Python, numerical optimization, genetic algorithms

Davide Rizzo
PyCon Italia Qu4ttro

daviderizzo.net

Optimization

- Actually *operations research*
  - Mathematical optimization is the tool
- Applied maths
  - Major academic and industrial research topic
- Computationally oriented
  - Many commercial solutions
  - Yes, you can use Python
- Not code optimization!

Operations research applications

- Airlines
  - scheduling planes and crews, pricing tickets, taking reservations, and planning fleet size
- Logistics
  - routing and planning
- Financial services
  - credit scoring, marketing, and internal operations
- Deployment of emergency services
- Policy studies and regulation
  - environmental pollution, air traffic safety, AIDS, and criminal justice policy

More operations research applications

- Project planning
- Network optimization
- Resources allocation
- Supply chain management
- Automation
- Scheduling
- Pricing

Optimization problem

- Many decision variables
  - How should I price my new products?
  - How many server resources should I assign to each client?
- An objective
  - Maximize net profit
  - Minimize client waiting time

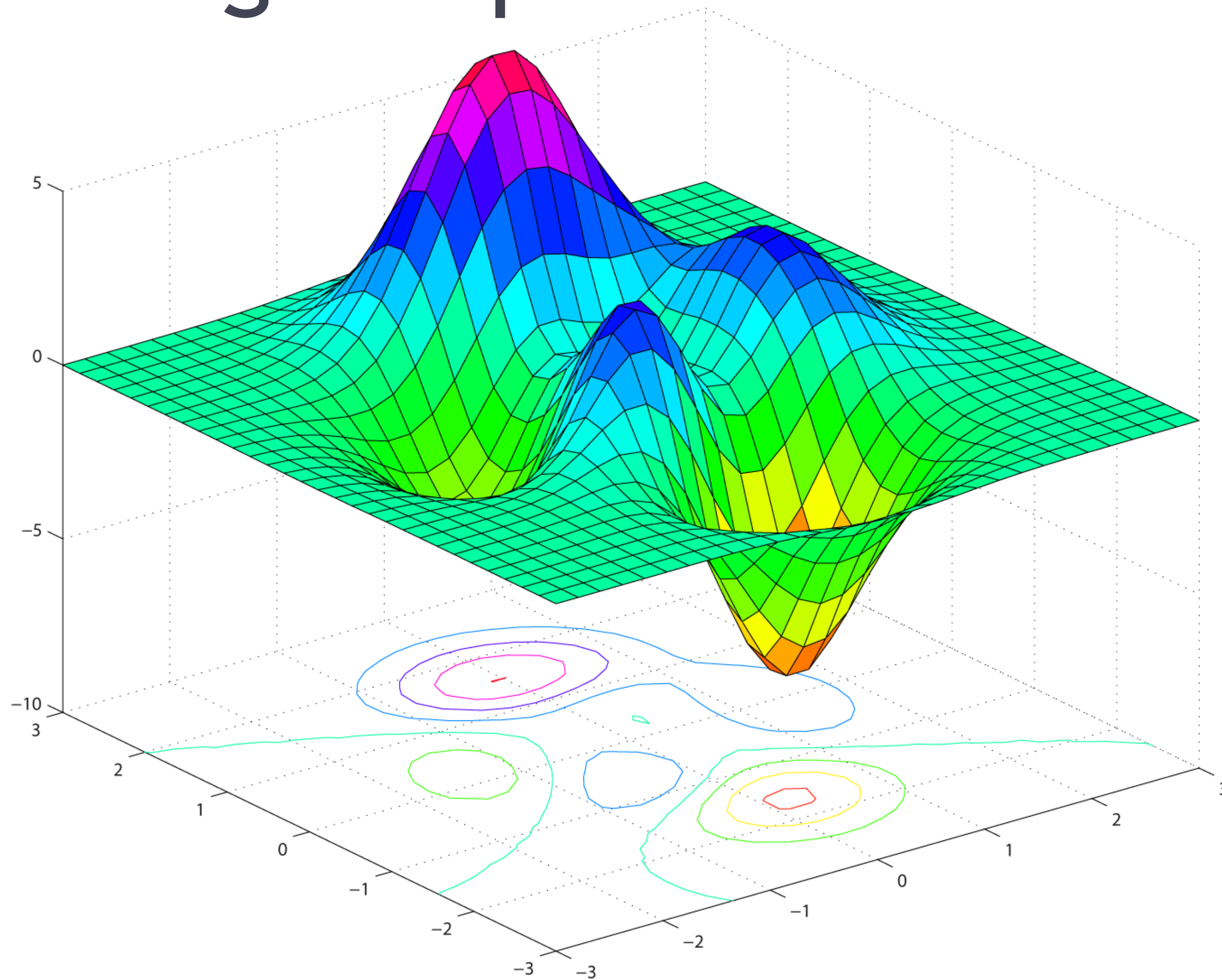
Objective function

- Objective: find the best choice for the decision variables
  - the values that give the maximum yield (or the minimum cost)
- **Objective function**: the yield (or the cost) I get for a given choice

Decisions and objectives

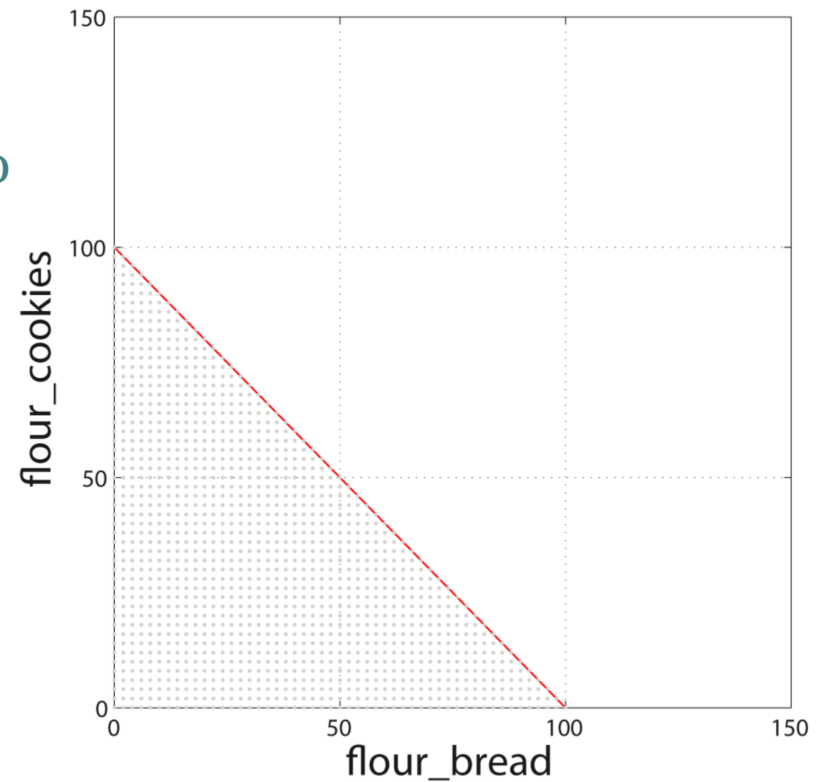
- I buy 100kg flour at 50€
  - Want to decide how to use it
- 70kg -> bread, 30kg -> cookies
  - Decision variables
    - X = 70, 30
- Sell 105€ of bread and 75€ of cookies
  - Objective function value:
    - f(X) = 105 + 75 – 50 = 130€

Visualizing the problem



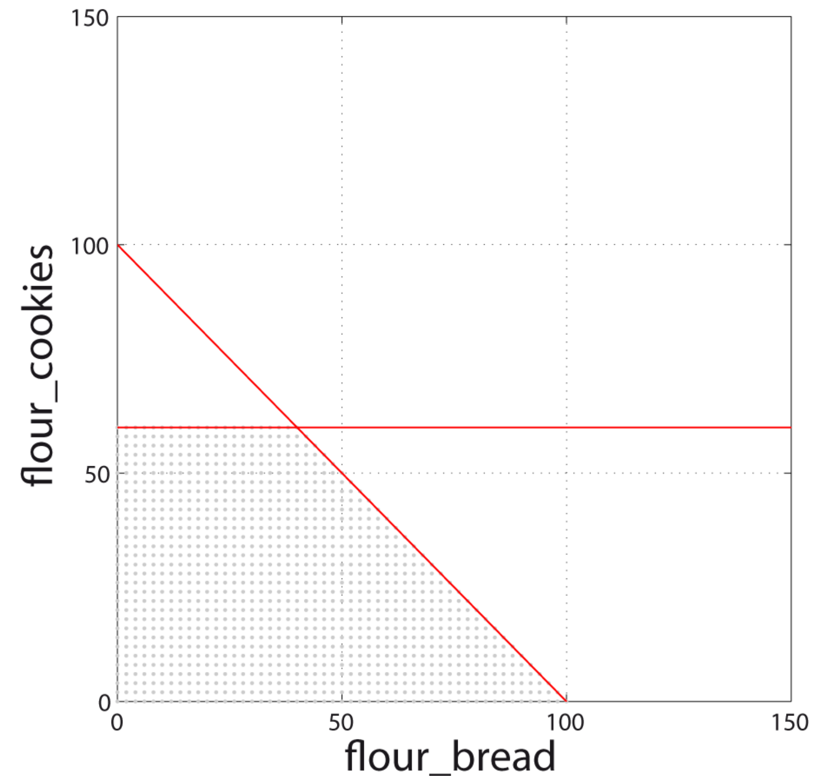
Constraints

- Not every solution is good
- 100kg flour → can't use 150kg for bread
    - flour_bread + flour_cookies ≤ 100



Constraints

- Inequalities often describe constraints
- Many constraints
  - There are problems with thousands inequalities
- Limited sugar
  - 1 part sugar, 3 parts flour
  - 20 kg sugar available
  - flour_cookies < 60



Search vs. optimization

Search algorithms	Optimization algorithms

• Well-known among programmers
• Include tree-search and graph-search algorithms
• Work on a discrete search space
	• Broader class of problems
	  ▫ Includes search problems
	• Continuous search space
	  ▫ Discrete as a special case
	• Search algorithms used to solve many optimization problems

Numerical optimization

Finding the optimum in linear and non-linear problems with exact methods

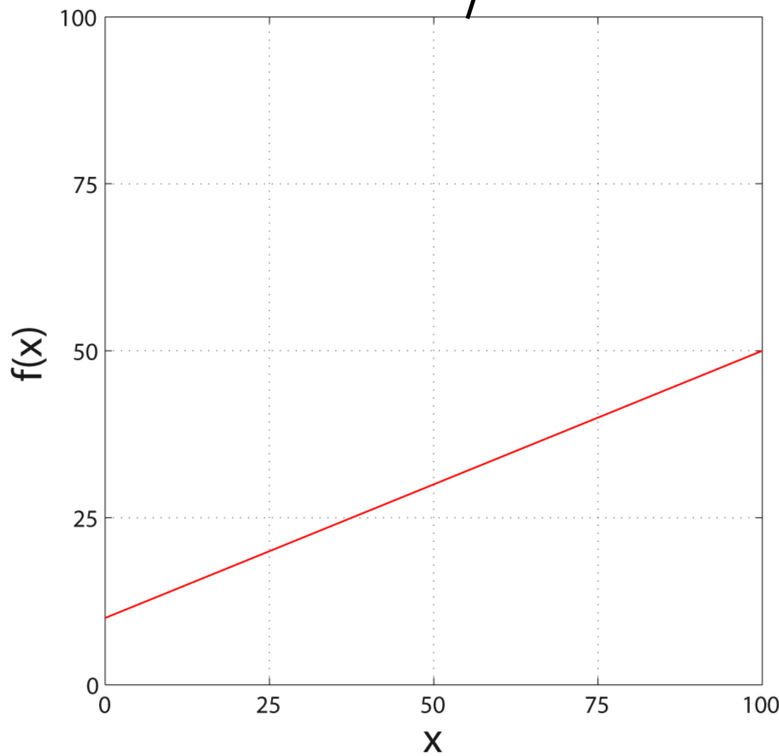
Objective function classification

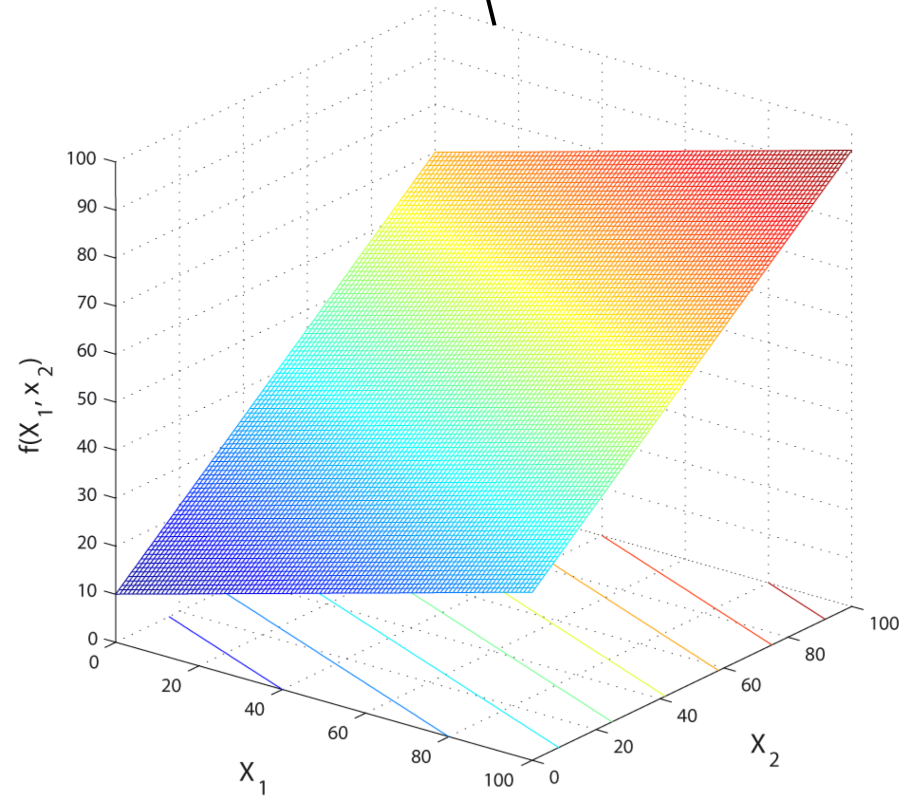
- General theory for every problem
  - Very good results for simple problems
- Objective function structure
  - Smoothness and regularity determine complexity
  - Good guarantees for linear functions
  - No exact results if irregular
- Factors other than objective functions
  - Constraints
  - Integrality on variables

Linear

Line (one variable)

Plane (two variables)





A greater number of variables does not complicate matters!

Linear

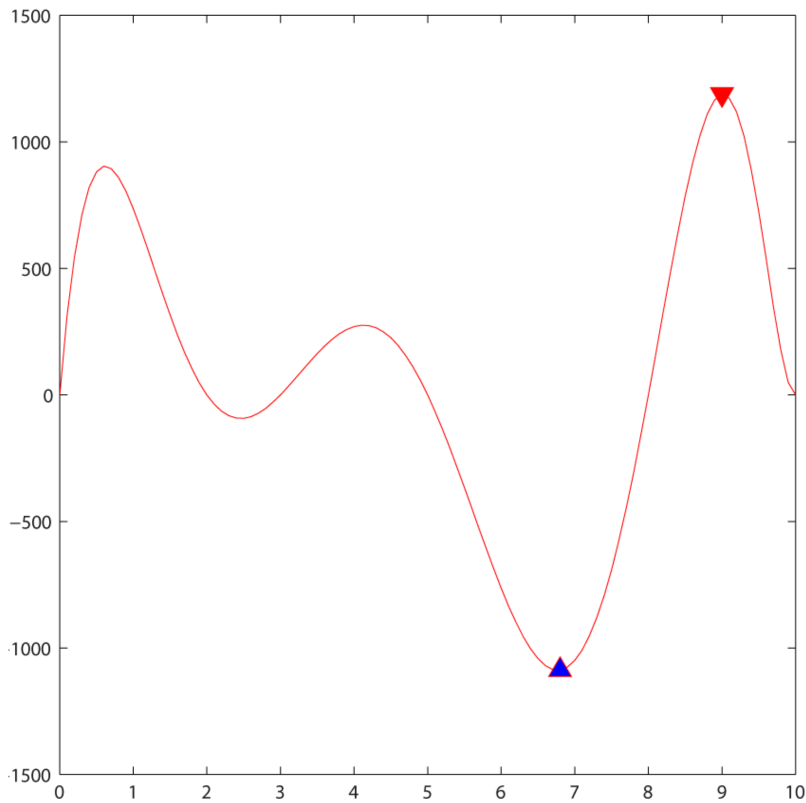
- Linear objective function
  - `c0 + sum(c[i]*x[i] for i in range(len(vars)))`

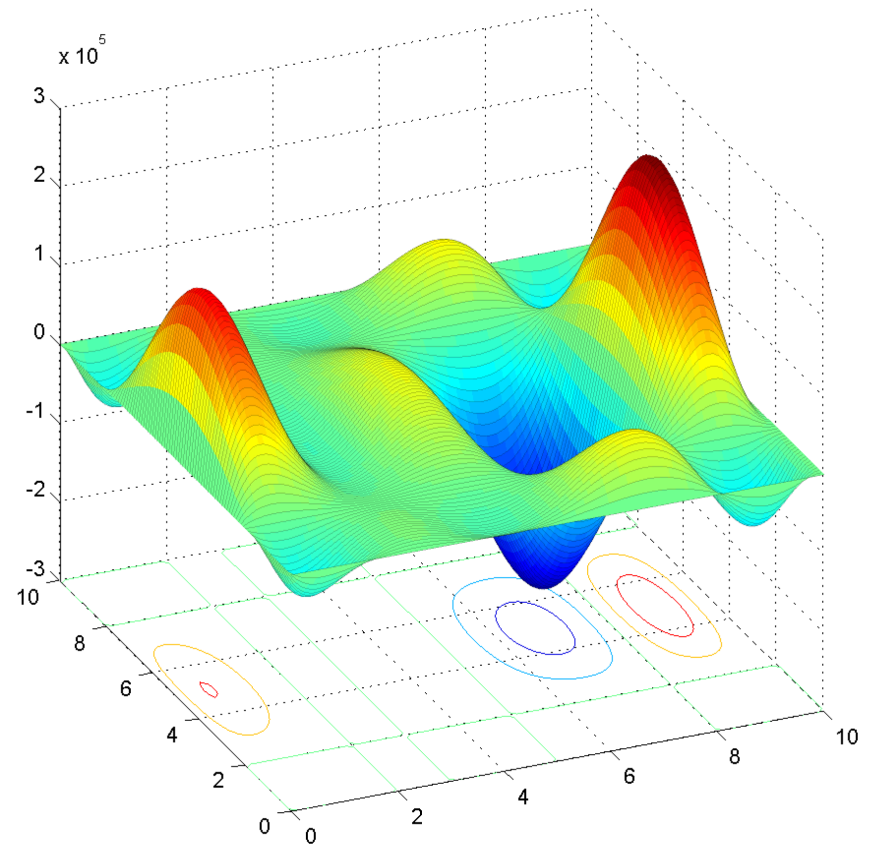
$$f(x) = c_0 + \sum_i c_i x_i$$

  - Bakery example

```
vars = ['flour_bread', 'flour_cookies']
c = [1.5, 2.5]; c0 = -50
>>> obj_fun([70, 30])
 130.0
```

Non-linear "smooth"

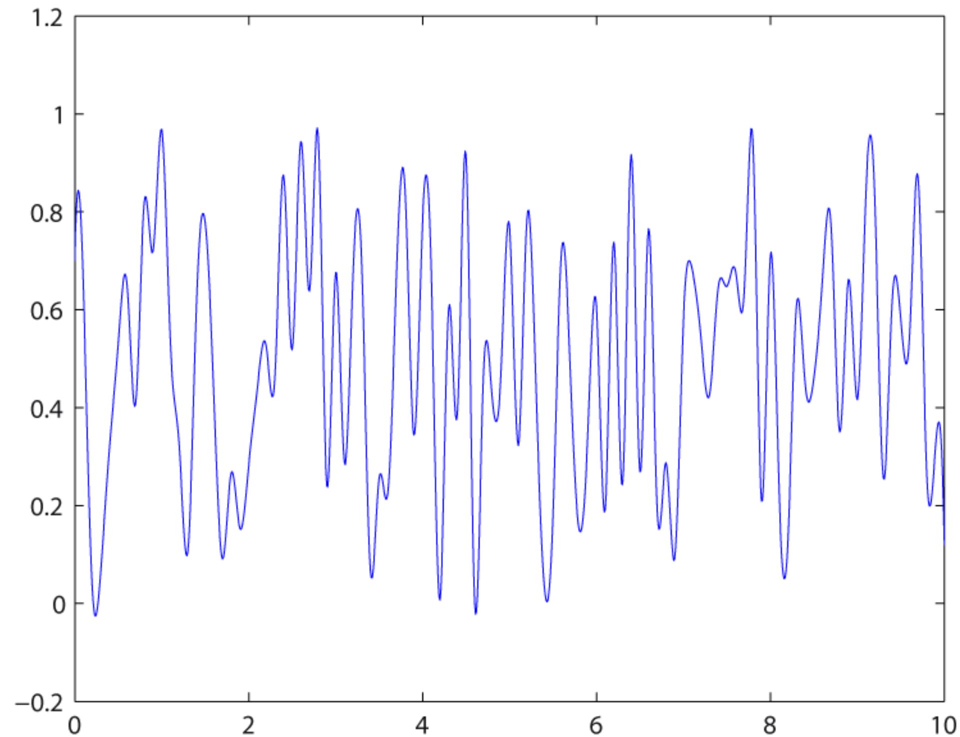




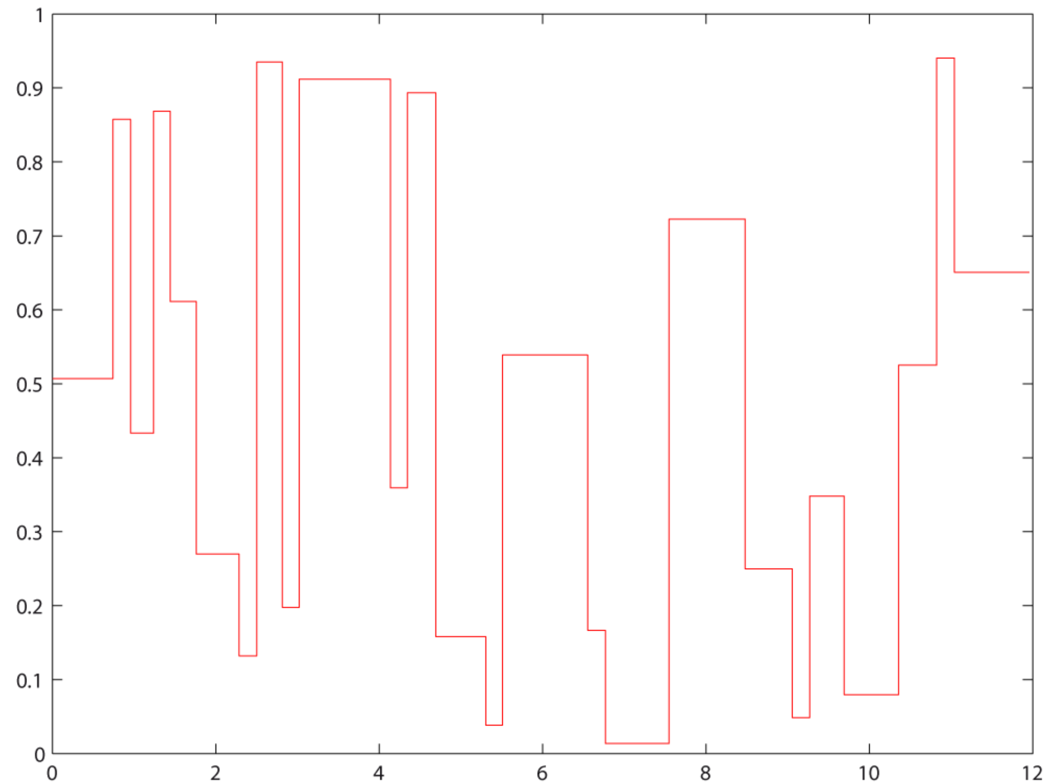
Non-linear "smooth"

- Non-linearity raises some issues
- Function must be "smooth" enough
  - You can stand on any point of the curved surface and assume you are on a plane
- No guarantee to find global optimum

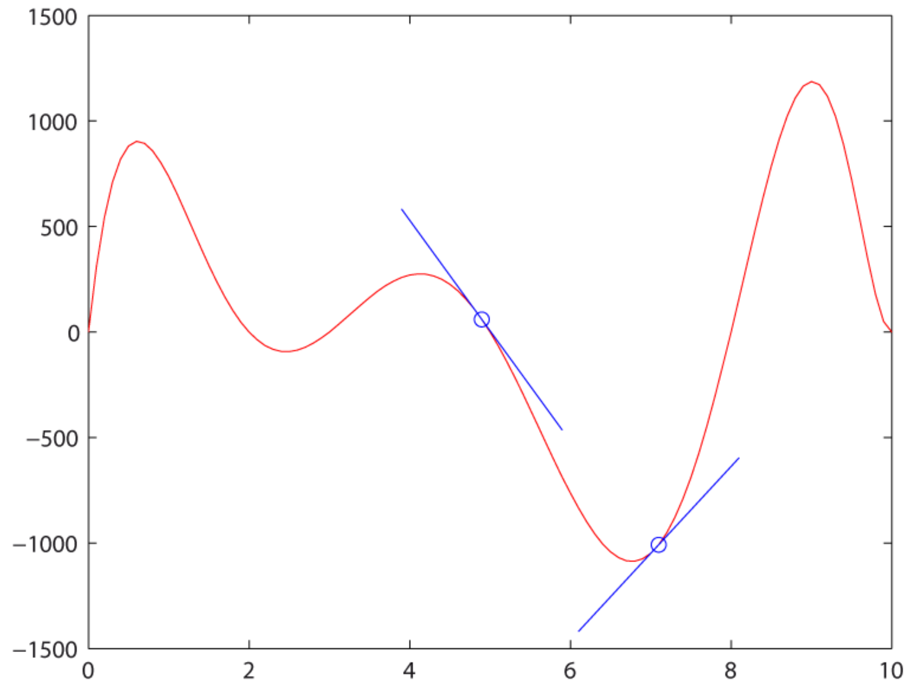
Non regular function



Non smooth function



Gradient descent



- Pick a point
- Pretend you are on a line
  - Can choose the optimal direction at each point
- Take steps until you fall in an optimum
  - Local optimum!

No smoothness?

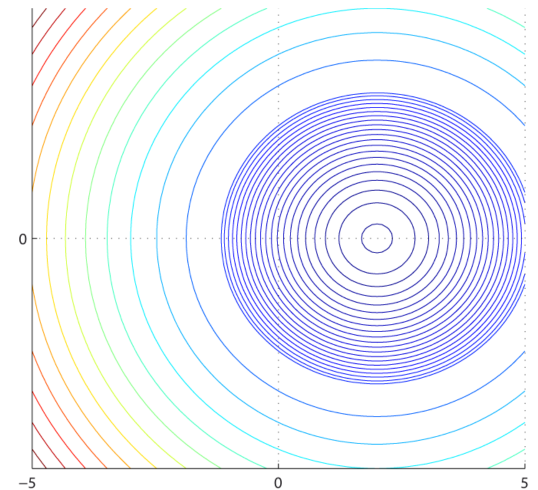
- Any other formulation for the problem?
  - Decrease model complexity
  - Put complexity in constraints
- Settle for an approximate solution
- Global search heuristics
  - Genetic algorithms are an example

SciPy optimization package

- Non-linear numerical function optimization
  - http://docs.scipy.org/doc/scipy/reference/optimize.html
- optimize.fmin(func, x0)
  - Unconstrained optimization
  - Finds the minimum of func(x) starting x with x0
  - x can be a vector, func must return a float
- Better algorithm for many variables: fmin_bfgs
- Algorithms for constrained optimization

Give me the code!

```
from scipy import optimize

def f(x):
    return x[0]**2 + (x[1]-2)**2

print optimize.fmin(f, [0,0])
```




```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 63
         Function evaluations: 120
[ -4.04997873e-06   2.00004847e+00]
```

Many variables

```
from scipy import optimize
from numpy import array
from random import uniform

n = 50
center = array([uniform(0, 10)
                for i in range(n)])

def f(x):
    return sum((x-center)**2)

optimum = optimize.fmin_bfgs(f, [0]*n)
print optimum - center
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 3
         Function evaluations: 312
         Gradient evaluations: 6
[  1.09850102e-08   1.00864010e-08  -3.44023343e-09   1.45686556e-08
   1.95572314e-09   1.63218594e-09   3.37632766e-09   1.44865293e-08
   1.52201984e-08   1.29321513e-08  -1.59485092e-09   1.23136292e-08
   1.62830638e-08  -3.52261820e-09  -5.56838653e-09   1.24844490e-08
   8.54035953e-09   1.55625752e-08   8.50658477e-09   1.08354117e-08
   1.62710236e-08  -3.44068374e-09  -5.63388847e-09   1.13670406e-09
   1.14938468e-08  -1.92982030e-09  -1.66206160e-09  -1.50014312e-09
   1.12226592e-08  -5.60273883e-09   3.07106607e-09   6.95412705e-09
  -4.58609706e-09   1.24199513e-08  -5.76112080e-09  -7.69927677e-10
  -6.56263044e-10  -6.08281736e-09   1.10073151e-08   3.35045769e-09
   1.25477273e-09   5.98582162e-09  -5.60075986e-10   1.00968194e-08
  -1.23273258e-09  -2.11921281e-09  -6.40983949e-09   8.99856101e-09
  -1.08943476e-09   1.55362354e-08]
```

Microeconomics model example

- Pricing problem for two products
  - Choose the prices that give the best profit
- Assume to know demand curves
  - From buyer's utility maximization
- Could be generalized
  - Many products
  - Complex demand curves
  - Complex costs and economies of scale
  - Beware of non smooth functions!

Microeconomics model example

```
def profit(p1, p2):
    x1 = demand1(p1, p2)
    x2 = demand2(p1, p2)
    income = x1*p1 + x2*p2
    cost = costf(x1, x2)
    return income - cost

def demand1(p1, p2):
    return 20*(20 - p1)

def demand2(p1, p2):
    return 40*(40 - p2)

def costf(x1, x2):
    c1 = 10*(x1**0.5)
    c2 = 20*(x2**0.5) + 2*x2
    cbase = 1000 + 5*(x1 + x2)
    return c1 + c2 + cbase
```

```
from scipy import optimize

def objective(x):
    return -profit(*x)

price = optimize.fmin(objective,
                      [1, 1])

print 'prices:', price
print 'amounts:‘
print demand1(*price), demand2(*price)
```

Microeconomics model results

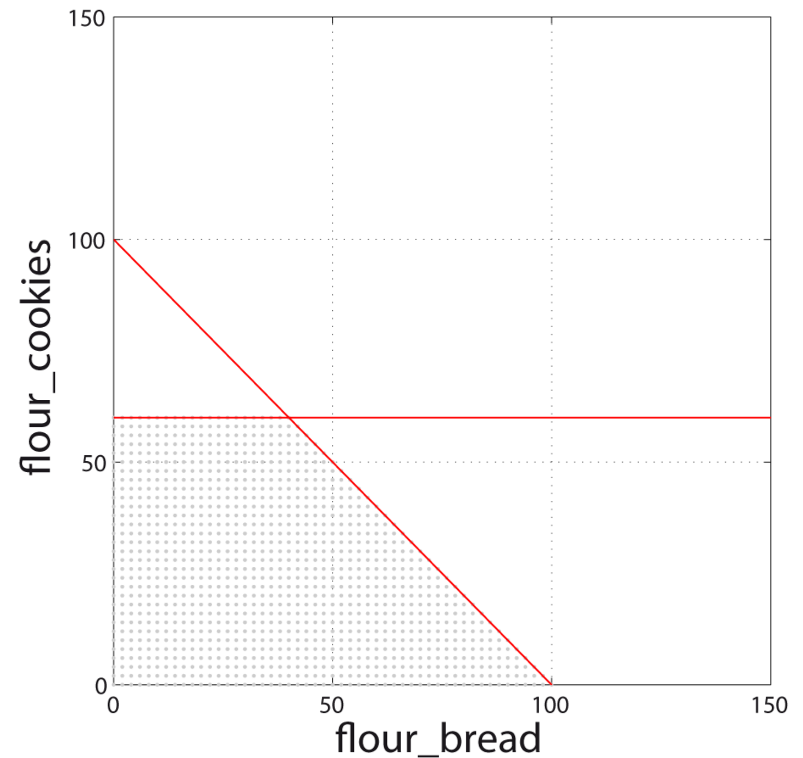
```
Optimization terminated successfully.
         Current function value: -10381.085398
         Iterations: 63
         Function evaluations: 122
prices: [ 12.7070282   23.69579991]
amounts: 145.859435961 652.168003412
```

Linear programming problem

- Linear objective function
  - Weighted sum of variables
- Constraints are linear inequalities
  - Weighted sum of some variables ≤ 0
- Variables can also be integer or binary
  - Mixed integer programming
  - NP-complete

Back to the bakery

- max $c_1x_1 + c_2x_2$
- subject to
  - $x_1 + x_2 \leq 100$
  - $x_2 \leq 60$



GNU Linear Programming Kit

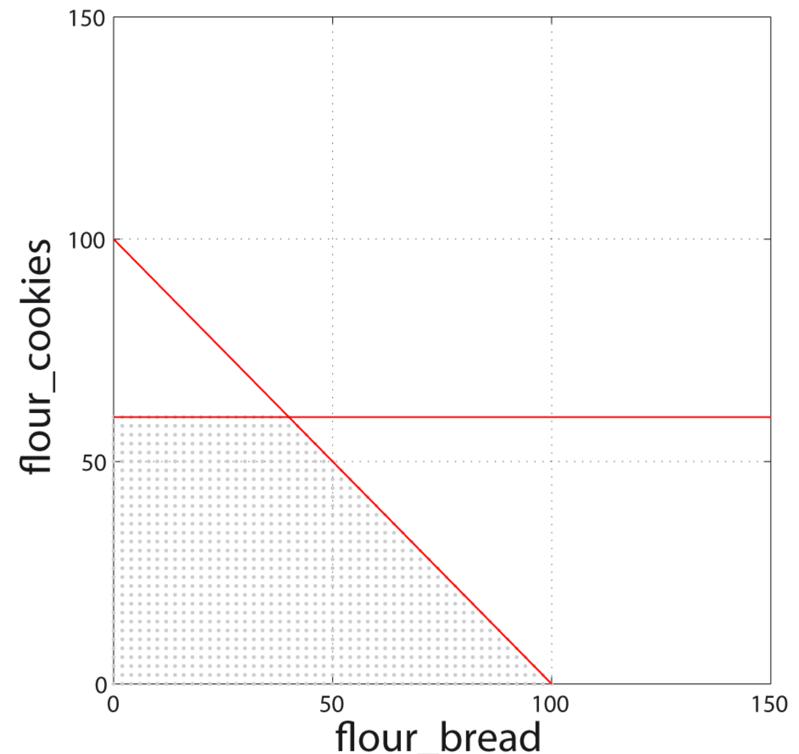
- C library for linear programming
  - Bindings for many languages
- Solver for large-scale problems
  - Linear programming
  - Mixed integer programming
- GNU MathProg modeling language
  - Subset of the AMPL proprietary language

PyMathProg

- Pure Python modeling language
  - http://pymprog.sourceforge.net/
  - Model described with natural Python operators
- Easy integration in the process workflow
- Interface to GLPK solver

PyMathProg bakery

```
import pymprog

# inits a model
m = pymprog.model('bakery')

# defines two named variables
x = m.var(range(2), 'x')

# objective function
m.max(1.5*x[0]+2.5*x[1])

# constraints
m.st(x[0]+x[1]<=100)
m.st(x[1]<=60)

m.solve()
print x
{0: x[0]=40.000000,
1: x[1]=60.000000}
```



Binary variables: Sudoku

```
p = pymprog.model("sudoku")
I, J, K = range(9), range(9), range(1, 10)
T = pymprog.iprod(I,J,K) #create Indice tuples
x = p.var(T, 'x', bool)
#x[i,j,k] = 1 means cell [i,j] is assigned number k
#assign pre-defined numbers using the "givens"
p.st( [ +x[i,j,k] == (1 if g[i][j] == k else 0)
       for (i,j,k) in T if g[i][j] > 0 ], 'given')

#each cell must be assigned exactly one number
p.st([sum(x[i,j,k] for k in K)==1 for i in I for j in J], 'cell')

#cells in the same row must be assigned distinct numbers
p.st([sum(x[i,j,k] for j in J)==1 for i in I for k in K], 'row')

#cells in the same column must be assigned distinct numbers
p.st([sum(x[i,j,k] for i in I)==1 for j in J for k in K], 'col')

#cells in the same region must be assigned distinct numbers
p.st([sum(x[i,j,k] for i in range(r,r+3) for j in range(c, c+3))==1
    for r in range(0,9,3) for c in range(0,9,3) for k in K],'reg')
```

CLV model example

- Marketing problem
- Potential and acquired customers
  - *Lifetime value* (expected cash flow)
  - Sensibility to promotions
- Limited budget for promotions
- Choose promotions as to maximize the total cash flow

CLV model example code

```
BUDGET = 5000

# model
m = pymprog.model('clv')

y = m.var(customers, 'y', bool)
x = m.var(c_p, 'x', bool)

m.max(sum(CLV[c]*y[c] for c in customers) -
       sum(C[p]*sum(x[c,p] for c in customers) for p in promos))

m.st(sum(x[c,p]*C[p] for c, p in c_p) <= BUDGET)
for c in customers:
    m.st(y[c] <= sum(x[c,p]*S[c][p] for p in promos))

m.solve()
```

CLV with activation costs

```
y = m.var(customers, 'y', bool)
x = m.var(c_p, 'x', bool)
z = m.var(promos, 'z', bool)

m.max(sum(CLV[c]*y[c] for c in customers) -
       sum(C[p]*sum(x[c,p] for c in customers) for p in promos) -
       sum(CP[p]*z[c] for p in promos))

m.st(sum(x[c,p]*C[p] for c, p in c_p)<=BUDGET)
m.st([([y[c] <= sum(x[c,p]*S[c][p] for p in promos))
       for c in customers])
m.st([(z[c] <= sum(x[c,p] for c in customers) for p in promos])
```

Genetic algorithms

«Take a bunch of random solutions, mix them randomly, repeat an undefined number of times, get the optimum»

Genetic algorithms

- Biological metaphor for an optimization technique
  - Individual = proposed solution
  - Gene = decision variable
  - Fitness = objective function
- Natural selection-like
  - The most fit individuals breed
  - The children are similar to the parents
  - Every generation better than the previous one

Individuals and genomes

- Genetic representation of the solution
  - How do the genes map to the solution?
  - Historically as a string of bits
  - Many alternatives: numbers, lists, graphs
- Genetic operators
  - How are children different from parents?
  - *Crossover* and *mutation*
- Fitness function
  - Just like the objective function

The simple genetic algorithm

- Randomly generate a population of individuals
- Repeat until I get a good enough solution
  - Select the individuals with highest fitness
  - Discard the least-fit ones
  - Generate children from crossover and mutation
- Termination condition not always well defined

Some genome encodings

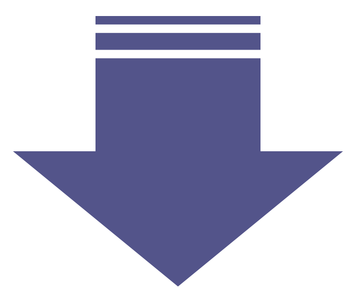
- Binary string encoding
  - Fixed length, each bit maps to a binary feature
  - Crossover splits the string
  - Mutation inverts a random bit
  - Example: knapsack problem
- Sequence encoding
  - Fixed elements, the order maps to the solution
  - Crossover splits the sequence
  - Mutation exchanges two elements
  - Example: travelling salesman problem

Single point crossover

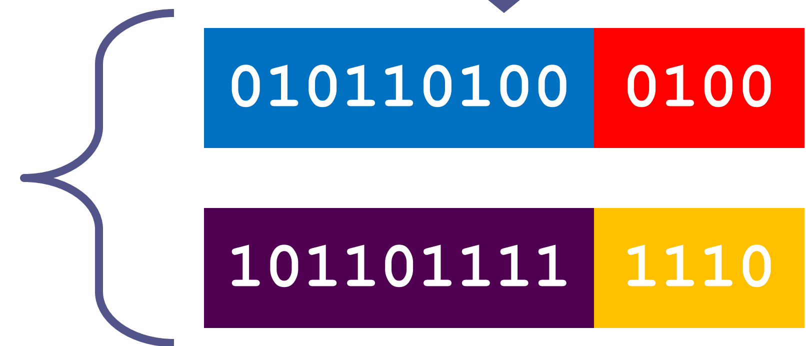
Parent 1

Parent 2

Crossover point



Offspring



Example: knapsack problem

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT

~ APPETIZERS ~

MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80

~ SANDWICHES ~

BARBECUE 6.55

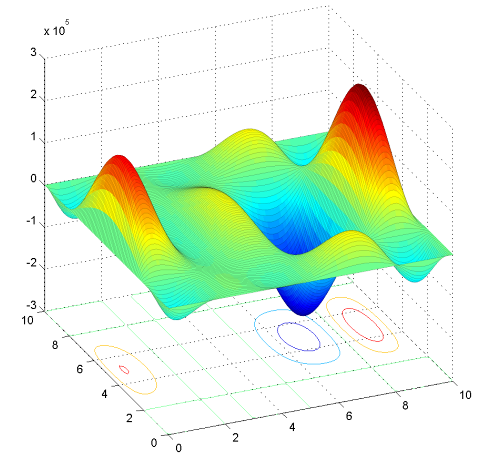
Knapsack problem

- Binary string encoding
  - One bit for each item
- Fitness = value of the knapsack
  - Zero if not fitting (is there a better way?)
- Single point crossover
- Simple mutation

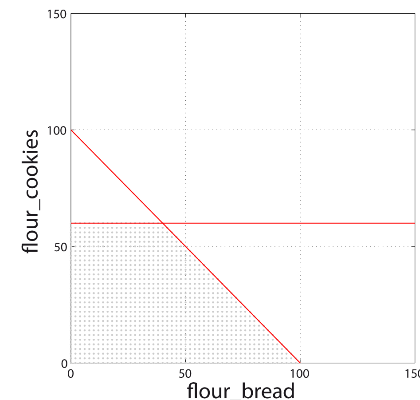
GAs pros and cons

- Very easy to implement
- Quite easy to get right
- Good at problems where exact methods fail
- Relatively very slow
- Don't scale well
- Not well defined end condition

Summing up



- Non-linear programming
  - scipy.optimization
- Linear and integer programming
  - GLPK and PyMathProg
- Global search heuristics
  - Genetic algorithms
- Get updates!
  - http://daviderizzo.net/blog/



Questions?