Minimization Engines

Max-Planck-Institut
für Radioastronomie

Brian Kloppenborg (bkloppen@mpifr.de)
2013-07-23 – MPIfR Group meeting

Overview

- The theory (pretty pictures and lots of math)
  - Traditional minimization methods
  - Stochastic exploration methods
  - … and bounding conditions mixed in somewhere
- Specific engines and example applications
  - Super simple, but poor performance
  - Good methods that can go bad
  - Need to find the needle? Burn the haystack! (And sift through the ashes.)
- Uncertainties

“Traditional” minimization

Gradient descent

As simple as it gets, just go downhill

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \underset{\substack{\uparrow \\ \text{Step size}}}{\gamma_n} \overset{\substack{\text{Take the gradient} \\ \downarrow}}{\nabla} F(\mathbf{x}_n),\ n \geq 0.$$

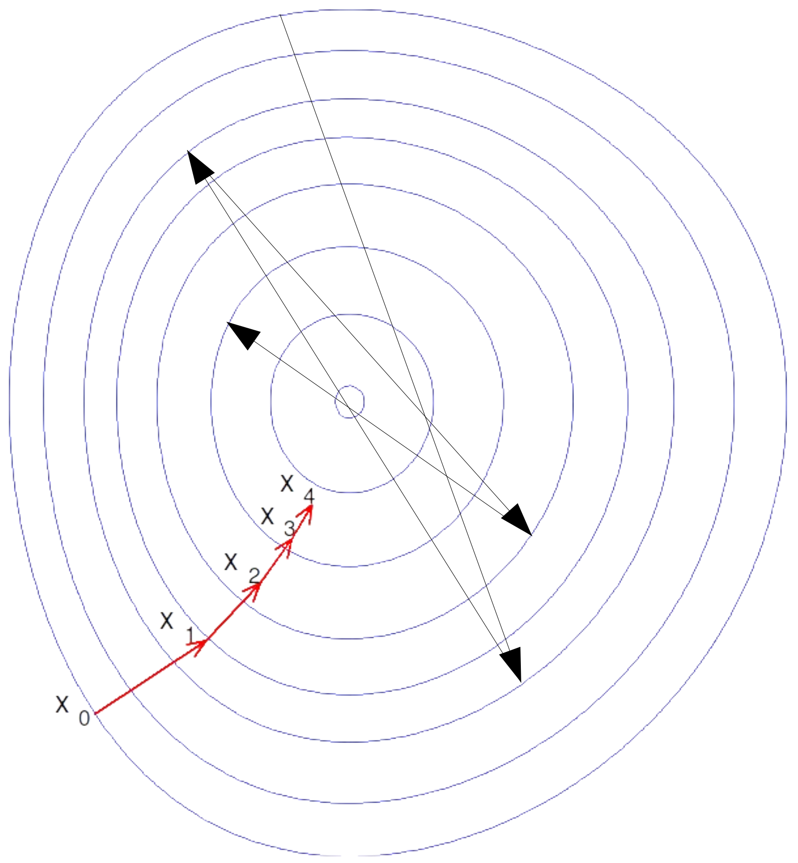


Image from Wikimedia commons “gradient descent”

Newton’s method

An improvement on the gradient descent method

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \underset{\text{Hessian}}{[H f(\mathbf{x}_n)]^{-1}} \underset{\text{Gradient}}{\nabla f(\mathbf{x}_n)}, \ n \geq 0.$$

The Hessian is the Jacobian of the gradient of F
… a.k.a. all second partial derivatives of F

$$H(f) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

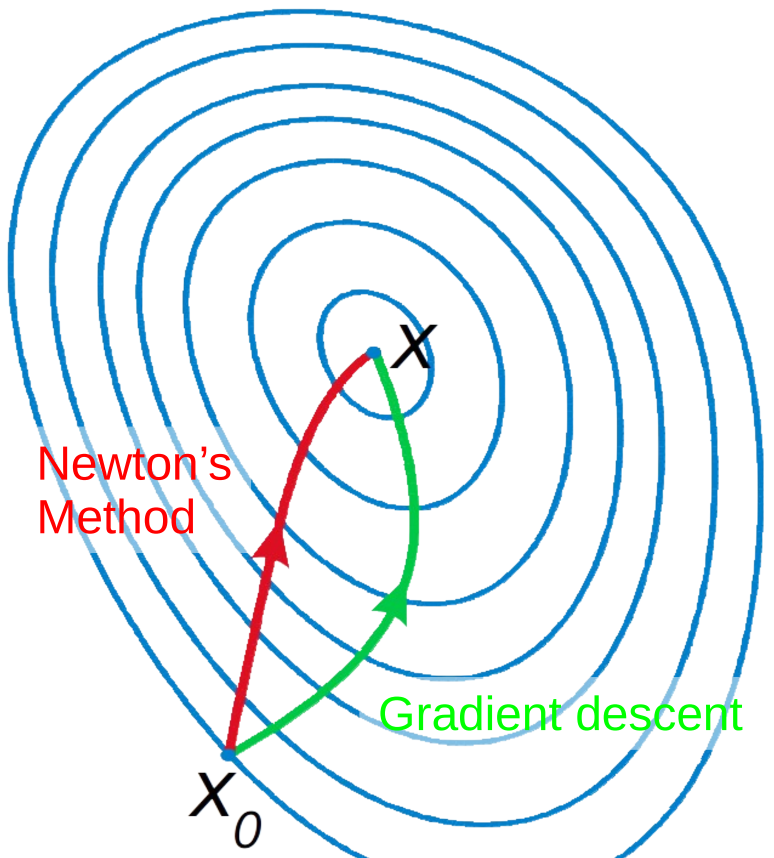


Image from Wikimedia commons “Newton optimization”

Gauss-Newton / Quasi-Newton

Don’t know / want to compute the Hessian? No problem, but...

Gauss-Newton:

- Only applicable for least-squared minimization problems

$$\mathbf{x}_{n+1} = \mathbf{x}_n - (J^T \overset{\text{Jacobian}}{\overset{\downarrow}{J}})^{-1} J^T \overset{\text{Residual function}}{\overset{\downarrow}{r}}(\mathbf{x}_n),\ n \geq 0.$$

Quasi-Newton

- Various algorithms
- Analyze success applications of gradient → ~ Hessian
- Often already implemented, so use a library. E.g.:
  - SR1
  - BHHH
  - BFGS/L-BFGS

Levenberg-Marquardt

A “damped” least-squares method... it’s all about the multiplier

Finds the (closest) local minimum, not necessarily the global minimum

Recall Gradient descent

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta_n \nabla F(\mathbf{x}_n),\ n \geq 0.$$

Multiplier found via:

$$(J^T J)\delta = J^T r(\mathbf{x})$$

In the LM algorithm, the descent id damped by lambda:

$$(J^T J + \lambda I)\delta = J^T r(\mathbf{x})$$

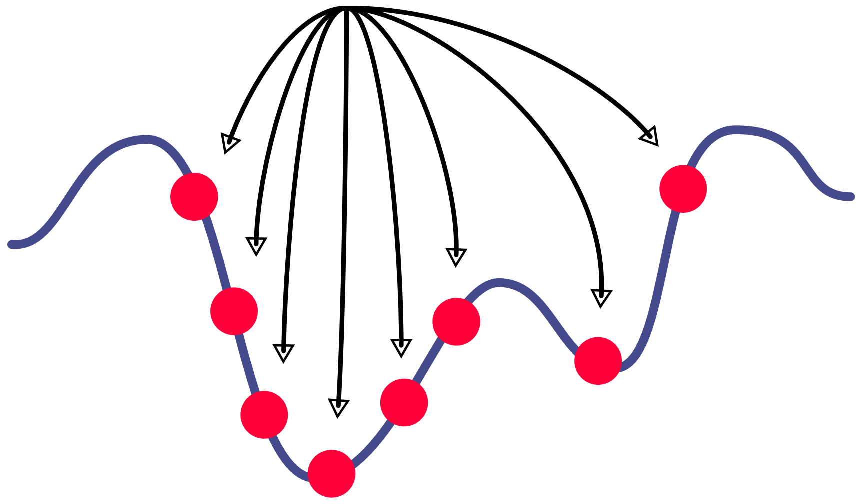
Or, more commonly:

$$(J^T J + \lambda \mathrm{diag}(J^T J))\delta = J^T r(\mathbf{x})$$

Stochastic Minimization

Random search

```
while search_not_done:
    x = random()
    if f(x) < f(x_best):
        x_best = x
```



Some variants apply bounding conditions on the step size:

- Fixed Step Size Random Search (FSSRS)
- Optimum Step Size Random Search (OSSRS)
- Adaptive Step Size Random Search (ASSRS)
- Optimized Relative Step Size Random Search (ORSSRS)

Graphic from Wikimedia commons (fr) “metaeuristics”

Metropolis-Hastings algorithm

Classic MCMC

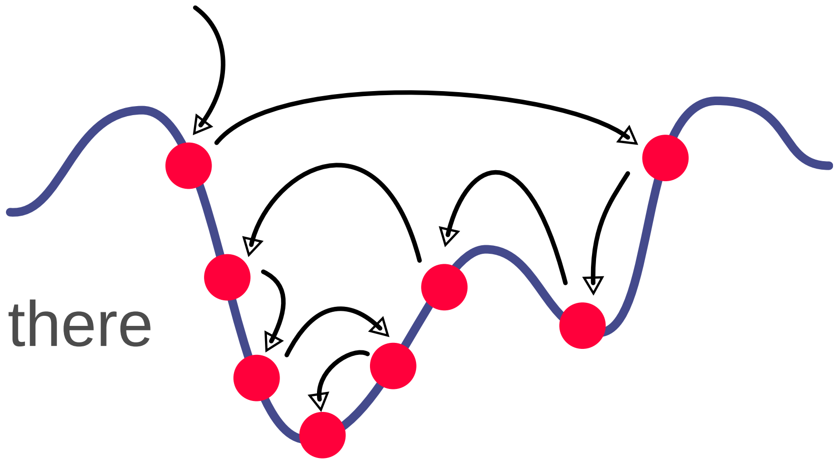
- Start at some random location in the parameter space
- Propose a move from the current position

$$\alpha(\mathbf{x}_1, \mathbf{x}_2) = \begin{cases} \min\left[\frac{\pi(\mathbf{x}_2)q(\mathbf{x}_2,\mathbf{x}_1)}{\pi(\mathbf{x}_1)q(\mathbf{x}_1,\mathbf{x}_2)}, 1\right], & \text{if } \pi(\mathbf{x}_1)q(\mathbf{x}_1,\mathbf{x}_2) > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Transition kernel enforces reversibility of moves

i.e.

- if the move is worse, you **might** go there
- if the move is better, go there

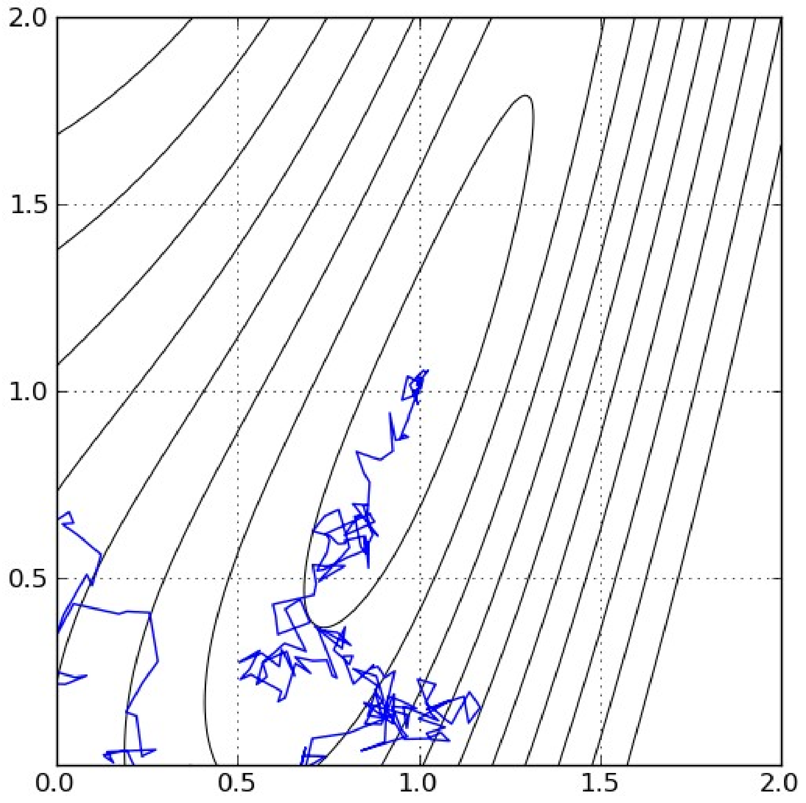


Graphic from Wikimedia commons (fr) “metaeuristics”

Simulated annealing

A modification to the Metropolis-Hastings algorithm

- Slowly decrease the probability of accepting a worse solution
- Requires an annealing schedule to be defined

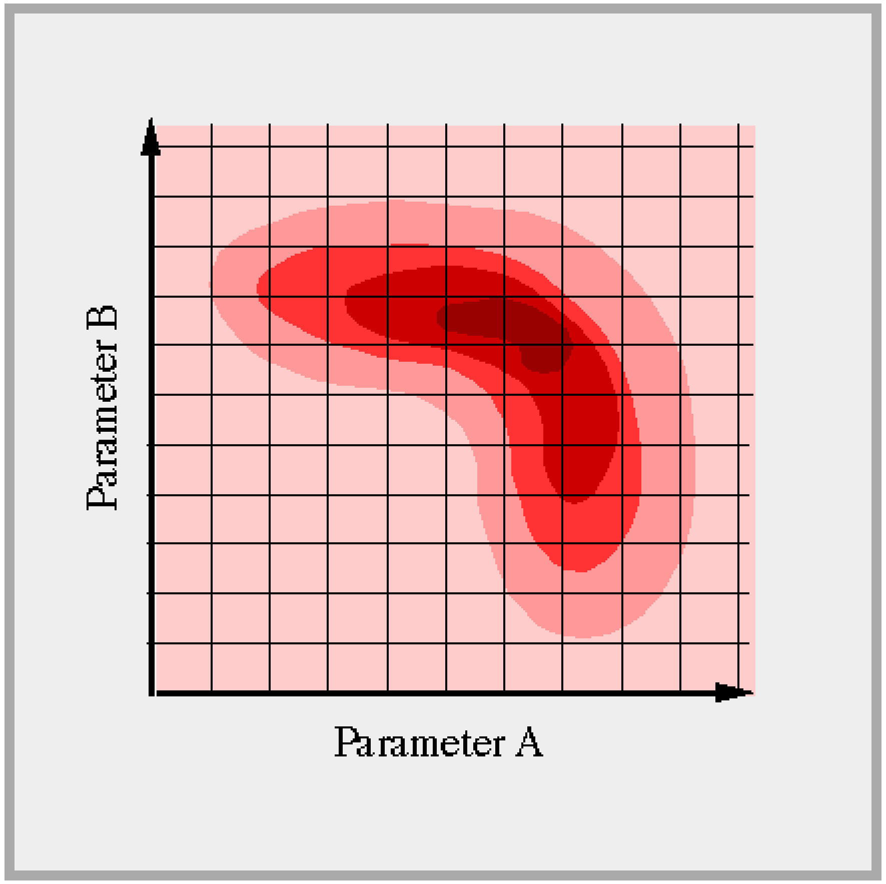


Graphic from Peach (AI library for Python, http://peach.googlecode.com)

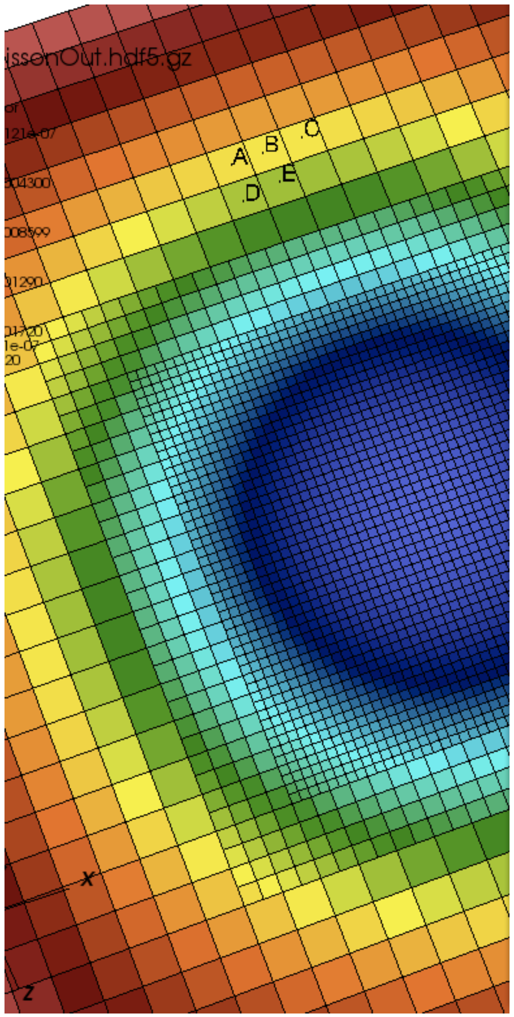
Engines

Simple methods

Gridsearch



Adaptive Mesh Refinement (AMR)



Images from iTough2 website (http://esd/lbl.gov/itough2)

MPFIT

Least-squares (Levenberg-Marquardt)

- Languages: IDL, C
- You define:
  - User function to compute the residuals
  - Minimizer stopping/step parameters
- Good
  - Permits fixed/linked variables
  - Explicit or numerical derivatives
- Bad
  - Recursion explicitly prohibited
  - Very sensitive to user-defined parameters.

Levmar

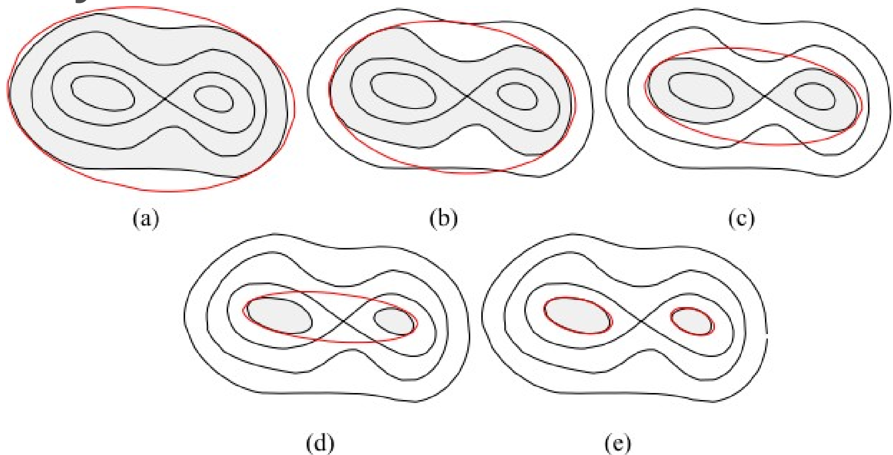
Least-squares (Levenberg-Marquardt)

- Languages: C
- You define:
  - User function to compute the residuals
  - Several parameters
- Good
  - Explicit or numerical derivatives
  - Supports both constrained and unconstrained problems.

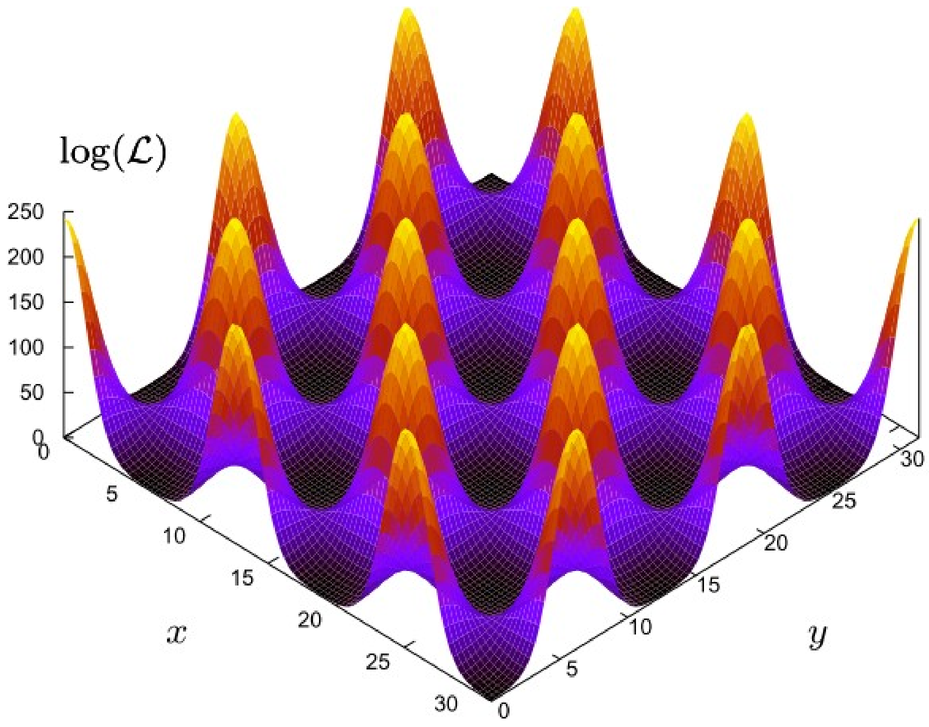
MultiNest

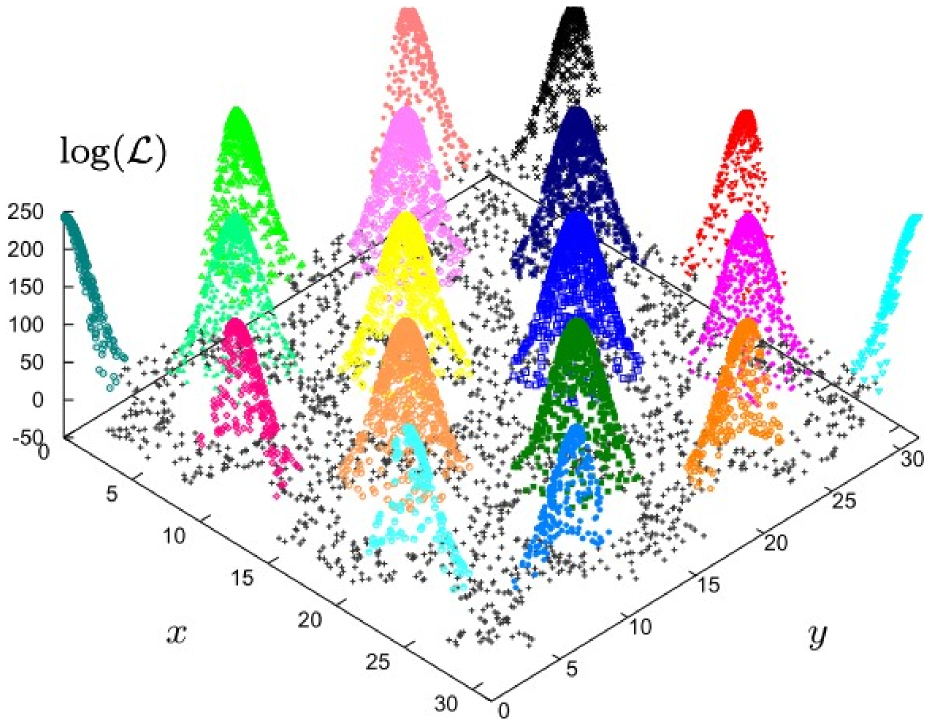
Bayesian nested sampling

- Languages: Fortran, C/C++
- You define:
  - User function to compute the Bayesian evidence
  - A few stopping parameters
  - Bounds on all parameters
- Good
  - Permits wrapped variables
  - Computes best-fit values along with Bayesian evidence
  - Works very well in high-dimensions
  - Ellipsoidal bounding conditions
- Bad
  - Highly computationally intensive



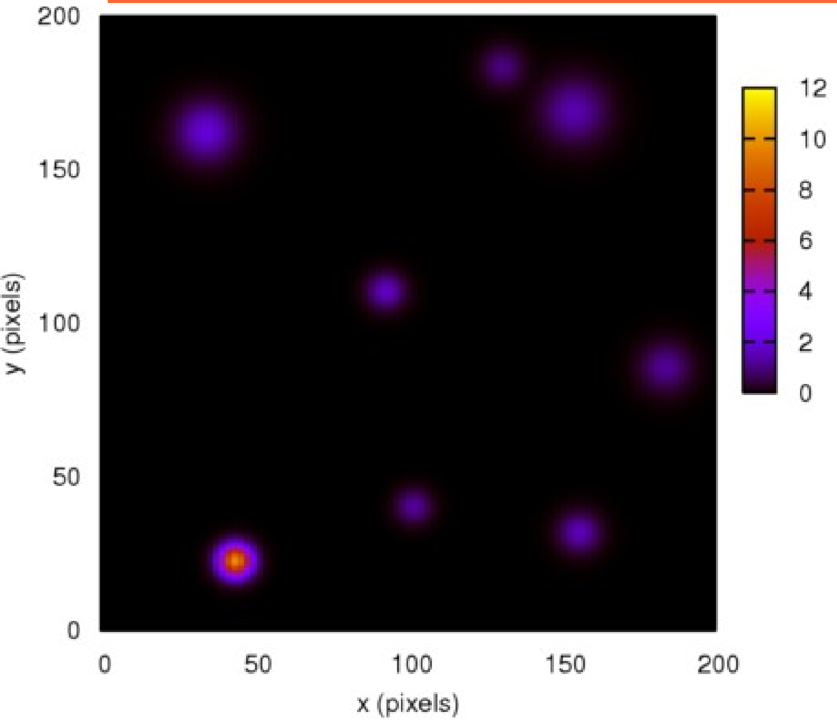
Multi-modal example

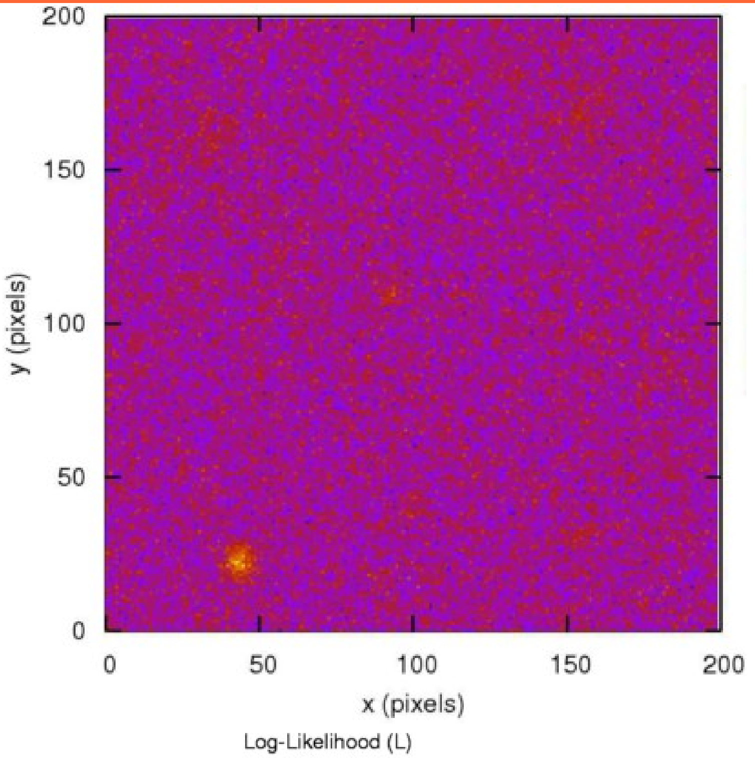


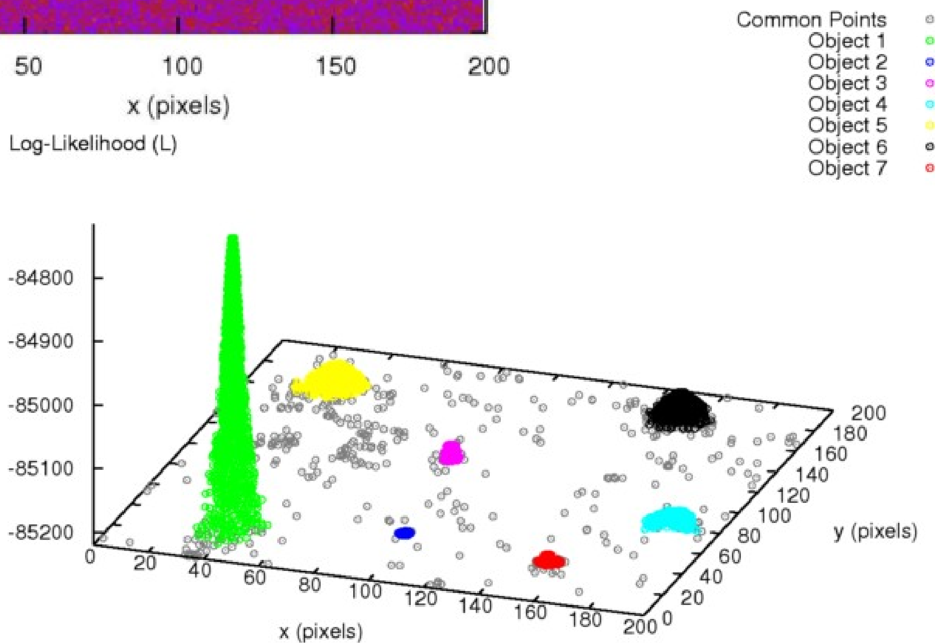


Graphics from Feroz et al. 2008

Multinest application: Star detection







Graphics from Feroz et al. 2008

Uncertainties

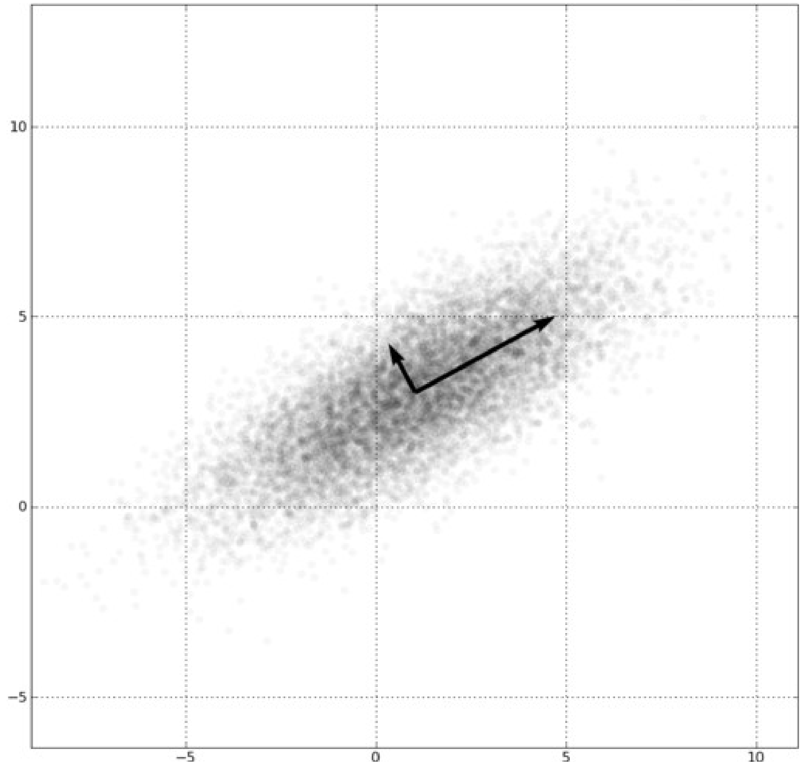
How do engines derive uncertainties?

$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1-\mu_1)(X_1-\mu_1)] & \mathrm{E}[(X_1-\mu_1)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_1-\mu_1)(X_n-\mu_n)] \\ \mathrm{E}[(X_2-\mu_2)(X_1-\mu_1)] & \mathrm{E}[(X_2-\mu_2)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_2-\mu_2)(X_n-\mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n-\mu_n)(X_1-\mu_1)] & \mathrm{E}[(X_n-\mu_n)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_n-\mu_n)(X_n-\mu_n)] \end{bmatrix}.$$

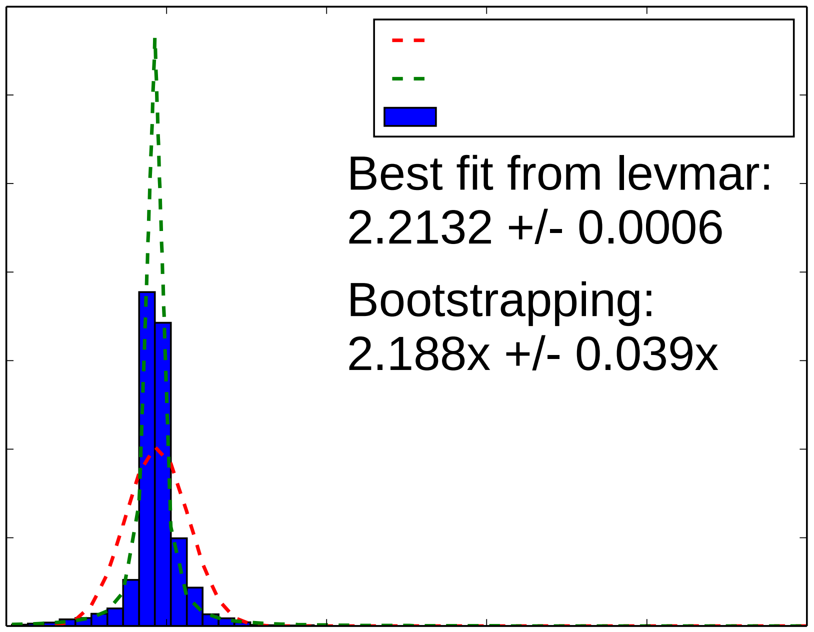
Most engines ignore covariance when reporting uncertainties

$$\begin{aligned} \sigma^2 &= \mathrm{var}(X_i) \\ &= \mathrm{E}[(X_i - \mathrm{E}(X_i))^2] \\ &= \mathrm{E}[(X_i - \mathrm{E}(X_i)) \cdot (X_i - \mathrm{E}(X_i))] \end{aligned}$$

And the reported uncertainties do not propagate any systematic errors in the data.

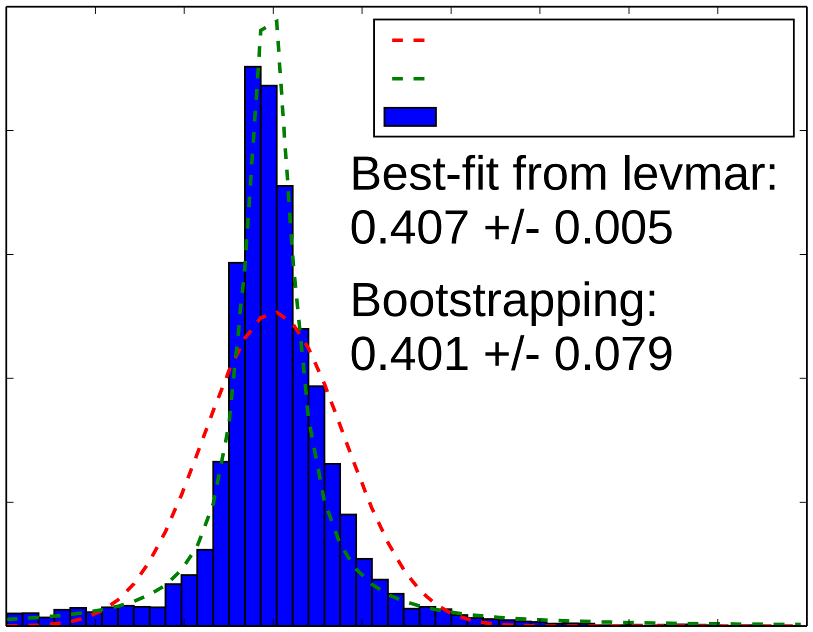


Most engines significantly underestimate uncertaintes



Best fit from levmar:
2.2132 +/- 0.0006

Bootstrapping:
2.188x +/- 0.039x

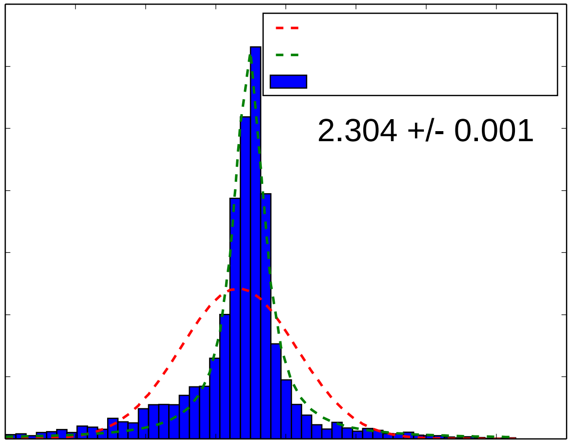


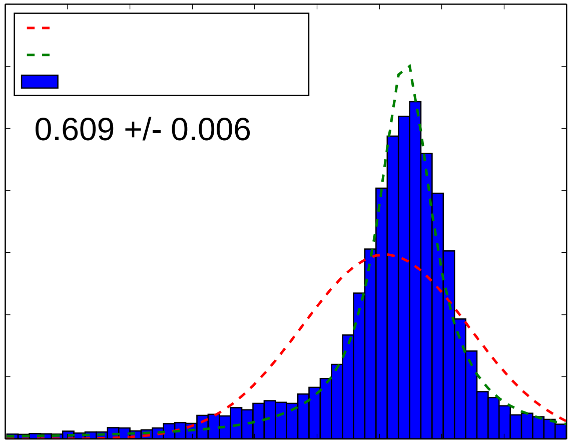
Best-fit from levmar:
0.407 +/- 0.005

Bootstrapping:
0.401 +/- 0.079

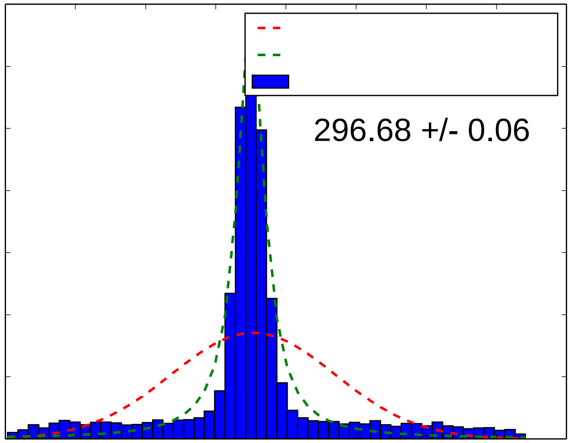
Symmetric error bars are probably not right for optical interferometry

Stellar Parameters

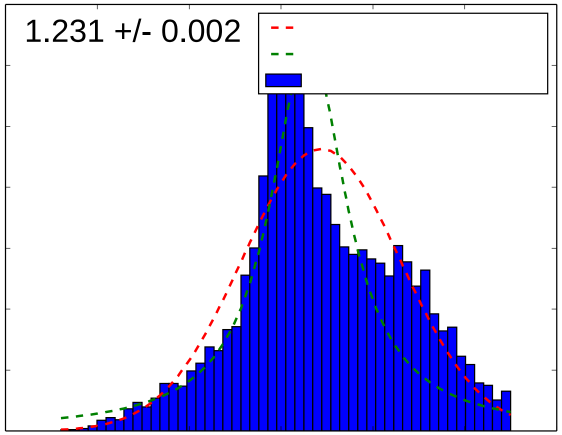


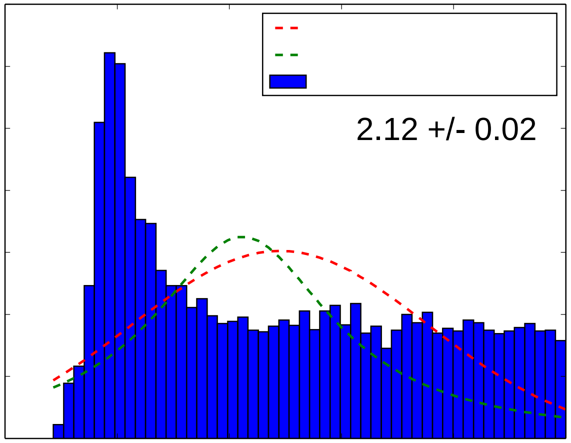


Orbital parameter



Disk Parameters





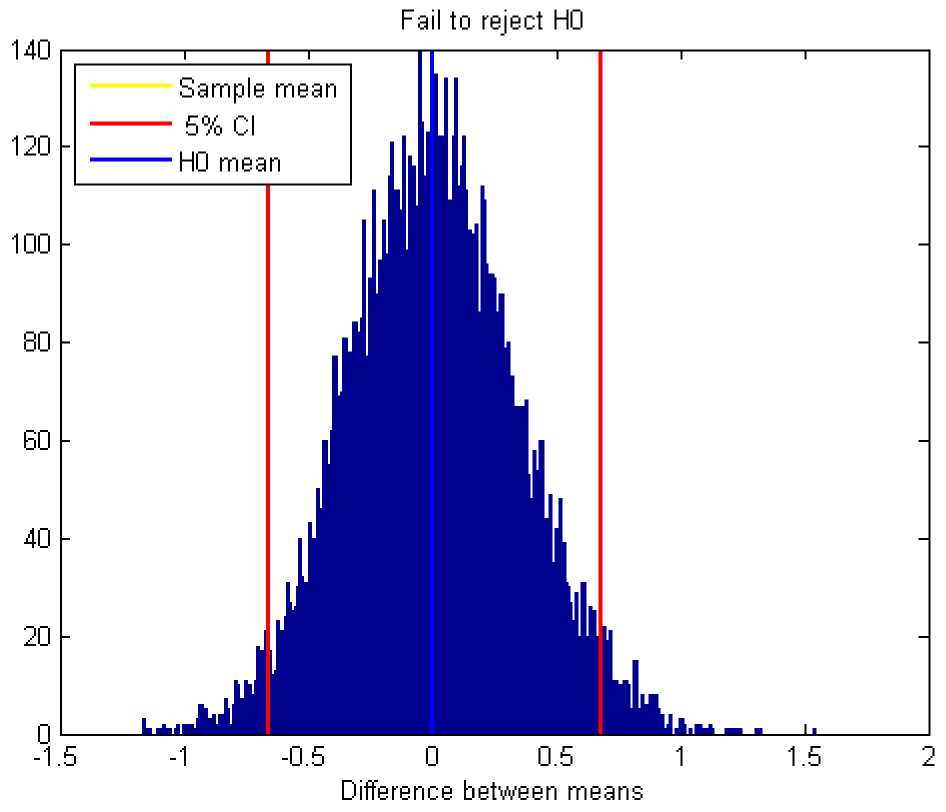
Bootstrapping

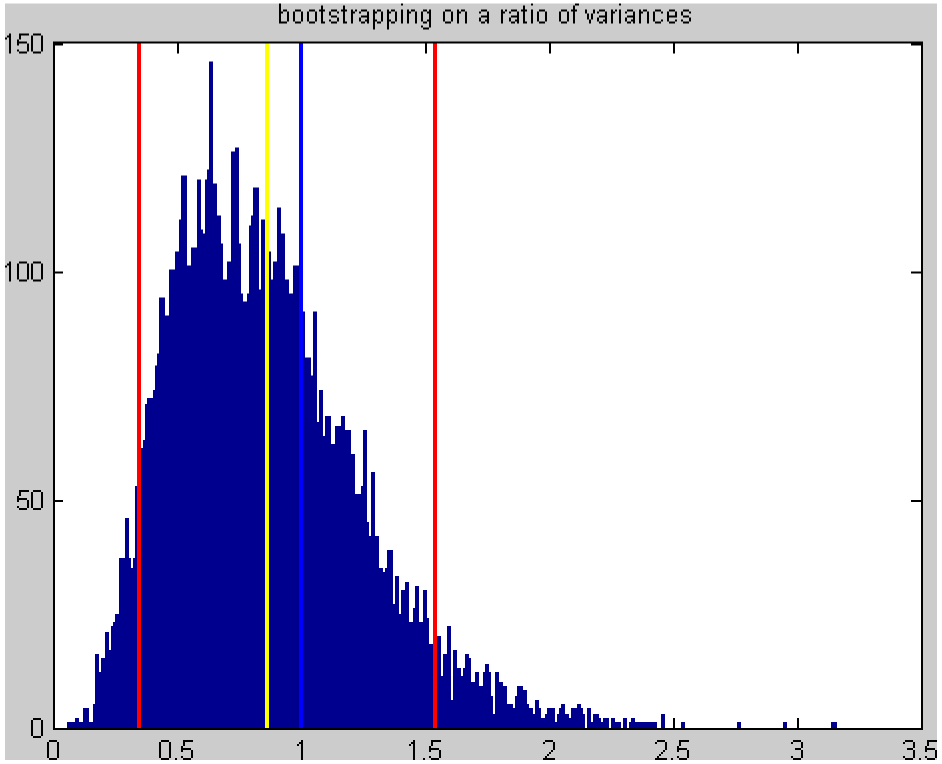
Better uncertainty estimates

- Derive uncertainties for an unknown distribution

- Procedure:
  - Fit real data once, store best-fit values
  - Repeat many (1k-10k) times:
    - Select N data at random from initial data set (of size N)
      - Take into account any correlations in the data
      - Repeated values are expected
    - Redistribute nominal values
    - Fit data, store best-fit results
  - Create a histogram of best-fit values
  - Derive uncertainties from this distribution

Bootstrapping example





Graphics from http://courses.washington.edu/matlab1/Bootstrap_examples.html