PYTHON FOR OPTIMIZATION

BEN MORAN

@BENM

HTTP://BENMORAN.WORDPRESS.COM/

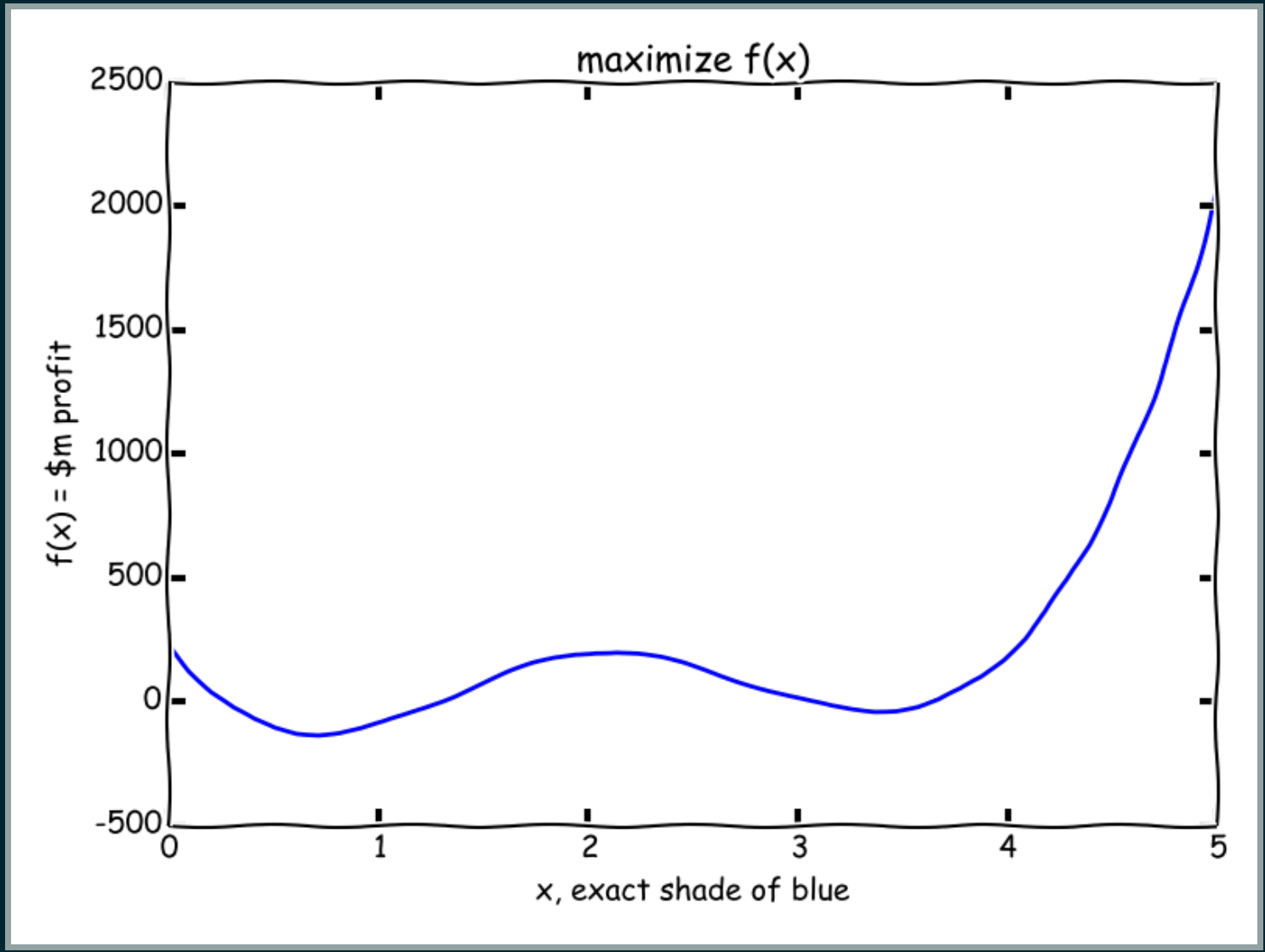
2014-02-22

INTRODUCTION

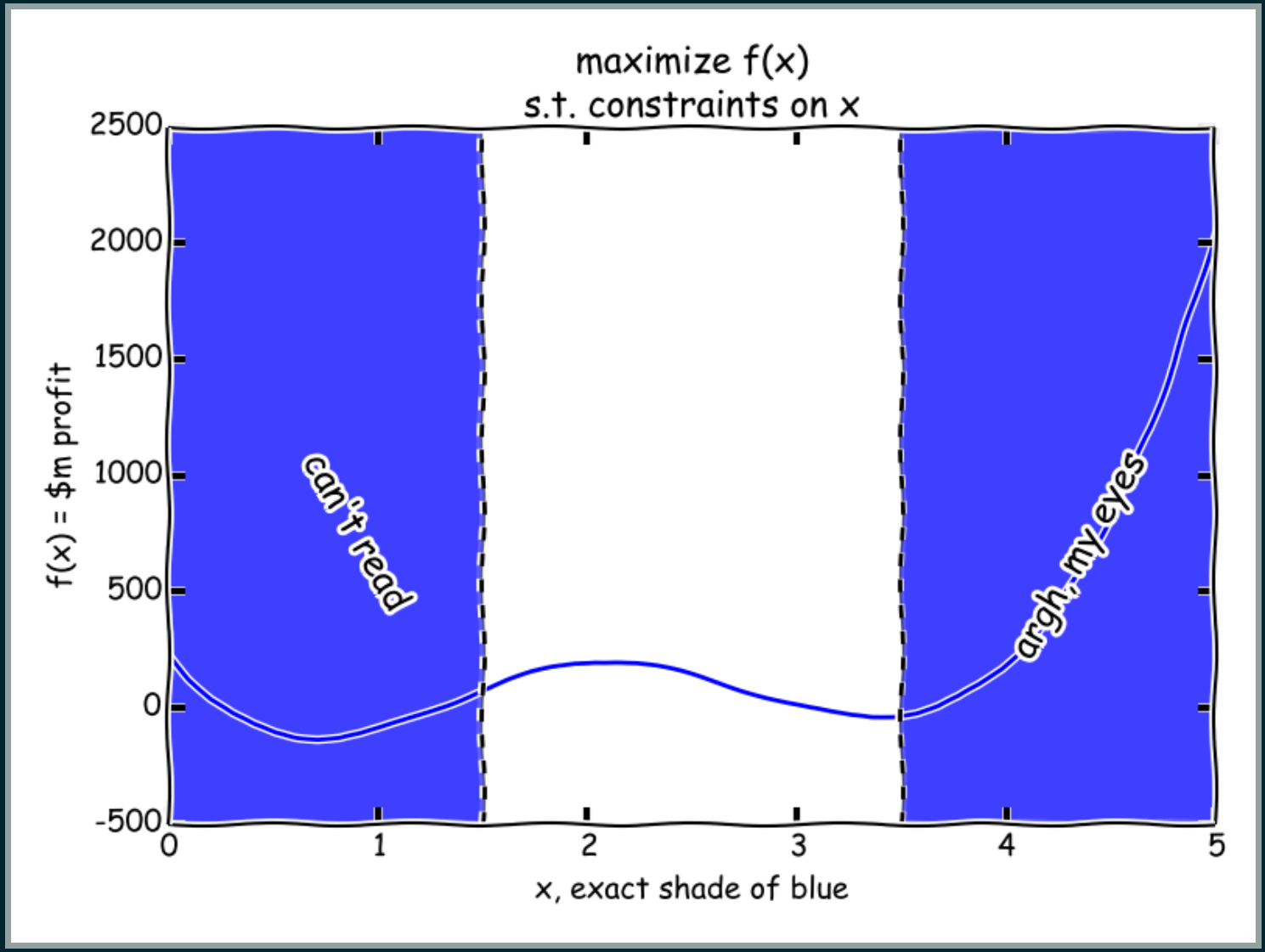
- Python for optimization
  - Not *optimizing Python programs*
  - Not *website optimization/SEO*
  - Mathematical optimization!
    - `scipy.optimize` and friends

MATHEMATICAL OPTIMIZATION

OBJECTIVE FUNCTION



WITH CONSTRAINTS



APPLICATIONS

- Engineering
- Finance
- Operations Research
- Machine learning
- Statistics
- Physics

PYTHON FOR OPTIMIZATION

- Exploration/visualization: IPython notebook/Matplotlib/…
- As a high level modelling language
- Batteries included: `scipy.optimize`, `statsmodels`, 3rd party solver support
- Cython, etc. for C bindings and high performance code

REST OF THE TALK

- Numerical optimization vs symbolic manipulation
- General purpose solvers: `scipy.optimize`
- Linear/convex solvers: Python as a modelling language
- Smooth optimization: Sympy for derivatives

OPTIMIZATION TYPES

NO STRUCTURE

```
import numpy as np
xx = np.arange(1000)
f = np.random.randn(len(xx))
x = np.argmin(f)
```

LOTS OF STRUCTURE

```
xx = np.arange(1000)
f = 2*xx
x = np.argmin(f)
```

QUADRATIC

```
xx = np.arange(-1000,1000)
f = (xx-400)**2
x = np.argmin(f)
```

SCIPY.OPTIMIZE

```
import numpy as np
import scipy.optimize
f = lambda x: np.exp((x-4)**2)
return scipy.optimize.minimize(f, 5)
```

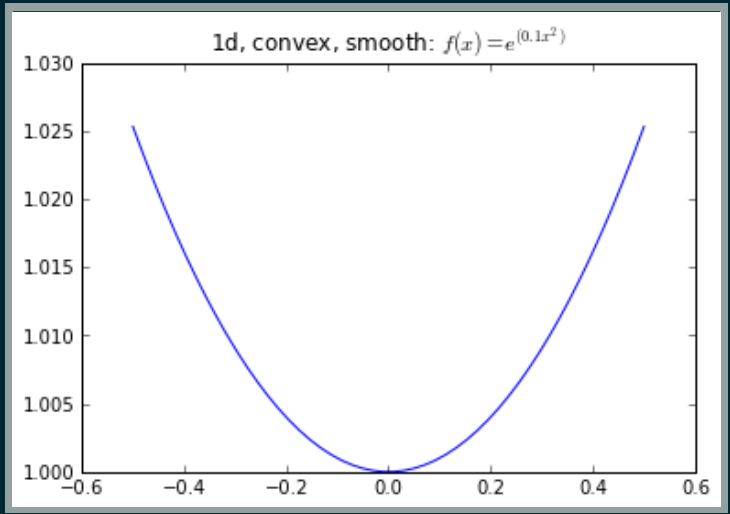
```
status: 0
  success: True
     njev: 8
     nfev: 24
 hess_inv: array([[ 0.49999396]])
      fun: 1.0
        x: array([ 3.99999999])
  message: 'Optimization terminated successfully.'
      jac: array([ 0.])
```

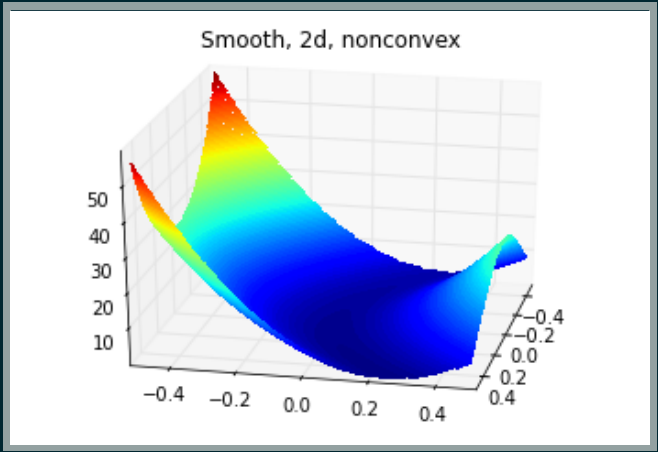
CHOOSING A SOLVER METHOD

Methods in `scipy.optimize.minimize`

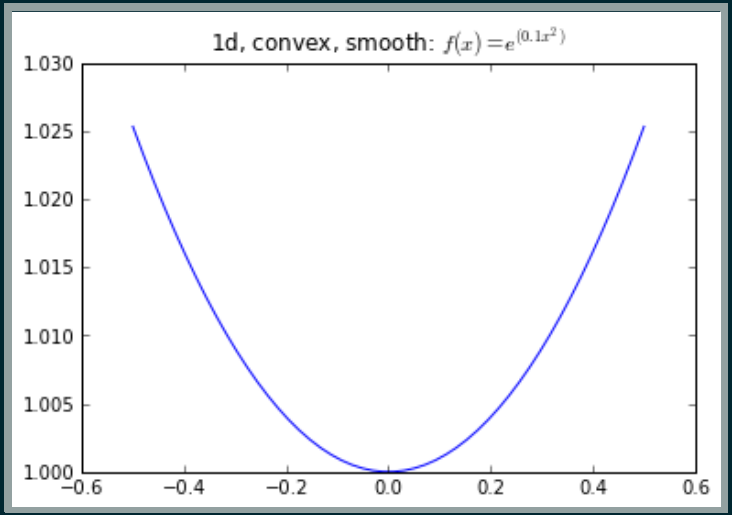
BFGS (default)	1st		
Nelder-Mead			
Powell			
CG	1st		
Newton-CG	2nd		
Anneal		Global	
dogleg	2nd		
L-BFGS-B	1st		bounds
TNC	1st		bounds
Cobyla			inequality
SLSQP			equality/inequality

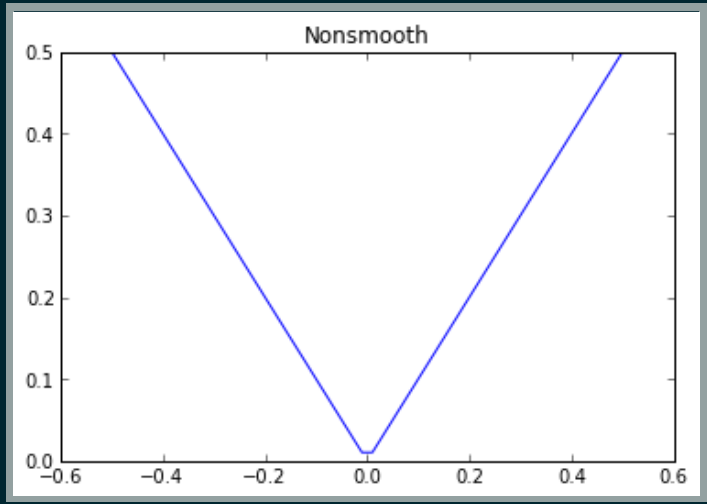
SIZE: FROM LOW TO HIGH DIMENSIONAL

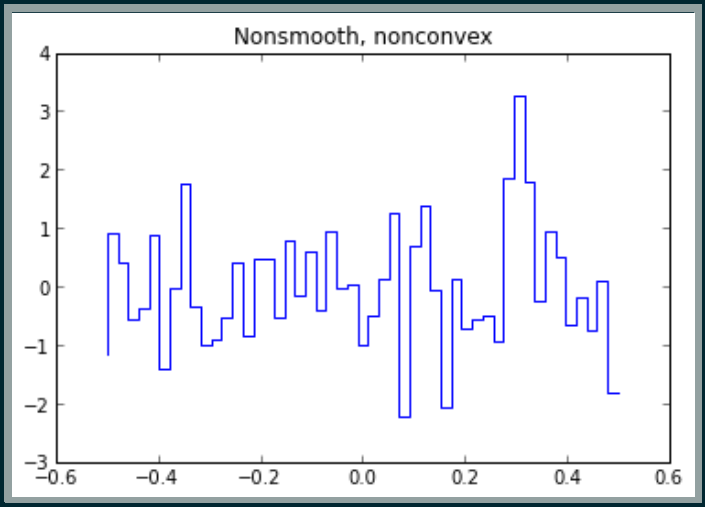




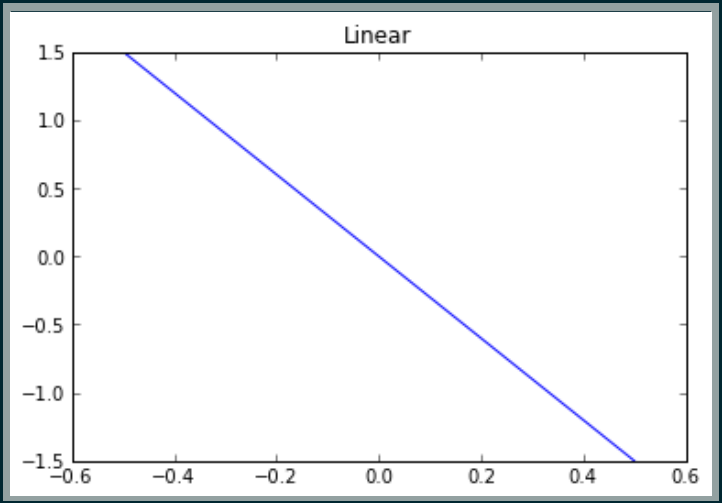
SMOOTHNESS: FROM SMOOTH TO DISCONTINUOUS

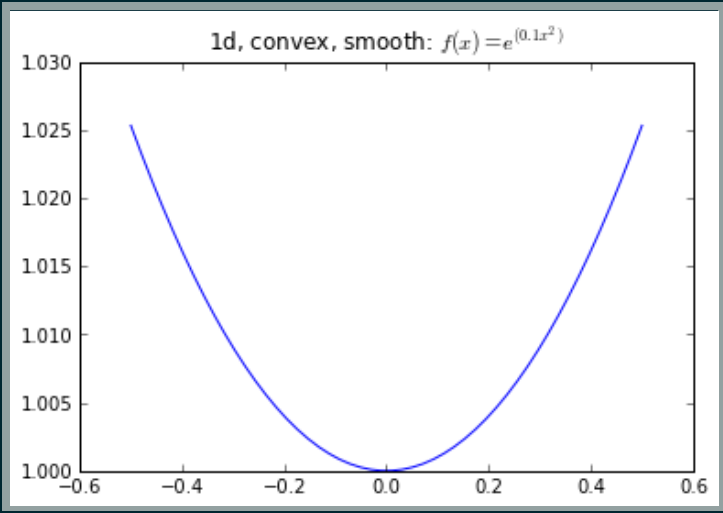


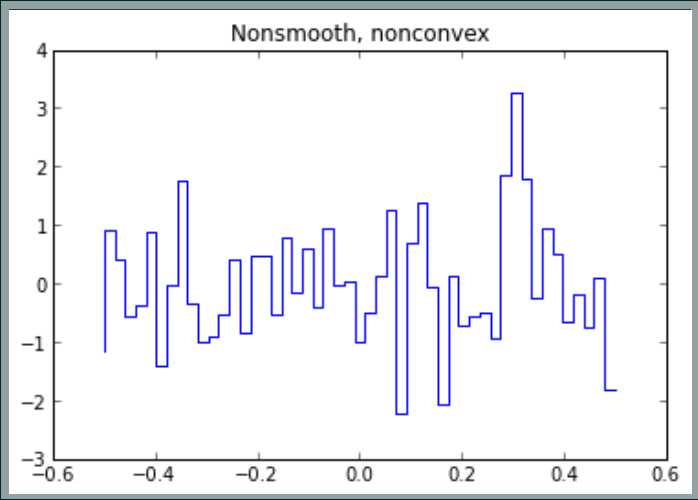


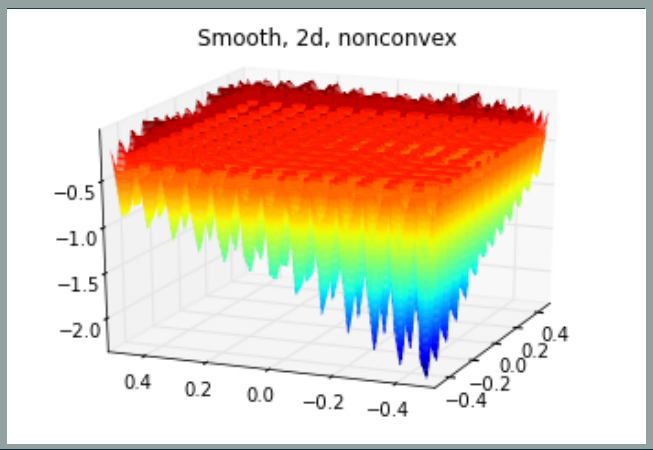


GLOBAL SHAPE: FROM LINEAR TO STRICTLY CONVEX TO MANY LOCAL MINIMA







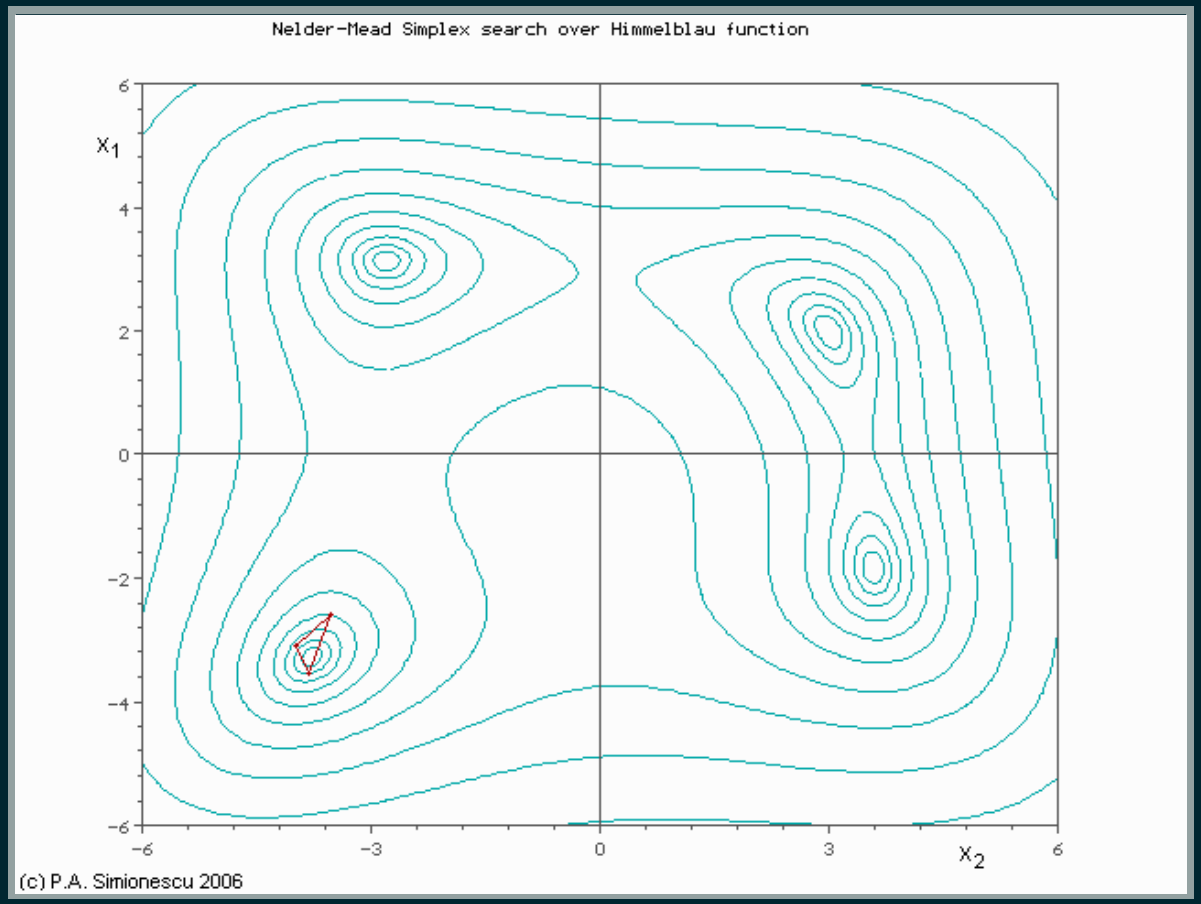


NELDER-MEAD

Just uses continuity, expects few minima

```
import numpy as np
import scipy.optimize
f = lambda x: np.exp((x-4)**2)
return scipy.optimize.minimize(f, 5, method='Nelder-Mead')
```

```
status: 0
    nfev: 32
 success: True
     fun: 1.0
       x: array([ 4.])
 message: 'Optimization terminated successfully.'
     nit: 16
```

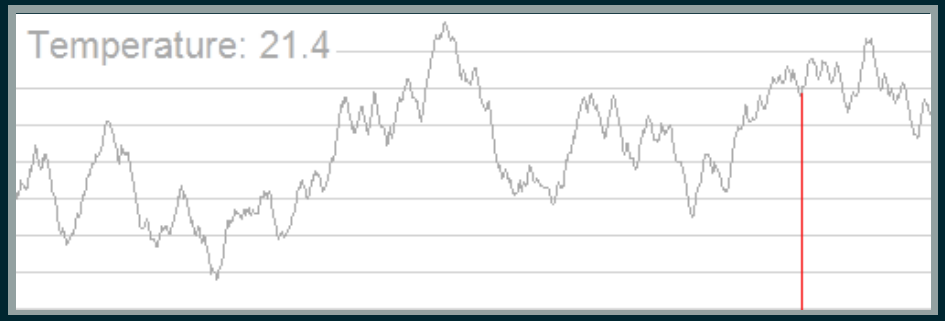

Nelder-Mead Simplex search over Himmelblau function
X1
X2
6
4
2
0
-2
-4
-6
-6
-3
0
3
6
(c) P.A. Simionescu 2006

SIMULATED ANNEALING

Expects multiple minima

```
import numpy as np
import scipy.optimize
f = lambda x: np.exp((x-4)**2)
return scipy.optimize.minimize(f, 5, method='anneal')
```

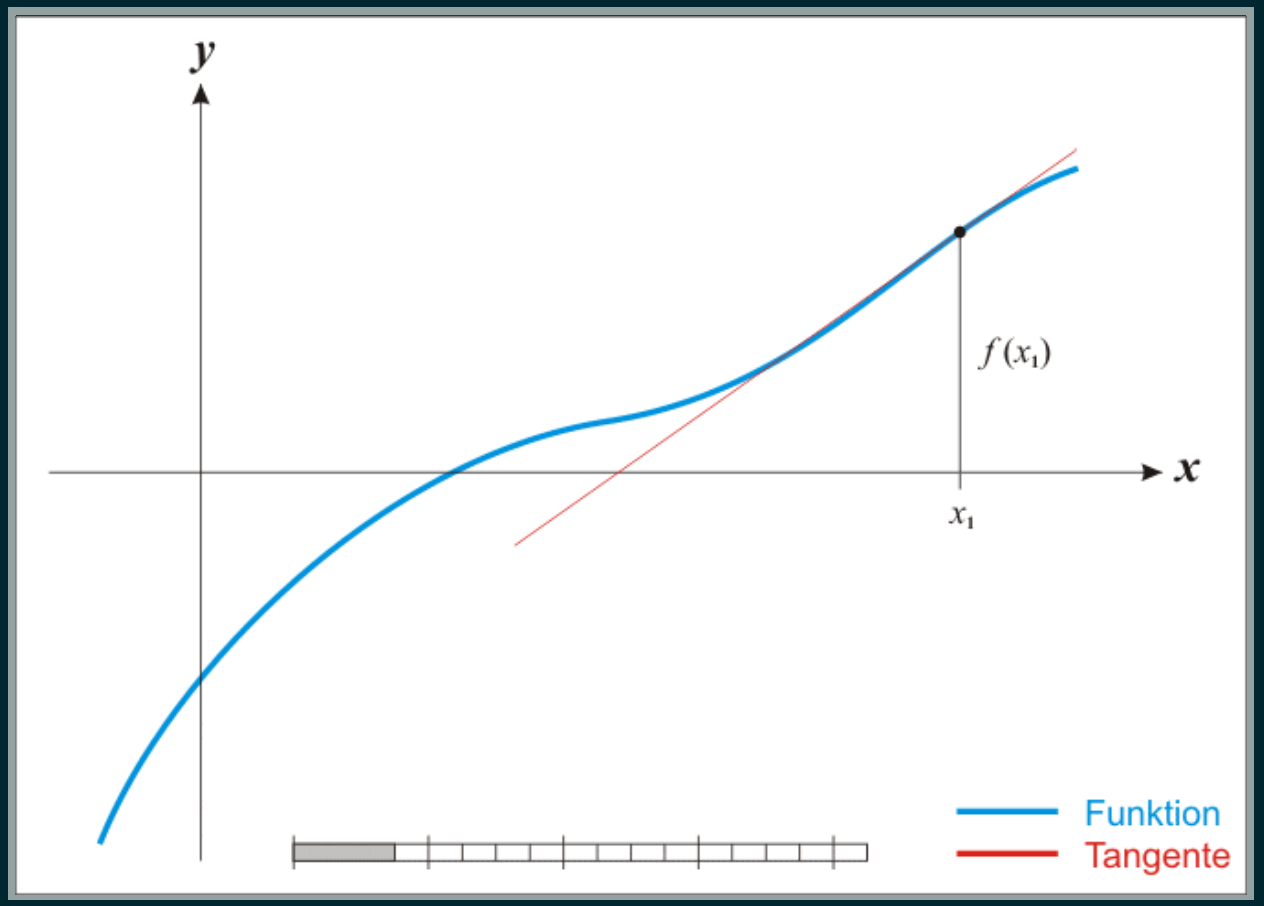
```
status: 0
 success: True
  accept: 0
    nfev: 251
       T: inf
     fun: 6.760670043672425
       x: array(2.617566636676699)
 message: 'Points no longer changing'
     nit: 4
```



NEWTON'S METHOD

```
import numpy as np
import scipy.optimize
f = lambda x: np.exp((x-4)**2)
fprime = lambda x: 2*(x-4)*np.exp((x-4)**2)
return scipy.optimize.minimize(f, 5, jac=fprime, method='Newton-CG')
```

```
status: 0
 success: True
    njev: 24
    nfev: 7
     fun: array([ 1.])
       x: array([ 4.00000001])
 message: 'Optimization terminated successfully.'
    nhev: 0
     jac: array([  2.46202099e-08])
```


y
x
$f(x_1)$
x_1
Funktion
Tangente

USAGE TIPS

- check warnings
- consider tolerances
- `check_grad` if supplying derivatives

PYTHON FOR MODELLING

- For large scale problems
- When we need to go fast
- Specialized solvers

LINEAR PROGRAMS AND PULP

$$\text{minimize } f(\mathbf{x}) = \mathbf{Ax} + \mathbf{b}$$
$$\text{subject to } \mathbf{Cx} \leq \mathbf{b}$$

- http://pythonhosted.org/PuLP/
- Modelling language for LP's
- Backends including Coin-OR, GLPK, Gurobi
- Supports integer constraints
- PuLP simple LP
- PuLP Sudoku

CVXPY

- Nonlinear convex optimization, backed by CVXOPT
- The LASSO L_1 -penalized least squares problem:

```
from cvxpy import *
import numpy as np
import cvxopt

# Problem data.
n = 10
m = 5
A = cvxopt.normal(n,m)
b = cvxopt.normal(n)
gamma = Parameter(sign="positive")

# Construct the problem.
x = Variable(m)
objective = Minimize(sum(square(A*x - b)) + gamma*norm1(x))
p = Problem(objective)
gamma.value = 0.2
result = p.solve()
return p.is_dcp()
```

WIDE RANGE OF EXPRESSIONS

- `kl_div(x, y)`
  - KL divergence
- `lambda_max(x)`, `lambda_min(x)`
  - the max/min eigenvalue of x.
- `log_det`
  - $\log(\det(x))$ for a positive semidefinite matrix x.
- `norm(x, [p = 2])`,
  - L1/L2/infinity/Frobenius/Spectral norm of x

DERIVATIVES AND SYMPY

LEAST SQUARES VIA THE FEYNMAN ALGORITHM

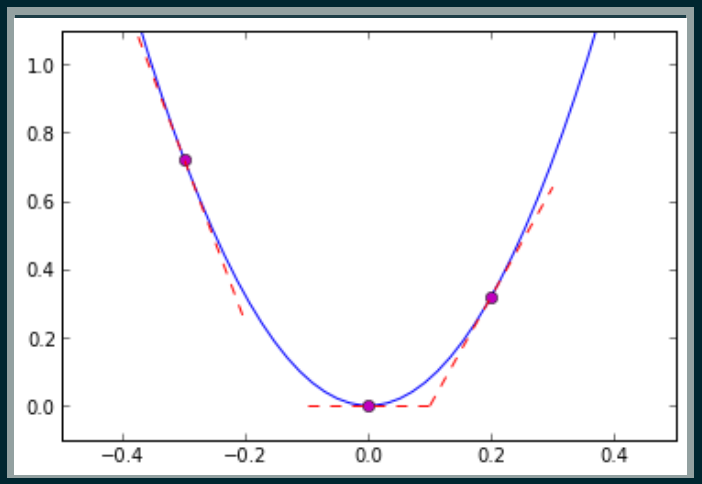
Write down the problem…

$$\text{minimize } f(x) = \sum_i (x_i - a_i)^2$$

… think really hard …

... AND THEN WRITE DOWN THE ANSWER!

$$\text{minimize } f(x) = \sum_i (x_i - a_i)^2$$



- Take derivatives
  The gradient is flat at the bottom of the hill. We can find the gradient in each direction x_i explicitly, and set it to zero:

$$\frac{\partial f}{\partial x_i} = 2(x_i - a_i) = 0$$

WHERE TO GET DERIVATIVES FROM?

- Numerical approximation - `numdifftools`
- (*Automatic differentiation - Julia, Stan, ADMB…*)
- Pencil and paper!
- Sympy

CONCLUSION

To choose a solver:

- Small problem/no hurry
  Use whatever you like!
- Big problem/top speed
  Understand and specialize
- Really big problem
  Try online methods like SGD
- Don't understand the problem at all?
  Try a genetic algorithm