New developments in statistical methods

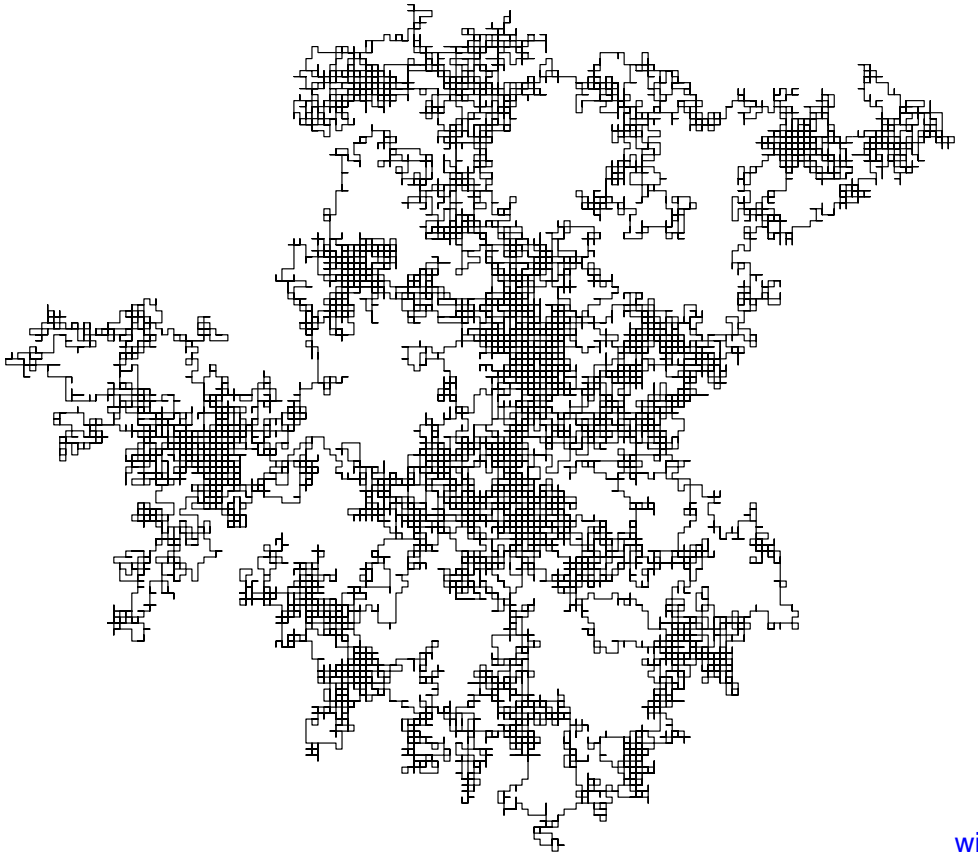
Frederik Beaujean
C2PAP, Universe Cluster, LMU Munich

IWHSS 2016

Getting home

restnova.com

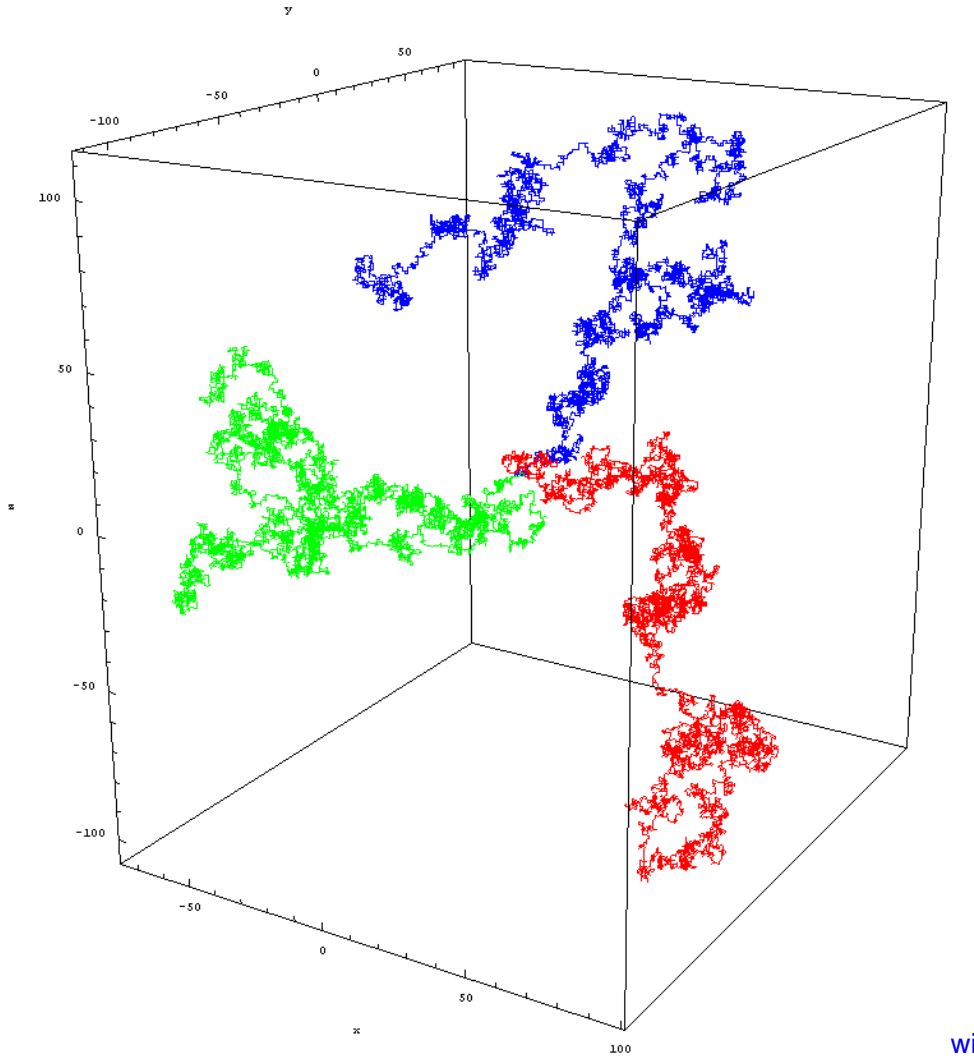
Getting home



wikipedia

symmetric random walk $P(\text{return}|1D) = P(\text{return}|2D) = 1$

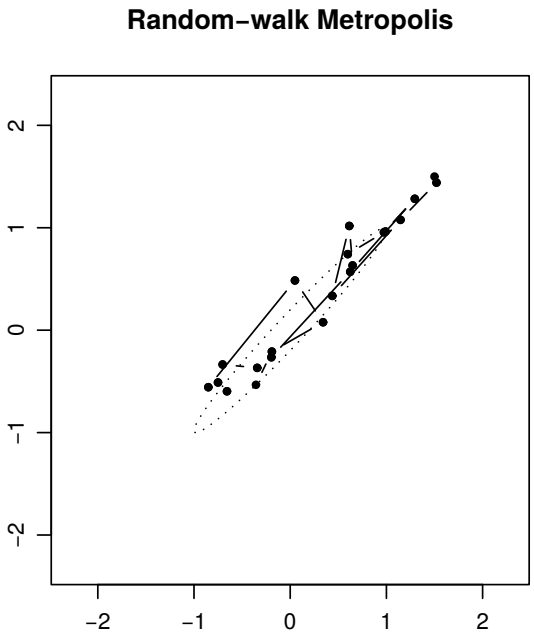
Getting home



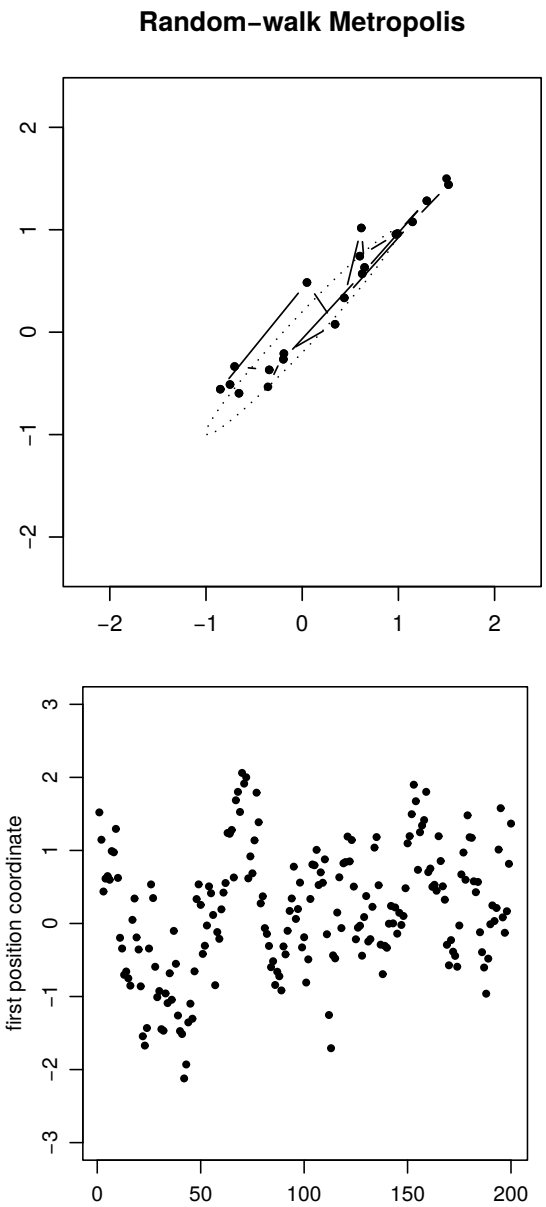
wikipedia

symmetric random walk $P(\text{return}|1D) = P(\text{return}|2D) = 1$
but $P(\text{return}|n > 3) < 1$

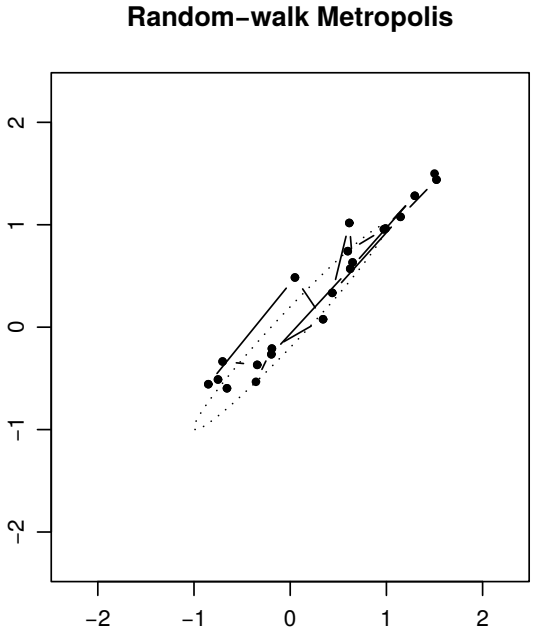
Random walk in 2D

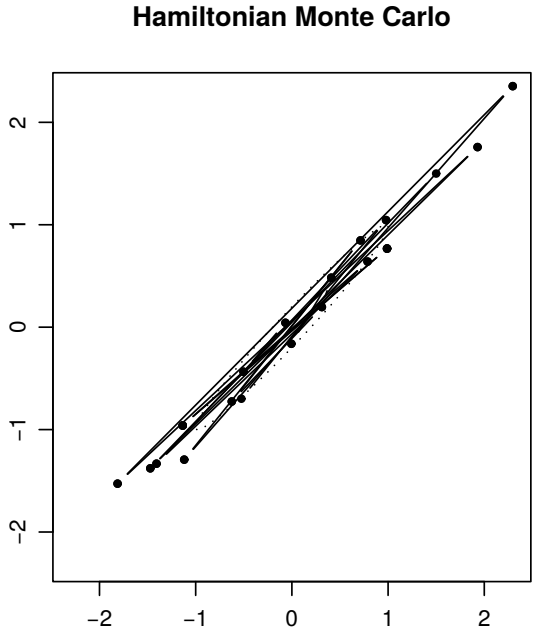


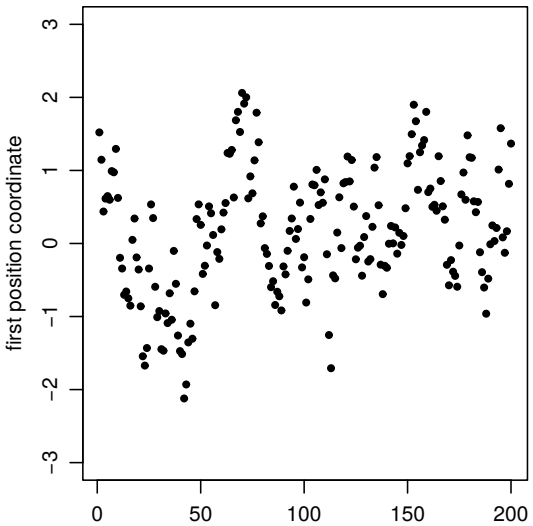
Random walk in 2D



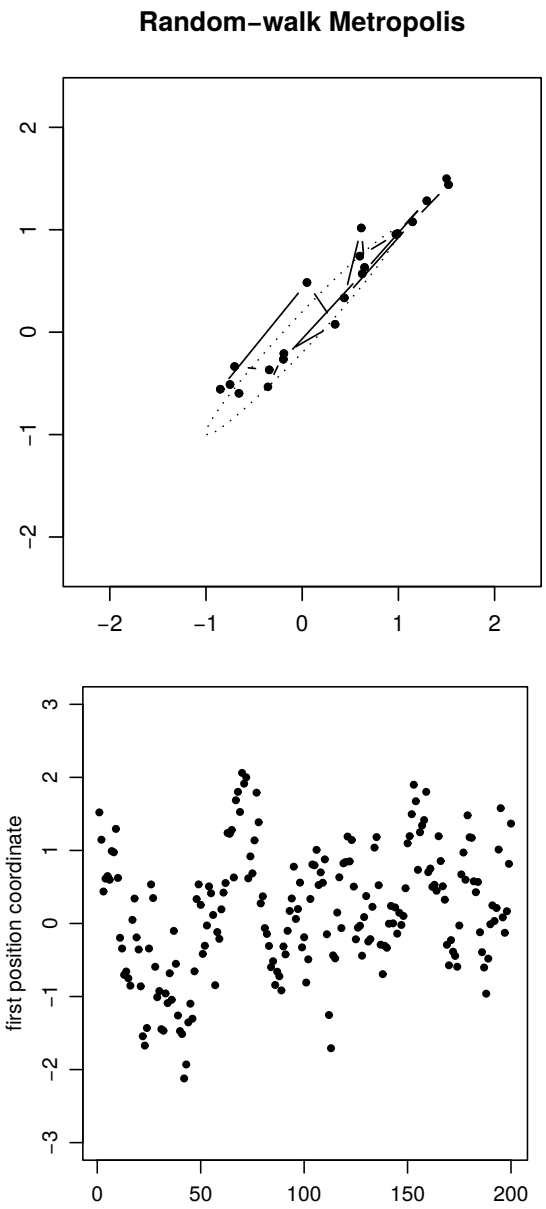
Random walk in 2D

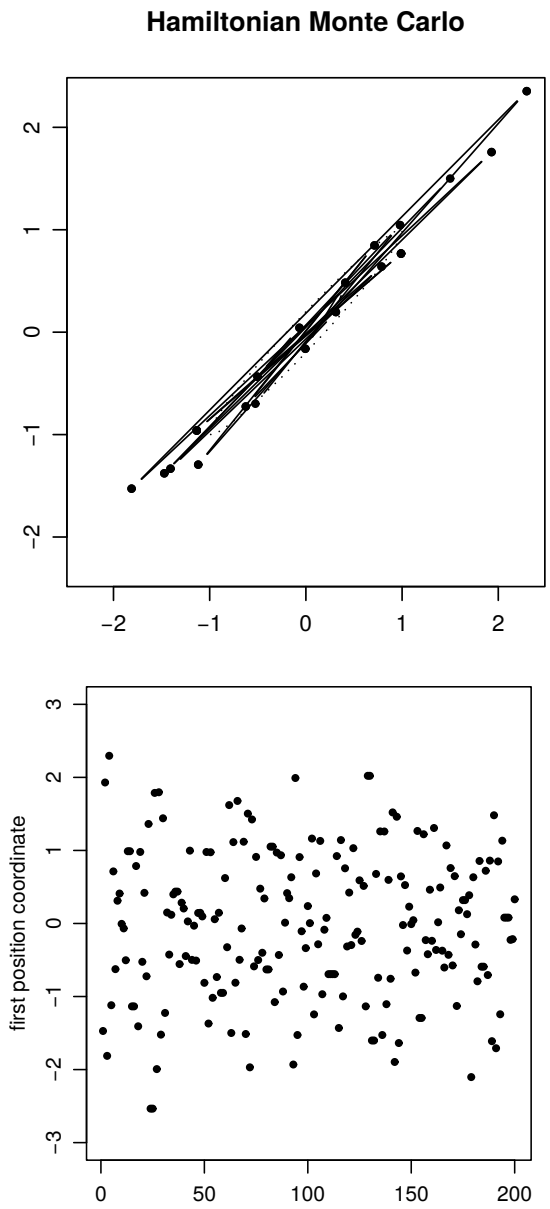






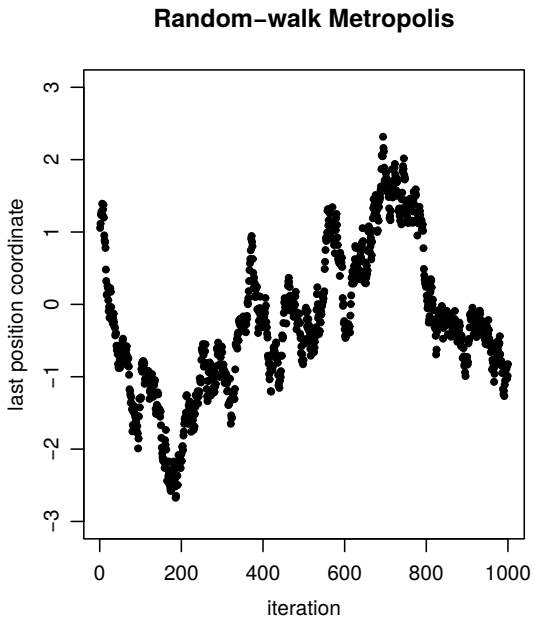
Random walk in 2D



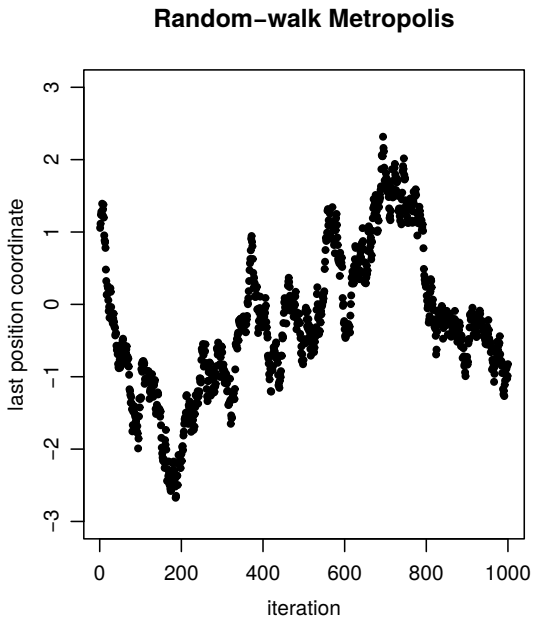


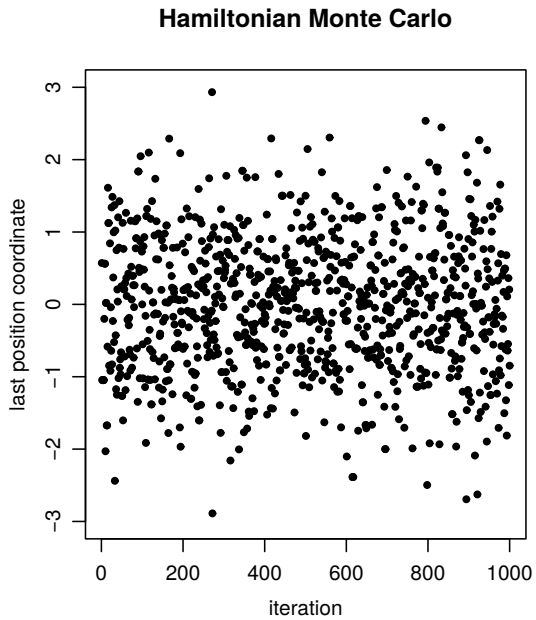
R. Neal, Handbook of MCMC methods

Random walk in 100D



Random walk in 100D





R. Neal, Handbook of MCMC methods

Hamiltonian Monte Carlo

Classical mechanics

- potential $U(\boldsymbol{x}) = -\log P(\boldsymbol{x})$
- add auxiliary momenta $\boldsymbol{p}$ in kin. energy
- Hamiltonian $H = \frac{1}{2}\boldsymbol{p}^T \boldsymbol{M}^{-1}\boldsymbol{p} + U(x)$

Update equations

From $(\boldsymbol{x}, \boldsymbol{p}) \to (\boldsymbol{x}', \boldsymbol{p}')$ by discretizing

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{p}} = \boldsymbol{M}^{-1}\boldsymbol{p}$$

$$\dot{\boldsymbol{p}} = -\frac{\partial H}{\partial \boldsymbol{x}} = -\nabla U(\boldsymbol{x})$$

Draw $\boldsymbol{p}'$, accept $\boldsymbol{x}'$ with prob. $\min(1, e^{-(H'-H)})$

How HMC beats "curse of dimensionality"

- gradient ∇U provides direction in space
- conserved quantities
- work even in $512^3 \approx 10^7$ dimensions J. Jasche (2015)

Hamiltonian Monte Carlo

Classical mechanics

- potential $U(\boldsymbol{x}) = -\log P(\boldsymbol{x})$
- add auxiliary momenta $\boldsymbol{p}$ in kin. energy
- Hamiltonian $H = \frac{1}{2}\boldsymbol{p}^T \boldsymbol{M}^{-1}\boldsymbol{p} + U(x)$

Update equations

From $(\boldsymbol{x}, \boldsymbol{p}) \to (\boldsymbol{x}', \boldsymbol{p}')$ by discretizing

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{p}} = \boldsymbol{M}^{-1}\boldsymbol{p}$$

$$\dot{\boldsymbol{p}} = -\frac{\partial H}{\partial \boldsymbol{x}} = -\nabla U(\boldsymbol{x})$$

Draw $\boldsymbol{p}'$, accept $\boldsymbol{x}'$ with prob. $\min(1, e^{-(H'-H)})$

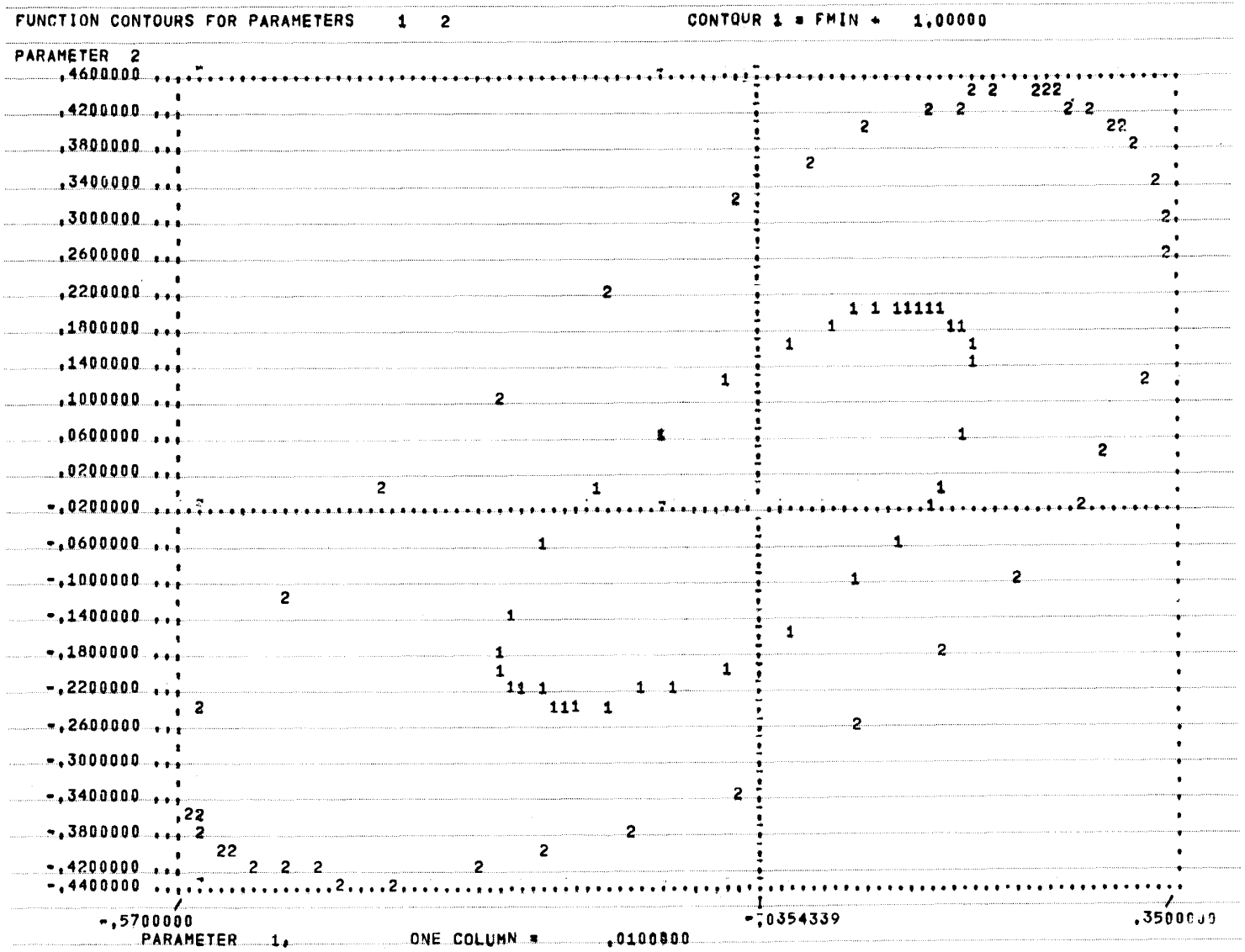
How HMC beats "curse of dimensionality"

- gradient ∇U provides direction in space
- conserved quantities
- work even in $512^3 \approx 10^7$ dimensions J. Jasche (2015)

Optimization

Minuit

- called from `hist.Fit()` in ROOT
- quasi-Newton method
  $f(\boldsymbol{x}_k + \boldsymbol{s}_k) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{s}_k + \frac{1}{2} \boldsymbol{s}_k^T B_k \boldsymbol{s}_k$
- update of inverse Hessian B_k needs ∇f, too
- default: estimate ∇f by central finite differencing
- problems in 1000D: 2001 calls of f + small error in each dimension but f falls off very rapidly
- minuit accepts user-supplied gradient, too
- pretty robust but dates back to late 1960s
- most popular in high dim.: L-BFGS (1980), e.g. in nlopt



F. James and M. Roos (1975)

automatic differentiation

- derivative via chain chain rule for basic operations
- get ∇f for $\lesssim 5$ calls of f
- more than 20 packages for C/C++ on wikipedia
- source-code transformation vs. operator overloading
- tuned for statistics: STAN math library, released 2015

Normal example

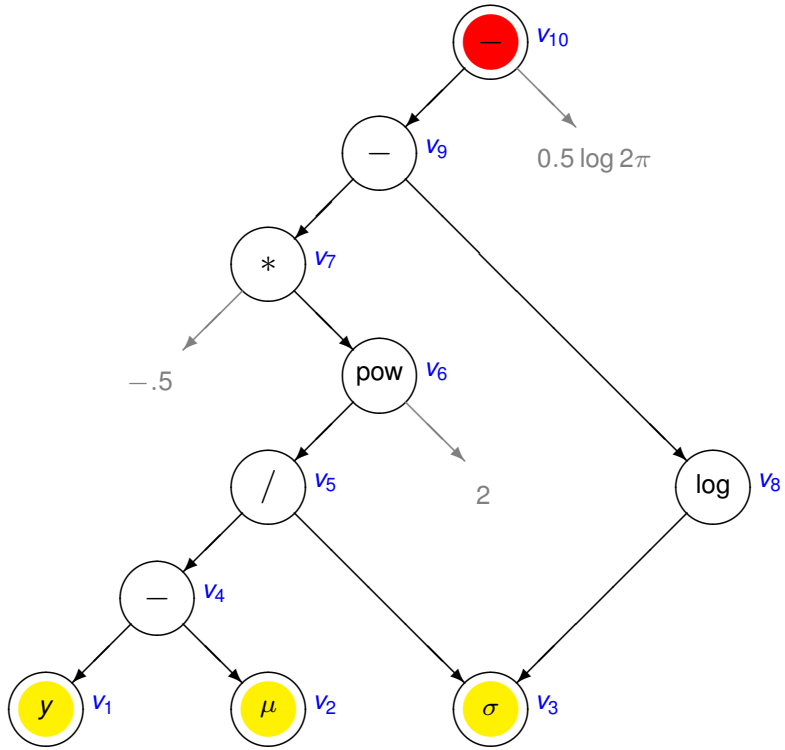
$$f(y, \mu, \sigma) = \log\left(\text{Normal}(y|\mu, \sigma)\right) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$

forward mode

- trace calculation from input y, μ, σ to output f
- need n passes for gradient of n inputs

reverse mode

- single forward pass to compute derivatives between nodes
- single reverse pass to compute gradient
- larger memory footprint: store derivatives and adjoints



STAN math documentation

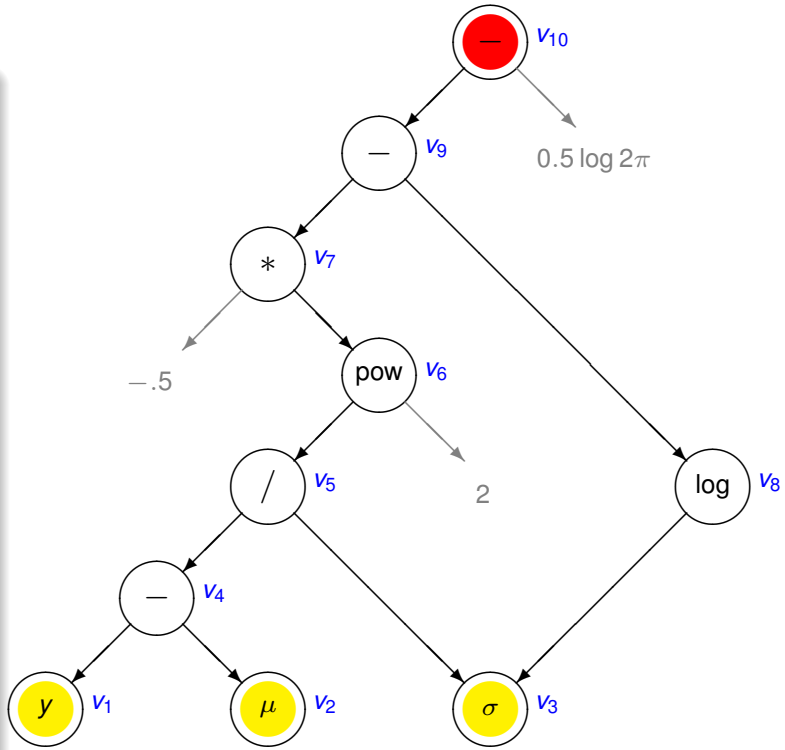
Applying the chain rule

$$f(y,\mu,\sigma) = \log\left(\text{Normal}(y|\mu,\sigma)\right) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$

forward pass

forward pass

var	*value*	*partials*
v_1	y	
v_2	μ	
v_3	σ	
v_4	$v_1 - v_2$	$\partial v_4/\partial v_1 = 1$ $\quad$ $\partial v_4/\partial v_2 = -1$
v_5	v_4/v_3	$\partial v_5/\partial v_4 = 1/v_3$ $\quad$ $\partial v_5/\partial v_3 = -v_4 v_3^{-2}$
v_6	$(v_5)^2$	$\partial v_6/\partial v_5 = 2v_5$
v_7	$(-0.5)v_6$	$\partial v_7/\partial v_6 = -0.5$
v_8	$\log v_3$	$\partial v_8/\partial v_3 = 1/v_3$
v_9	$v_7 - v_8$	$\partial v_9/\partial v_7 = 1$ $\quad$ $\partial v_9/\partial v_8 = -1$
v_{10}	$v_9 - (0.5\log 2\pi)$	$\partial v_{10}/\partial v_9 = 1$



STAN math documentation

$$\frac{\partial f}{\partial \sigma} = \frac{\partial f}{\partial v_{10}}\frac{\partial v_{10}}{\partial \sigma} = \frac{\partial f}{\partial v_{10}}\frac{\partial v_{10}}{\partial v_9}\frac{\partial v_9}{\partial \sigma} = \frac{\partial f}{\partial v_{10}}\frac{\partial v_{10}}{\partial v_9}\left(\frac{\partial v_9}{\partial v_7}\frac{\partial v_7}{\partial \sigma} + \frac{\partial v_9}{\partial v_8}\frac{\partial v_8}{\partial \sigma}\right) = \dots$$

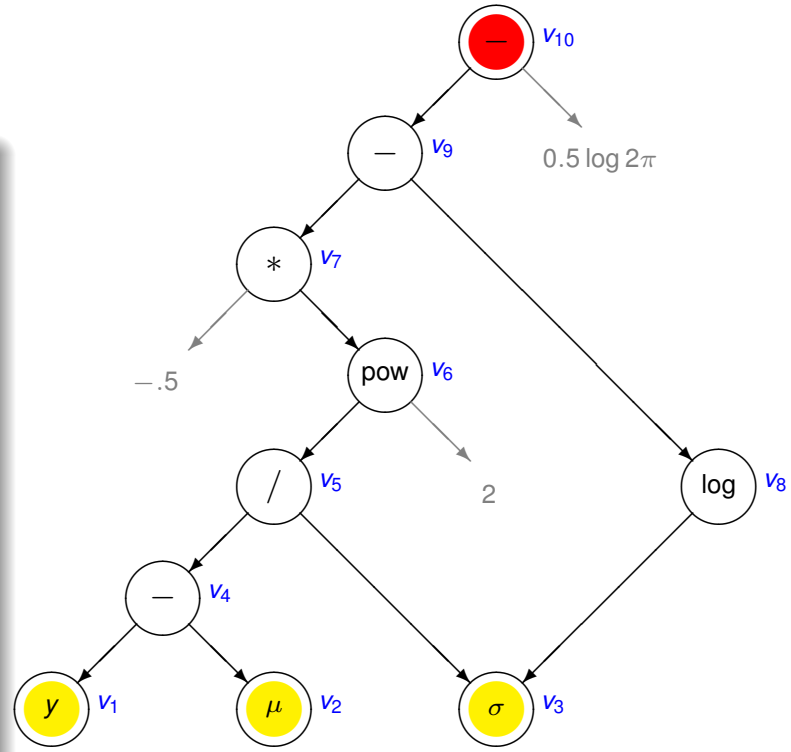
Applying the chain rule

$$f(y,\mu,\sigma) = \log\left(\text{Normal}(y|\mu,\sigma)\right) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$

reverse pass

duplicate graph with adjoints a_i

var	*operation*	*adjoint*	*result*
$a_{1:9}$	$=$	0	$a_{1:9} = 0$
a_{10}	$=$	1	$a_{10} = 1$
a_9	$+=$	$a_{10} \times \partial v_{10}/\partial v_9$	$a_9 = 1 \times 1 = 1$
$\dots$			
a_3	$+=$	$a_8 \times \partial v_8/\partial v_3$	$a_3 = -1 \times 1/v_3$
$\dots$			
a_3	$+=$	$a_5 \times \partial v_5/\partial v_3$	$a_3 = -1/v_3 + v_5 v_4 v_3^{-2}$
a_1	$+=$	$a_4 \times (1)$	$a_1 = -v_5/v_3$
a_2	$+=$	$a_4 \times (-1)$	$a_2 = v_5/v_3$



STAN math documentation

$$\frac{\partial f}{\partial \sigma} = \frac{\partial f}{\partial v_{10}}\frac{\partial v_{10}}{\partial \sigma} = \frac{\partial f}{\partial v_{10}}\frac{\partial v_{10}}{\partial v_9}\frac{\partial v_9}{\partial \sigma} = \frac{\partial f}{\partial v_{10}}\frac{\partial v_{10}}{\partial v_9}\left(\frac{\partial v_9}{\partial v_7}\frac{\partial v_7}{\partial \sigma} + \frac{\partial v_9}{\partial v_8}\frac{\partial v_8}{\partial \sigma}\right) = \dots$$

operator overloading in C++

$$f(y, \mu, \sigma) = \log\left(\text{Normal}(y|\mu, \sigma)\right) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$

```
template <typename T1, typename T2, typename T3>
typename boost::math::tools::promote_args<T1, T2, T3>::type
normal_log(const T1& y, const T2& mu, const T3& sigma) {
using std::pow; using std::log;
return -0.5 * pow((y - mu) / sigma, 2.0) - log(sigma)
       -0.5 * log(2 * stan::math::pi());
}
```

regular evaluation: `T = double`, autodiff: `T = stan::math::var`

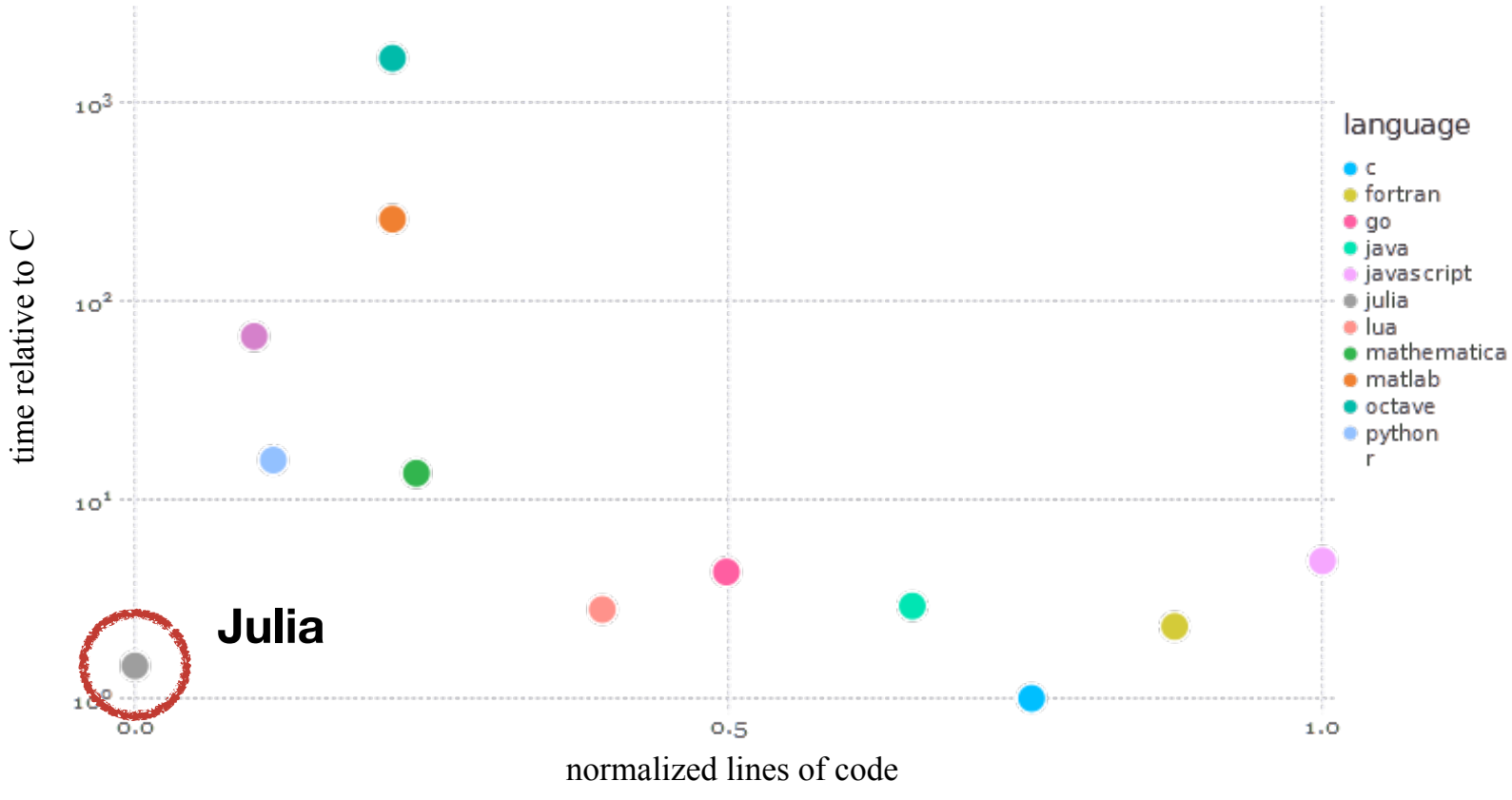
invasive, hard to use in existing projects

Julia

Main features

- new open-source language (2012), julialang.org
- designed for technical computing
- as expressive and interactive as python
- just-in-time compiled via LLVM, near C speed
- call C, Fortran, python, R w/o glue code
- advanced code inspection: codes can inspect itself
- multiple dispatch instead of classes with methods

Speed vs. Productivity



Stefan Karpinski, DIANA meeting at CERN, Jul 2016

julia functions and automatic differentiation

```
# define function w/o types
julia> f(x) = x^3
f (generic function with 1 method)

# 1st call: compile
julia> @time f(5.0)
  0.000600 seconds (149 allocations: 10.167 KB)
125.0

# 2nd call: use compiled version
julia> @time f(5.0)
  0.000004 seconds (5 allocations: 176 bytes)

# specific code for Float64 argument
julia> code_llvm(f, (Float64,))
define double @julia_f_22990(double) {
top:
  %1 = fmul double %0, %0
  %2 = fmul double %1, %0
  ret double %2
}
```

julia functions and automatic differentiation

```
# define function w/o types
julia> f(x) = x^3
f (generic function with 1 method)

# 1st call: compile
julia> @time f(5.0)
  0.000600 seconds (149 allocations: 10.167 KB)
125.0

# 2nd call: use compiled version
julia> @time f(5.0)
  0.000004 seconds (5 allocations: 176 bytes)

# specific code for Float64 argument
julia> code_llvm(f, (Float64,))
define double @julia_f_22990(double) {
top:
  %1 = fmul double %0, %0
  %2 = fmul double %1, %0
  ret double %2
}
```

```
# install package from internet
# including dependencies
julia> Pkg.add("ReverseDiffSource")
INFO: Installing ReverseDiffSource v0.2.3
INFO: Package database updated

# import symbols of the package
julia> using ReverseDiffSource

# generate source of gradient
# from source code
julia> rdiff( :(x^3), x=5.)
quote
    (x ^ 3,3 * x ^ 2)
end

# function and gradient from a function
julia> f_gradf = rdiff(f, (5.0,))
(anonymous function)

# evaluate function and gradient together
julia> f_gradf(5)
(125,75)
```

reverse mode: source transformation

julia functions and automatic differentiation

```
using Optim, ForwardDiff

# .^ = square elementwise
h(x::Vector) = sum(x.^2)

# Newton's method needs gradient and Hessian
julia> res = optimize(h, [111, -2222.2], Newton(), OptimizationOptions(autodiff=true))
 ...
 * Minimizer: [-1.4210854715202004e-14,0.0]
 * Minimum: 2.019484e-28
 * Iterations: 1
 ...
 * Objective Function Calls: 5
 * Gradient Calls: 5

julia> ForwardDiff.gradient(h, [0.0, 0.0])
2-element Array{Float64,1}:
 0.0
 0.0

julia> ForwardDiff.hessian(h, Optim.minimizer(res))
2x2 Array{Float64,2}:
 2.0  0.0
 0.0  2.0
```

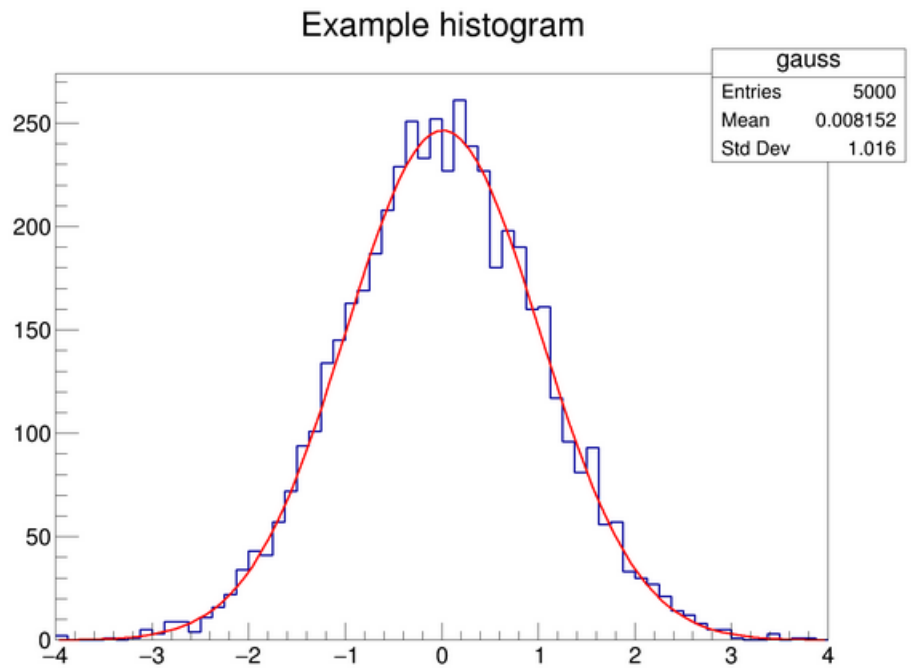
forward mode: operator overloading

notebooks

- ipython notebook $\Rightarrow$ jupyter
- supports over 40 languages
- explore data from the browser
- try C++ online via cling at CERN
- try julia online on juliabox

Let's now fit the histogram.

In [6]:
```
h.Fit("gaus", "S");
c.Draw();
```




```
 FCN=47.4997 FROM MIGRAD    STATUS=CONVERGED      53 CALLS          54 TOTA
L
                     EDM=8.44224e-09    STRATEGY= 1      ERROR MATRIX ACCUR
ATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     2.46469e+02   4.31494e+00   1.19094e-02  -2.44811e-05
   2  Mean         1.04782e-02   1.43576e-02   4.87656e-05  -6.34020e-03
   3  Sigma        1.00315e+00   1.03818e-02   9.45504e-06  -2.70309e-02
```

notebooks

- ipython notebook $\Rightarrow$ jupyter
- supports over 40 languages
- explore data from the browser
- try C++ online via cling at CERN
- try julia online on juliabox

In [7]:
```
const ⊗ = kron
eye(2,2) ⊗ rand(2,2)
```

Out[7]:
```
4x4 Array{Float64,2}:
 0.68628   0.553253  0.0       0.0
 0.141211  0.25523   0.0       0.0
 0.0       0.0       0.68628   0.553253
 0.0       0.0       0.141211  0.25523
```

Functions and JIT-compilation

Functions can be defined in several ways, and *don't*

In [8]:
```
# verbose form:
function foo(x)
    return x + 1
end

# one-line form:
foo(x) = x + 1

# anonymous function
x -> x + 1
```

Out[8]:
```
(anonymous function)
```

Conclusion

- optimizing/sampling w/o gradient = driving w/o a map
- automatic differentiation $\Rightarrow$ gradient, Hessian
- will julia replace python in 5–10 years?
- notebooks can change the way we work and teach