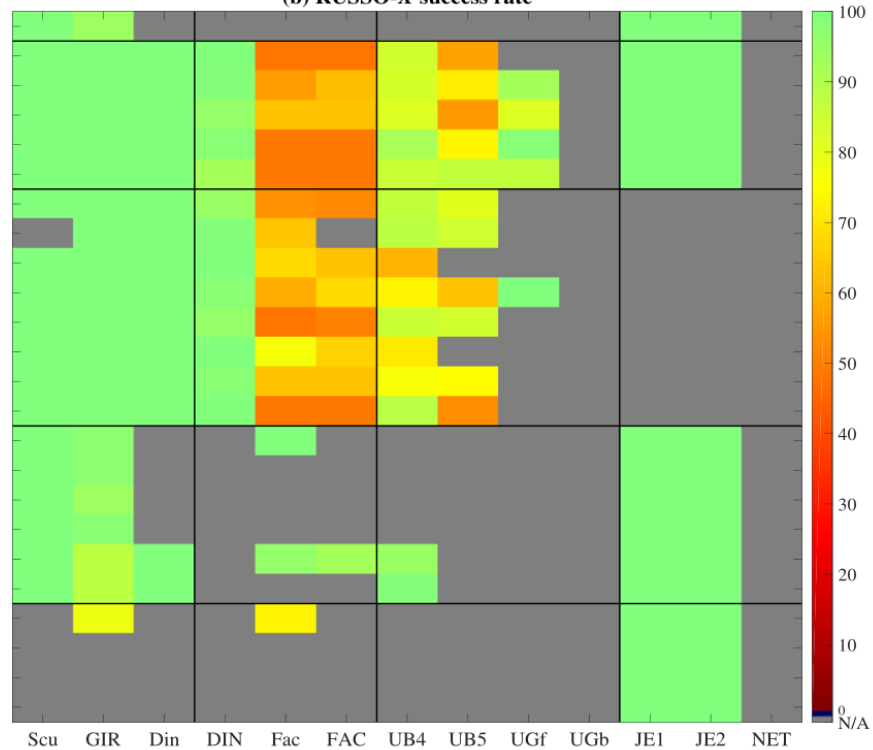
Run algorithm  $X$  from random starting points until you see the same optimum twice.

```
while true
    fmin := X(M, rand(...))
    if fmin < best_so_far
        best_so_far := fmin
    else if fmin = best_so_far
        success!
end
```

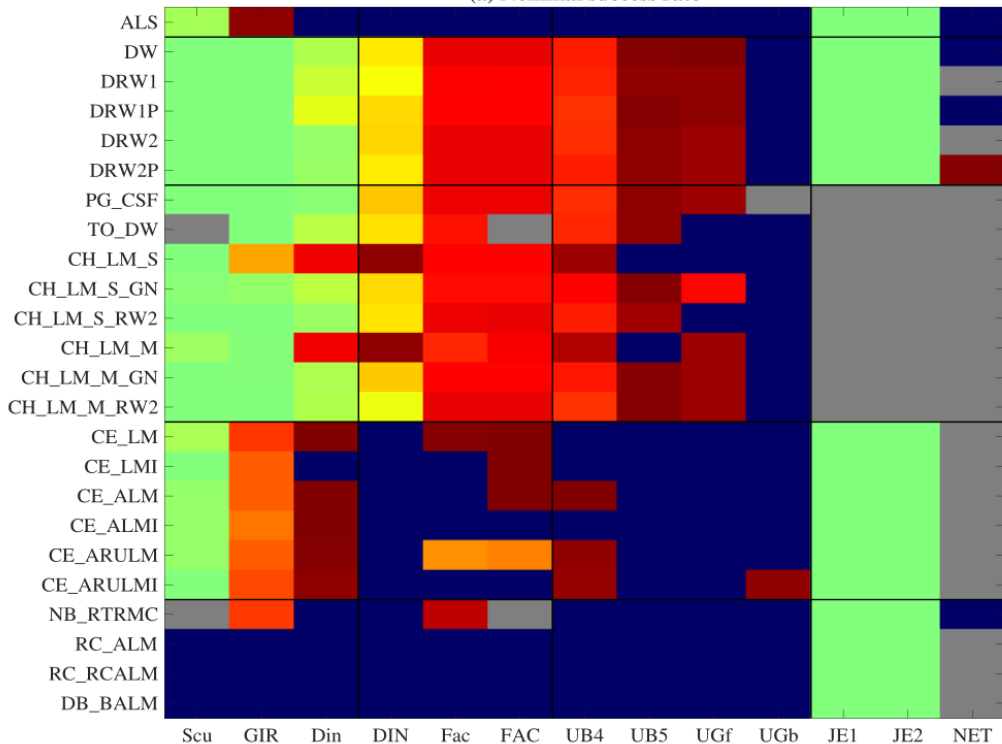
(a) Nominal success rate



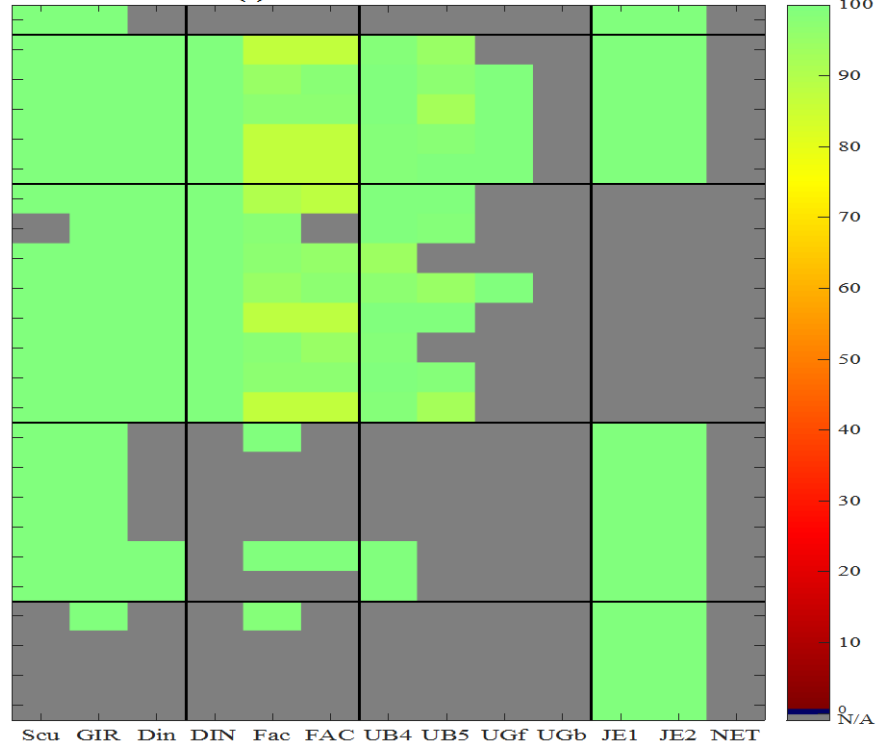
(b) RUSSO-X success rate

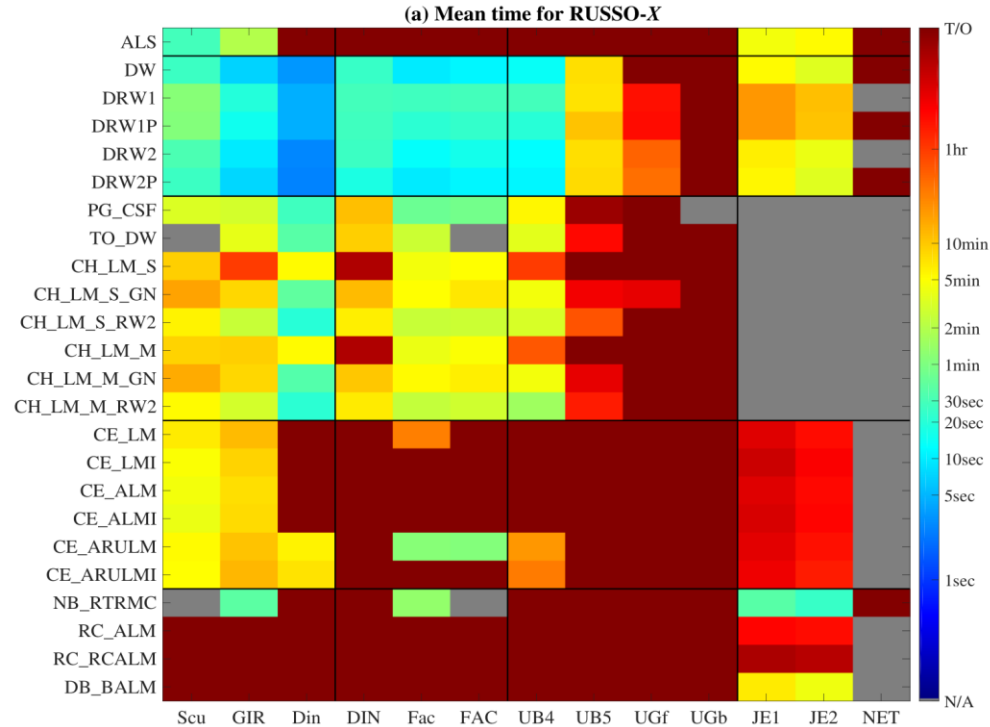
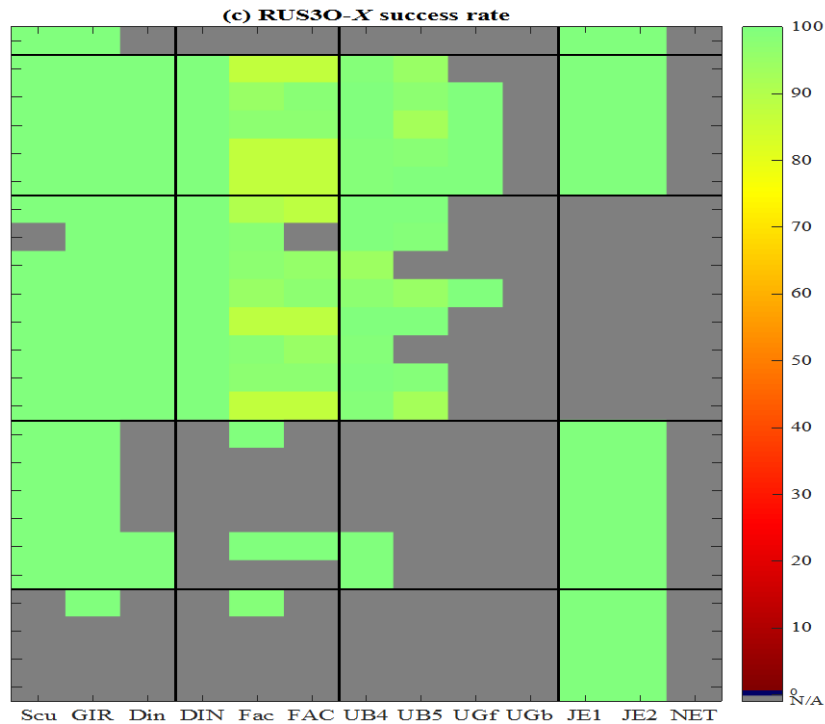


(a) Nominal success rate



(c) RUS3O-X success rate





So speed is all that matters. How can we measure it fairly?  
 Re-implement everything, faster and more accurate than original authors.

# Lifting and VarPro are almost exact opposites

- In much of my previous work, lifting helped
- In [Zach, ECCV '14], VarPro was drastically worse
- In [Okatani '11] VarPro is dramatically better

Everything else is detail: numerics, manifold projection, Hessian approximations, ...



- Using subdvs is easy
  - The messy stuff is encapsulated in `Eval_M*()`, and `Plus()`
  - Google's "Ceres" solver does all the Levenberg-Marquardt
- Continuous optimization often doesn't need a very good initial estimate
- Using subdvs allows correspondences  $\mathbf{u}_i$  to update *during* the optimization
  - If ICP takes a long time, this may not...
  - But you **must** exploit sparsity
- Future work:
  - Dogs, hinted ARAP, skeleton, even more speed, ...

- Seen a few students nastily bitten by collapsing meshes
- So what's changed? How do I get bitten by the bug, not the hornet?
  1. Sum over data, not model
  2. Use modern (2006) regularizers
  3. Vary everything
  4. Define clean interpolants

- CLOSED FORM" SOLUTIONS OFTEN SOLVE A NEARBY CONVEX PROBLEM.
- SO DOES ANY 2<sup>ND</sup> ORDER OPTIMIZER.
- IF YOU HAVE FOUND A QUADRATIC SUBPROBLEM, SO WILL LEVENBERG-MARQ.
- YOU CAN DIFFERENTIATE THROUGH PRETTY MUCH ANYTHING.
- SCALING IS IMPORTANT. MEASURE IN NATURAL UNITS.

- Finite diffs fine, just expensive
- Myths: you don't need to find the optimum
- Parameter tuning
- Constrained optimization