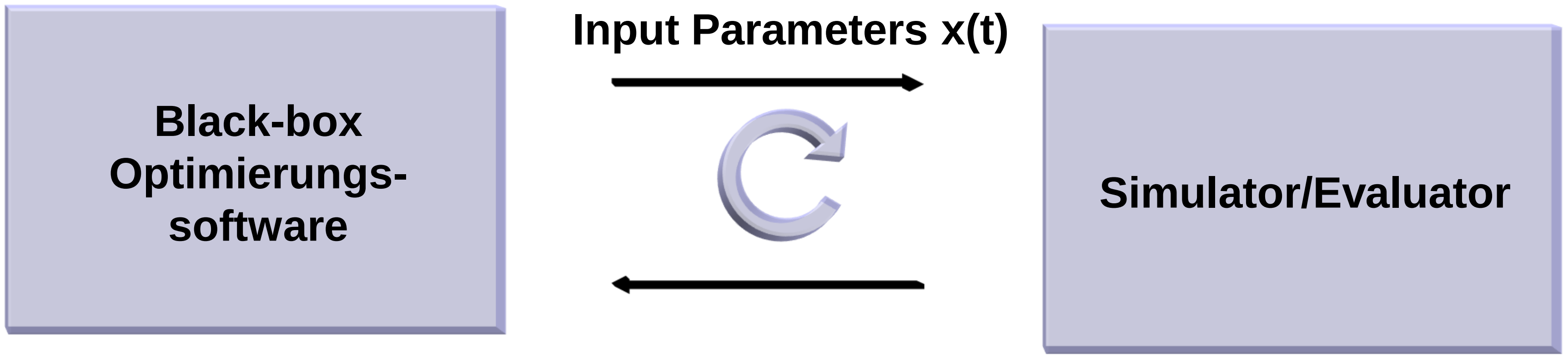
Unit: Multiobjective Hill-climbing Methods

- Steepest Descent Method
- Newton Raphson Methods
- Quasi Newton Methods (BFGS, DFP)
- Discrete Local Search
- Local Search for Pareto front approximation
- Set-Oriented Gradient and Newton's Method for Multiobjective Optimization

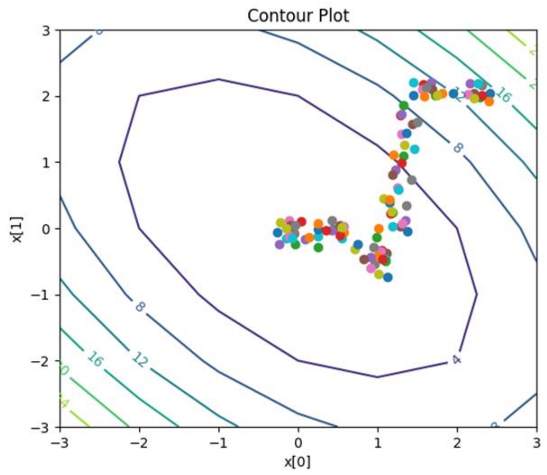
Universiteit Leiden

Basic strategy in Black-box optimization



$$\boldsymbol{f}(\boldsymbol{x}(\boldsymbol{t})) + \boldsymbol{penalty}(\boldsymbol{r}(\boldsymbol{x}(\boldsymbol{t})))$$

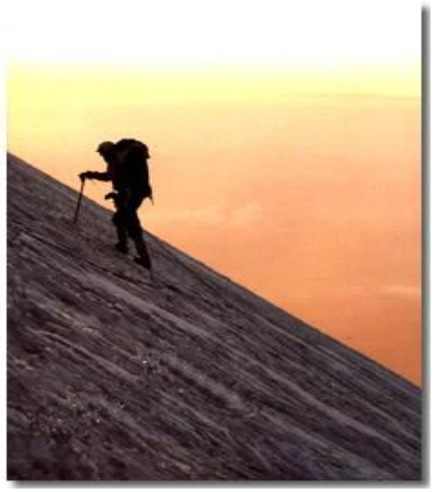
1. Stochastic Hillclimbing
2. Gradient Descent
3. Newton Method
4. Simulated Annealing
5. Evolutionary Algorithm
6. Bayesian Optimization
7. Etc.



Hill-climbing Methods for Single-Objective Optimization

Path oriented (hill climbers) can be defined by a general iterative formula:

$$\mathbf{x_{t+1}} = \mathbf{x_t} + \sigma_\mathbf{t} \mathbf{d_t}$$



$\mathbf{x_t}$: Current search point

σ_t: Step size

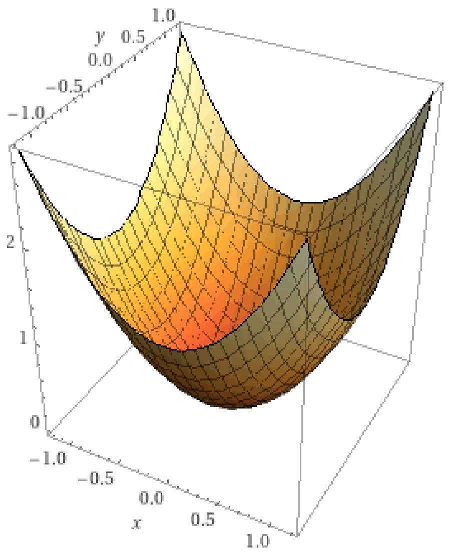
$\mathbf{d}_t$: Current search direction

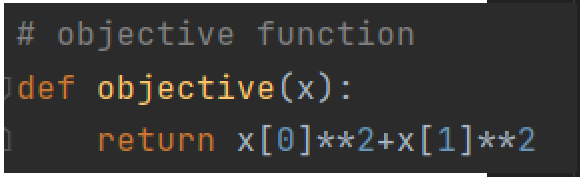
Hill-climbers generates a sequence of points $\{\mathbf{x}_t\}_{t=1,2,\ldots}$ that gradually improve the value of the objective function.

Picture from wikipedia commons, hillclmbing

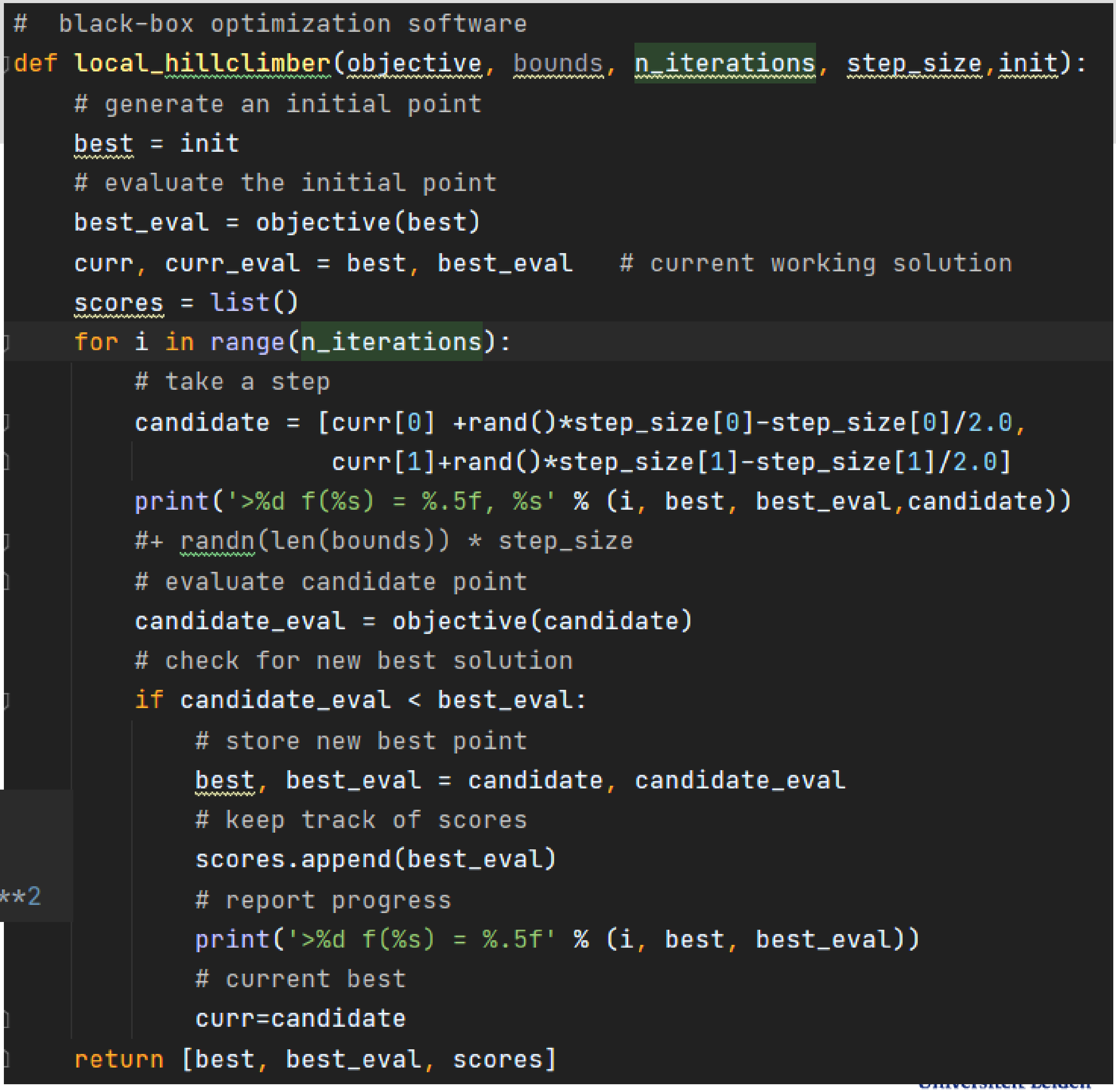
Universiteit Leiden

Simple 2-D stochastic hillclimber*





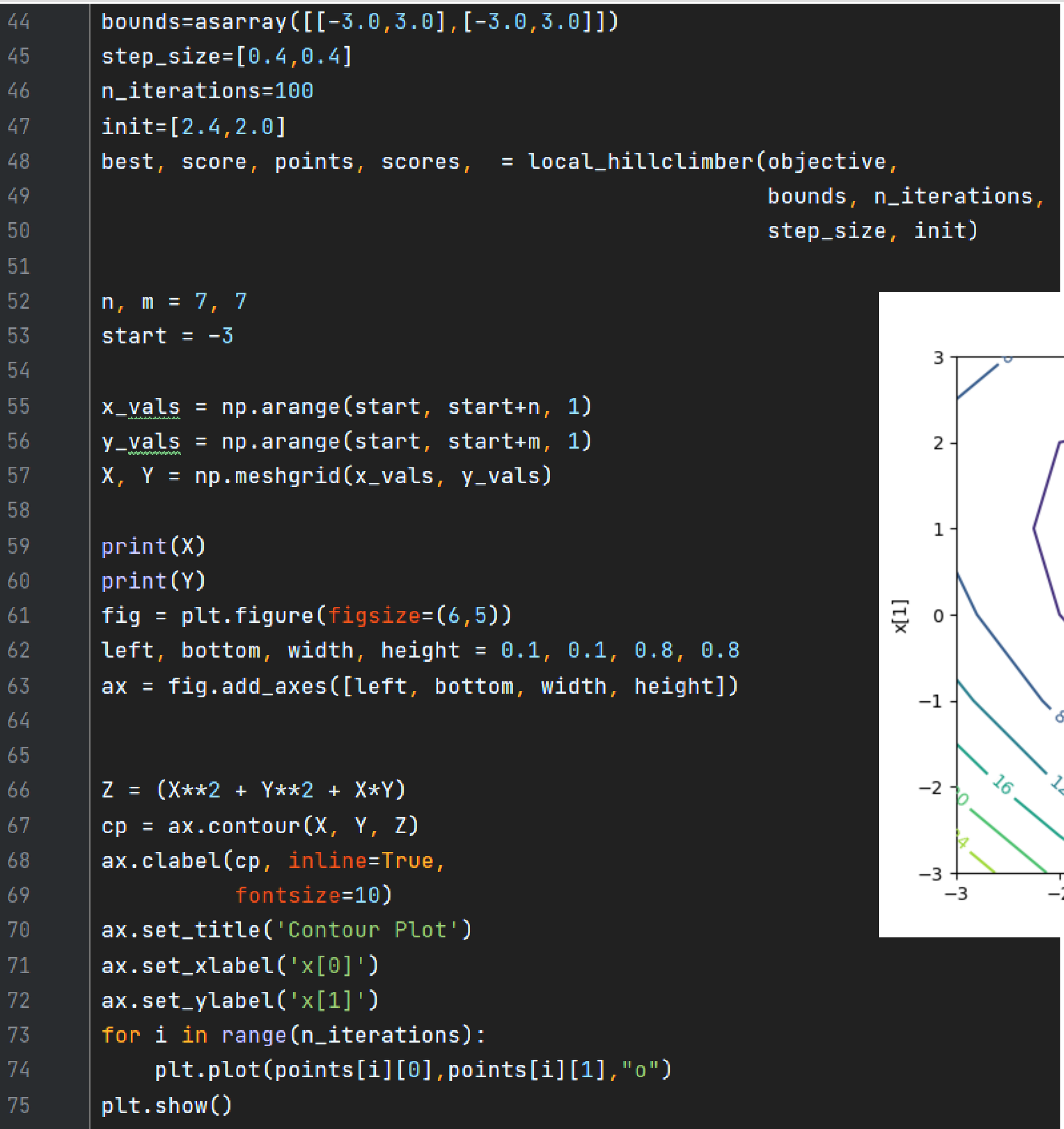
```
# objective function
def objective(x):
    return x[0]**2+x[1]**2
```



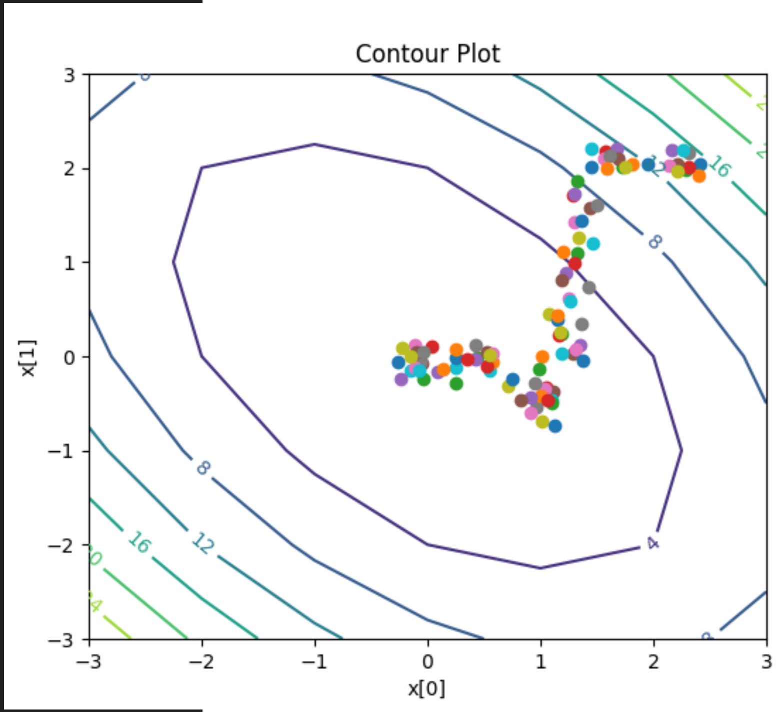
```
#  black-box optimization software
def local_hillclimber(objective, bounds, n_iterations, step_size,init):
    # generate an initial point
    best = init
    # evaluate the initial point
    best_eval = objective(best)
    curr, curr_eval = best, best_eval   # current working solution
    scores = list()
    for i in range(n_iterations):
        # take a step
        candidate = [curr[0] +rand()*step_size[0]-step_size[0]/2.0,
                     curr[1]+rand()*step_size[1]-step_size[1]/2.0]
        print('>%d f(%s) = %.5f, %s' % (i, best, best_eval,candidate))
        #+ randn(len(bounds)) * step_size
        # evaluate candidate point
        candidate_eval = objective(candidate)
        # check for new best solution
        if candidate_eval < best_eval:
            # store new best point
            best, best_eval = candidate, candidate_eval
            # keep track of scores
            scores.append(best_eval)
            # report progress
            print('>%d f(%s) = %.5f' % (i, best, best_eval))
            # current best
            curr=candidate
    return [best, best_eval, scores]
```

*Sources on brightspace couse materials

Plotting the history

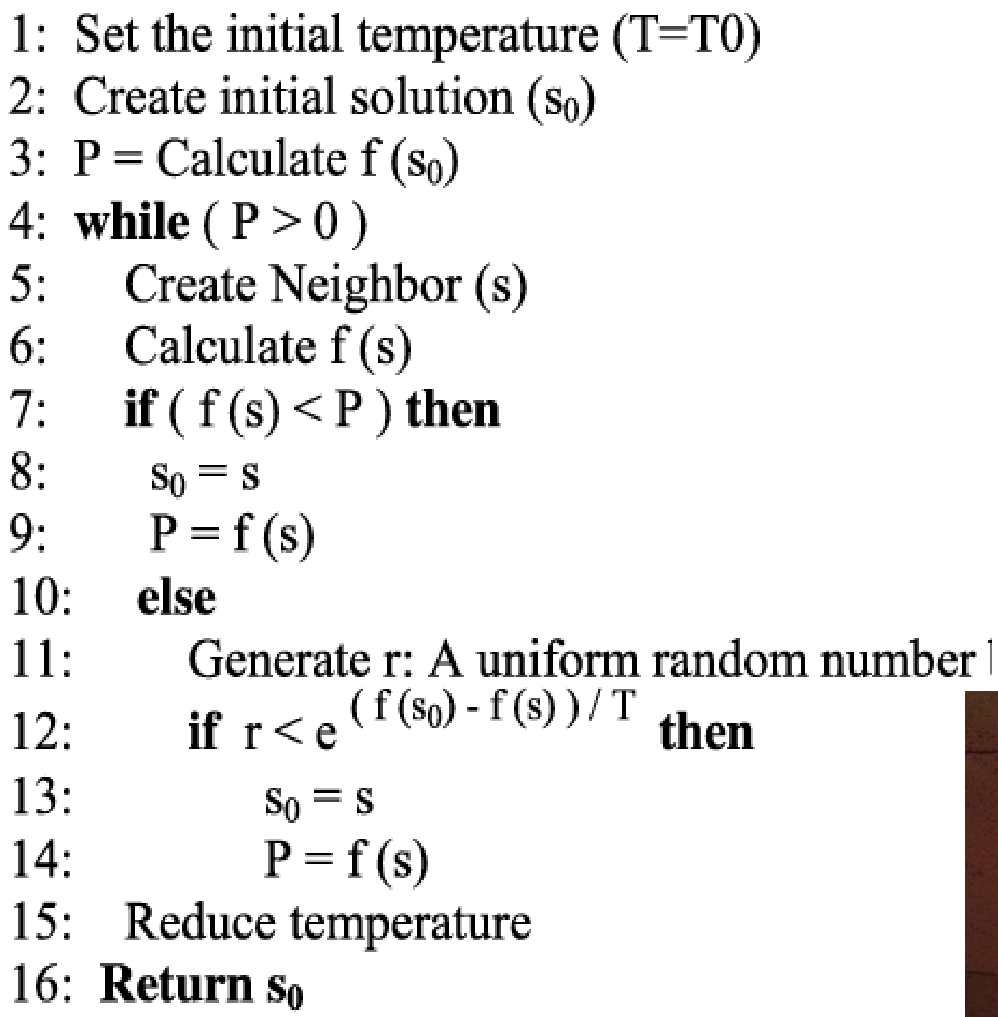


```python
bounds=asarray([[-3.0,3.0],[-3.0,3.0]])
step_size=[0.4,0.4]
n_iterations=100
init=[2.4,2.0]
best, score, points, scores,  = local_hillclimber(objective,
                                                  bounds, n_iterations,
                                                  step_size, init)

n, m = 7, 7
start = -3

x_vals = np.arange(start, start+n, 1)
y_vals = np.arange(start, start+m, 1)
X, Y = np.meshgrid(x_vals, y_vals)

print(X)
print(Y)
fig = plt.figure(figsize=(6,5))
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
ax = fig.add_axes([left, bottom, width, height])


Z = (X**2 + Y**2 + X*Y)
cp = ax.contour(X, Y, Z)
ax.clabel(cp, inline=True,
          fontsize=10)
ax.set_title('Contour Plot')
ax.set_xlabel('x[0]')
ax.set_ylabel('x[1]')
for i in range(n_iterations):
    plt.plot(points[i][0],points[i][1],"o")
plt.show()
```



Simulated Annealing

Kirkpatrick, S. Optimization by simulated annealing: Quantitative studies. *J Stat Phys* **34,** 975–986 (1984). https://doi.org/10.1007/BF01009452

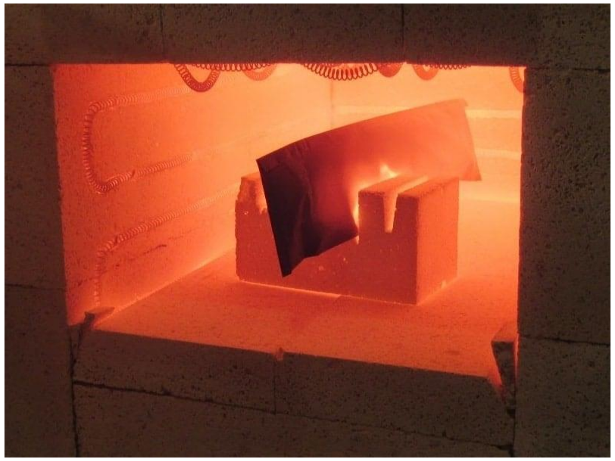


```
Set the initial temperature (T=T0)
Create initial solution ($s_0$)
P = Calculate f ($s_0$)
while ( P > 0 )
    Create Neighbor (s)
    Calculate f (s)
    if ( f (s) < P ) then
        $s_0$ = s
        P = f (s)
    else
        Generate r: A uniform random number
        if $r < e^{(f(s_0) - f(s))/T}$ then
            $s_0$ = s
            P = f (s)
    Reduce temperature
Return $s_0$
```

Stochastic Hillclimbing inspired by Annealing process in crystals.

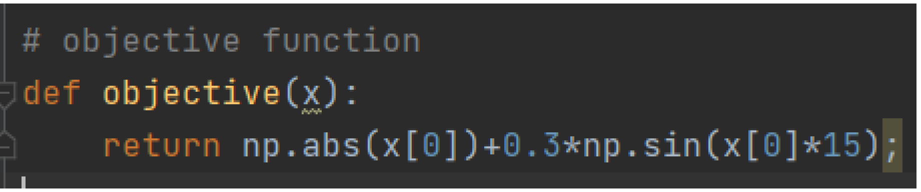
Simulated Annealing can always accepts improvements, but also worse solutions with some probability.

In order to get to global optima one might have to accept steps to get worse temporarily



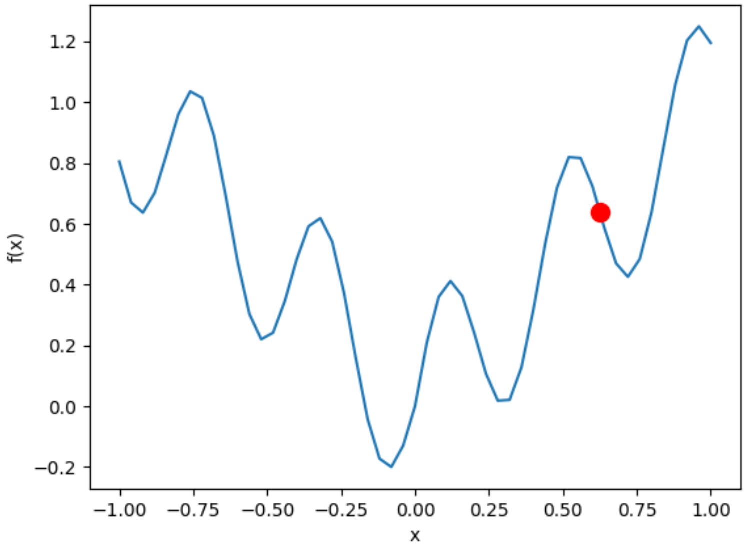
https://makeitfrommetal.com/beginners-guide-on-how-to-anneal-steel/

Simulated Annealing



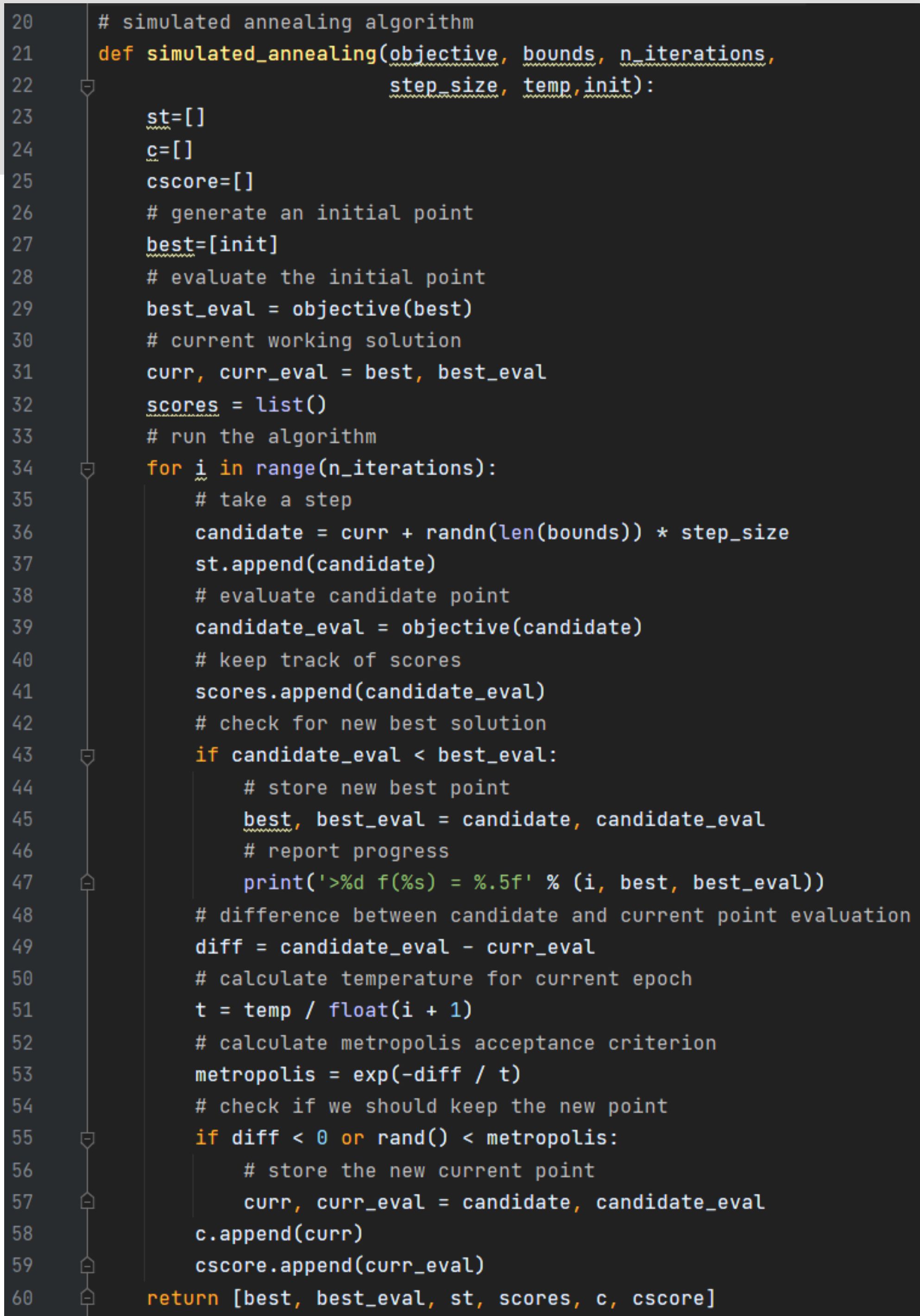
```python
# objective function
def objective(x):
    return np.abs(x[0])+0.3*np.sin(x[0]*15);
```

https://trinket.io/python3/b22300f21e

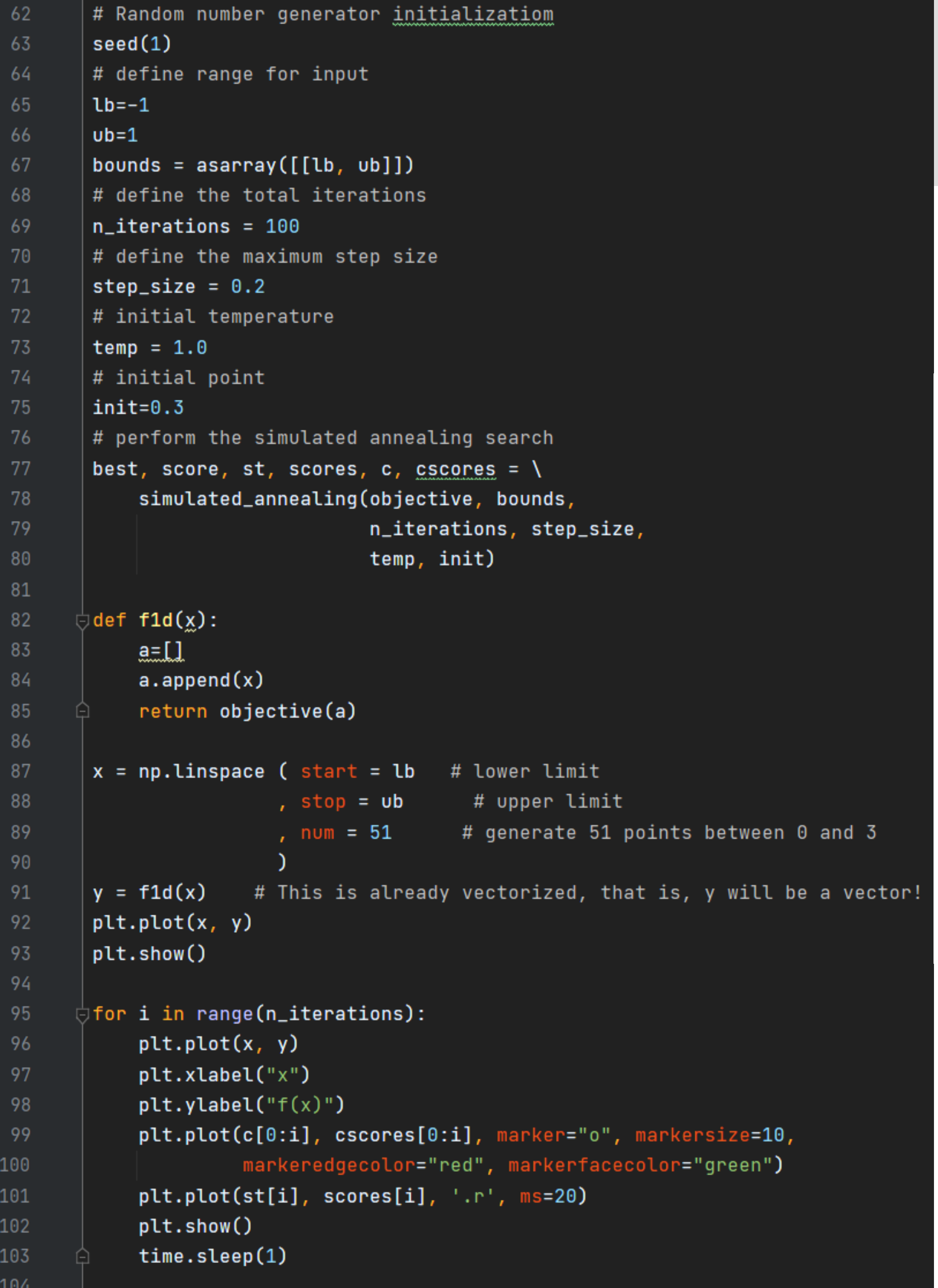


1-D Objective Function with local optima
$f(x) = |x| + 0.3\sin(15\,x), x \in [-1,1]$

Simulated Annealing can be implemented in 2-D and N-D .



```python
# simulated annealing algorithm
def simulated_annealing(objective, bounds, n_iterations,
                        step_size, temp,init):
    st=[]
    c=[]
    cscore=[]
    # generate an initial point
    best=[init]
    # evaluate the initial point
    best_eval = objective(best)
    # current working solution
    curr, curr_eval = best, best_eval
    scores = list()
    # run the algorithm
    for i in range(n_iterations):
        # take a step
        candidate = curr + randn(len(bounds)) * step_size
        st.append(candidate)
        # evaluate candidate point
        candidate_eval = objective(candidate)
        # keep track of scores
        scores.append(candidate_eval)
        # check for new best solution
        if candidate_eval < best_eval:
            # store new best point
            best, best_eval = candidate, candidate_eval
            # report progress
            print('>%d f(%s) = %.5f' % (i, best, best_eval))
        # difference between candidate and current point evaluation
        diff = candidate_eval - curr_eval
        # calculate temperature for current epoch
        t = temp / float(i + 1)
        # calculate metropolis acceptance criterion
        metropolis = exp(-diff / t)
        # check if we should keep the new point
        if diff < 0 or rand() < metropolis:
            # store the new current point
            curr, curr_eval = candidate, candidate_eval
        c.append(curr)
        cscore.append(curr_eval)
    return [best, best_eval, st, scores, c, cscore]
```

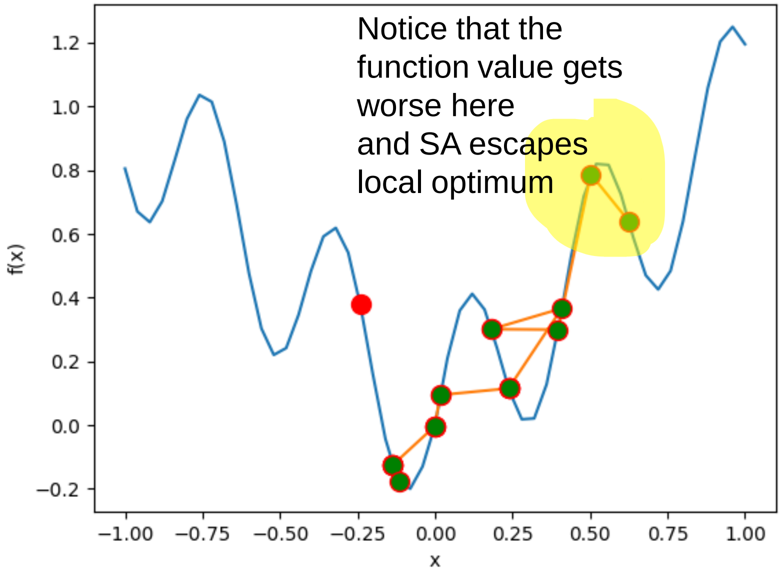


```
# Random number generator initializatiom
seed(1)
# define range for input
lb=-1
ub=1
bounds = asarray([[lb, ub]])
# define the total iterations
n_iterations = 100
# define the maximum step size
step_size = 0.2
# initial temperature
temp = 1.0
# initial point
init=0.3
# perform the simulated annealing search
best, score, st, scores, c, cscores = \
    simulated_annealing(objective, bounds,
                        n_iterations, step_size,
                        temp, init)

def f1d(x):
    a=[]
    a.append(x)
    return objective(a)

x = np.linspace ( start = lb   # lower limit
                , stop = ub      # upper limit
                , num = 51       # generate 51 points between 0 and 3
                )
y = f1d(x)    # This is already vectorized, that is, y will be a vector!
plt.plot(x, y)
plt.show()

for i in range(n_iterations):
    plt.plot(x, y)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.plot(c[0:i], cscores[0:i], marker="o", markersize=10,
             markeredgecolor="red", markerfacecolor="green")
    plt.plot(st[i], scores[i], '.r', ms=20)
    plt.show()
    time.sleep(1)
```

Simulated Annealing

- Plot shows the function and linked successful moves

https://trinket.io/python3/b22300f21e
(© by M. Emmerich, interactive in browser)



- Homework:

Optimize design with N-D Simulated annealing

Steepest Descent Method

Gradient based methods make use of the gradient of the objective function:

$$\nabla f(\mathbf{x}) = (\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n})$$

The most straightforward gradient based minimization method is the steepest descent method:

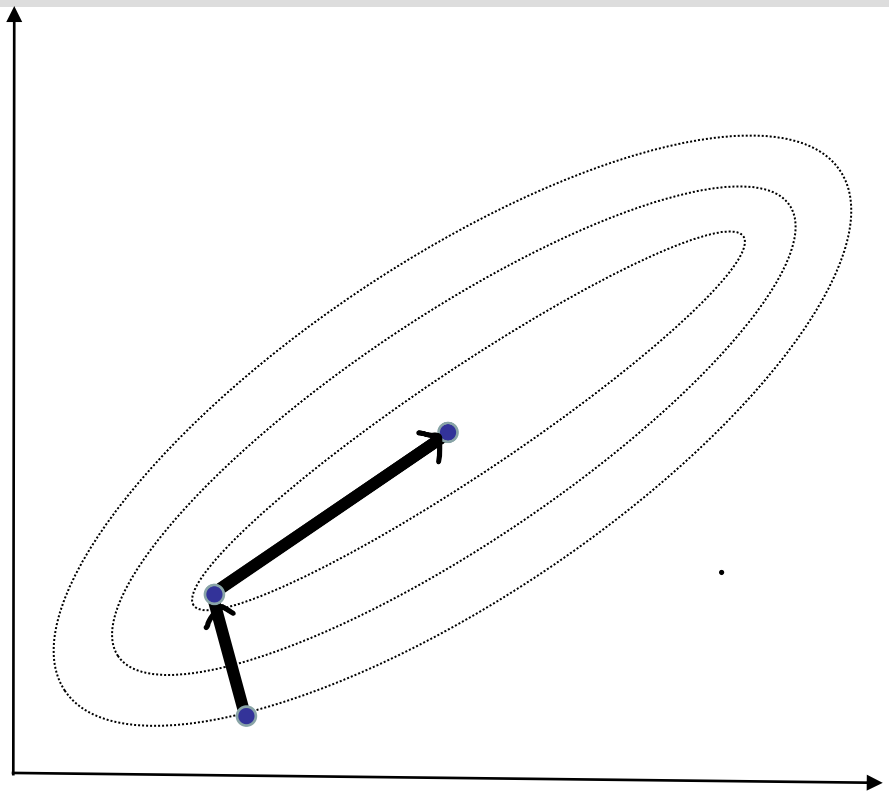
$$\mathbf{d}_t = -\frac{\nabla f(\mathbf{x}_\mathrm{t})}{||\nabla f(\mathbf{x}_\mathrm{t})||}$$

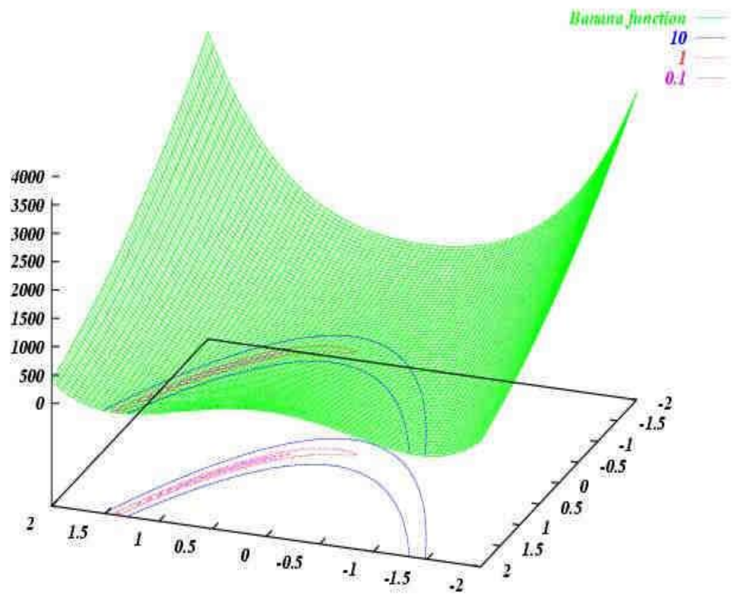
The step size is then chosen as the result (so-called relative minimum) of an one dimensional optimization in this direction (so-called line search):

$$\sigma_t = \text{arg min}_{\sigma>0}\{f(\mathbf{x}_\mathrm{t} + \sigma \mathbf{d}_\mathrm{t})\}$$

Universiteit Leiden

Discussion of steepest descent





At this so-called Banana Function The steepest descent method will be trapped because the relative optima are infinitely close to each other

Remark: Gradients can be approximated by simple numerical methods (e.g. forward differences) if the function is a black-box. It requires n evaluations of perturbations of the vector components: $\frac{\partial f}{\partial x_i} \approx (f(x_1, \dots x_i + \Delta, \dots x_n) - f(x))/\Delta$ for some small positive Δ. In case of box costraints use projected gradient (truncate all coordinates outside the range to the exceeded range boundary)

Newton Raphson Optimization Strategy

Second order methods build a local quadratic model of the objective function by means of its Taylor expansion

$$f(\mathbf{x}) \approx \mathbf{f}(\mathbf{x_t}) + \nabla \mathbf{f}(\mathbf{x_t})(\mathbf{x} - \mathbf{x_t})^{\mathbf{T}} + \frac{1}{2}(\mathbf{x} - \mathbf{x_t})^{\mathbf{T}} \nabla^2 \mathbf{f}(\mathbf{x_t})(\mathbf{x} - \mathbf{x_t})^{\mathbf{T}}.$$

Then, $\mathbf{x_{t+1}}$ is set to the optimum of this approximation. This can be determined by using Newton's method in order to find the zero of the gradient, supposed that the Hessian matrix is positive definite (i. e. the problem is strictly convex):

Newton Raphson optimization strategy:

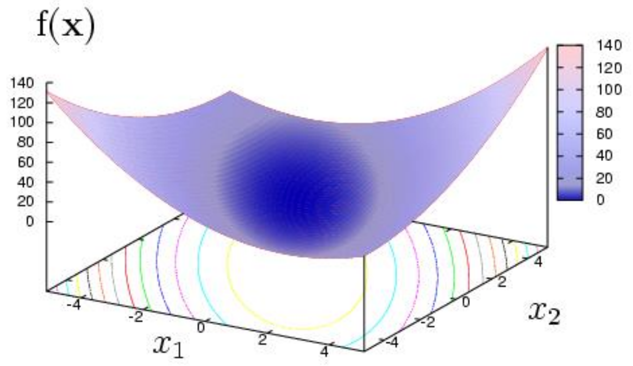
$$\mathbf{x_{t+1}} = \mathbf{x_t} - \nabla \mathbf{f}(\mathbf{x_t})[\nabla^2 \mathbf{f}(\mathbf{x_t})]^{-1}$$

Variable Metric Methods

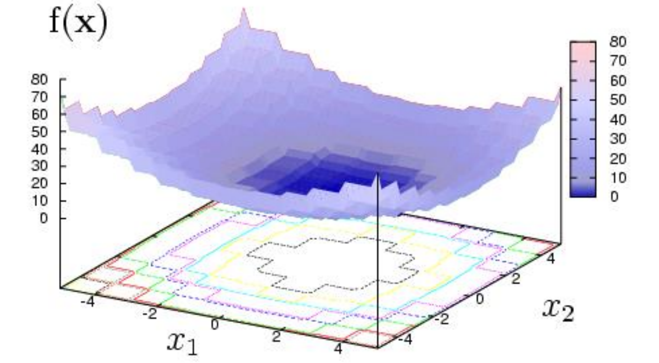
- Basic idea is to approximate inverse Hessian
- Two popular variants:
  - Davidon Fletcher Powell Method (DFPM)
  - Broyden-Fletcher-Goldfarb-Shanno Method (BFGS)
- In each step the gradient is computed ($O(d)$, d is number of variables)
- The gradients of different time-steps are combined to dynamically update an approximation of the Hessian Matrix
- Similar idea in evolutionary algorithms: Covariance Matrix Adaptation Evolution Strategy (here the Covariance Matrix adapts to the inverse Hessian)

Universiteit Leiden

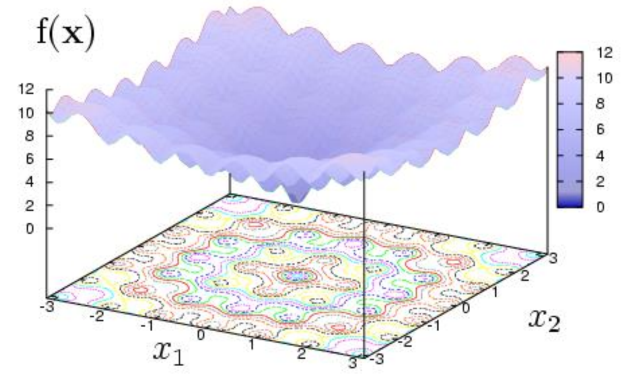
Hill climbing methods: Limitations and Capabilities

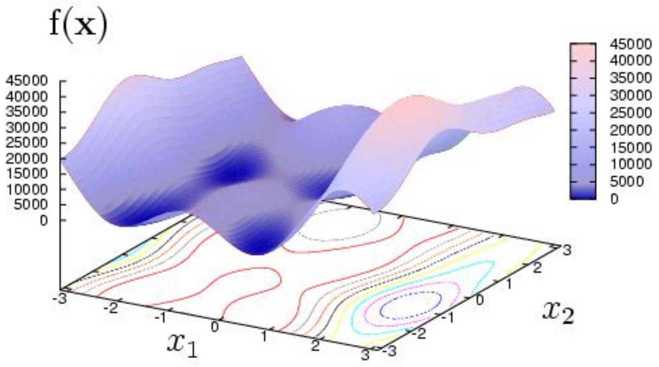


Convex − quadratic functions:
Gradient − based work efficiently



Discontinuous functions:
fail, gradient of zero/infinity

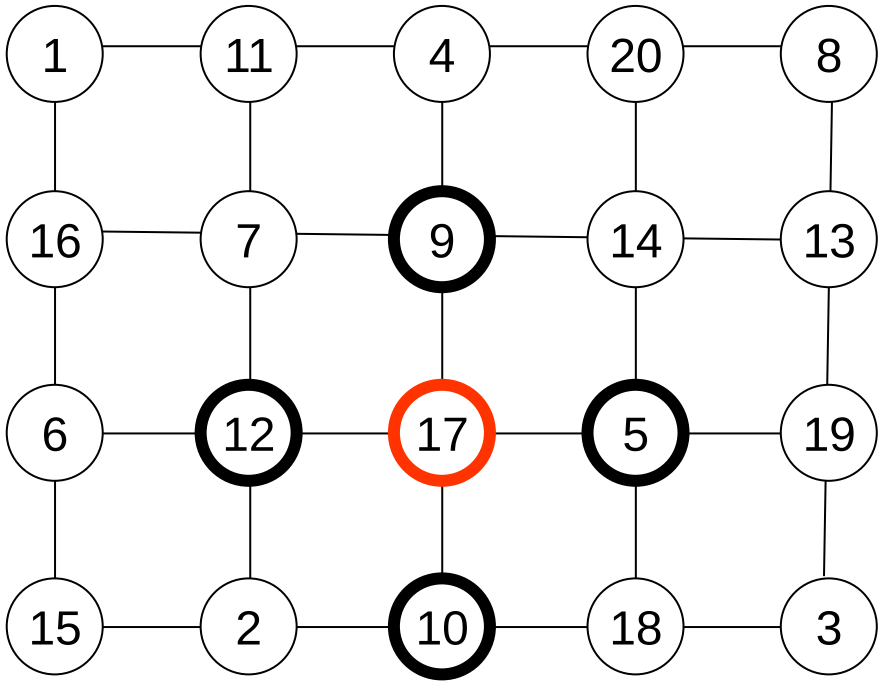




Multimodal functions: *fail, convergence to local optimum if bad starting point*
=> ***Global optimization*** *methods are needed*! → *next chapter*

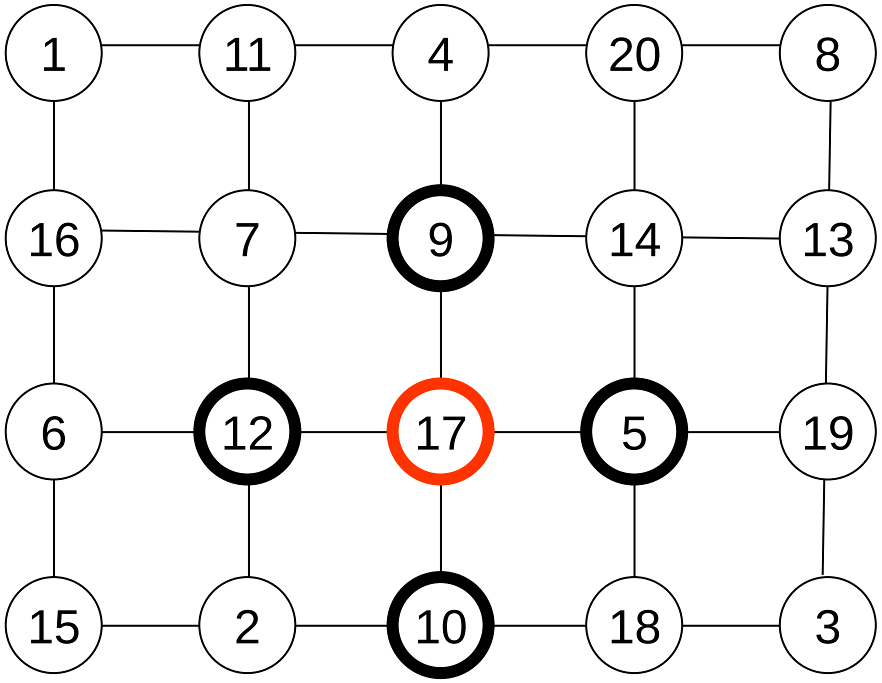
Hill-climbing in discrete spaces: Local Search

- Heuristic 1: Exhaustive neighborhood search
  - Look at all neighbors
  - Go to best neighbor
- Heuristic 2: Greedy neighborhood search
  - Look at neigbors one by one
  - Change current position to neigbor as soon as improvement was found
- Both heuristics converge to local optimaö Neighorhood definition crucial
- Further reading. Iterative Local Search (Hoos, Stützle, Springer 2007) provides a good algorithmic framework and taxonomy.

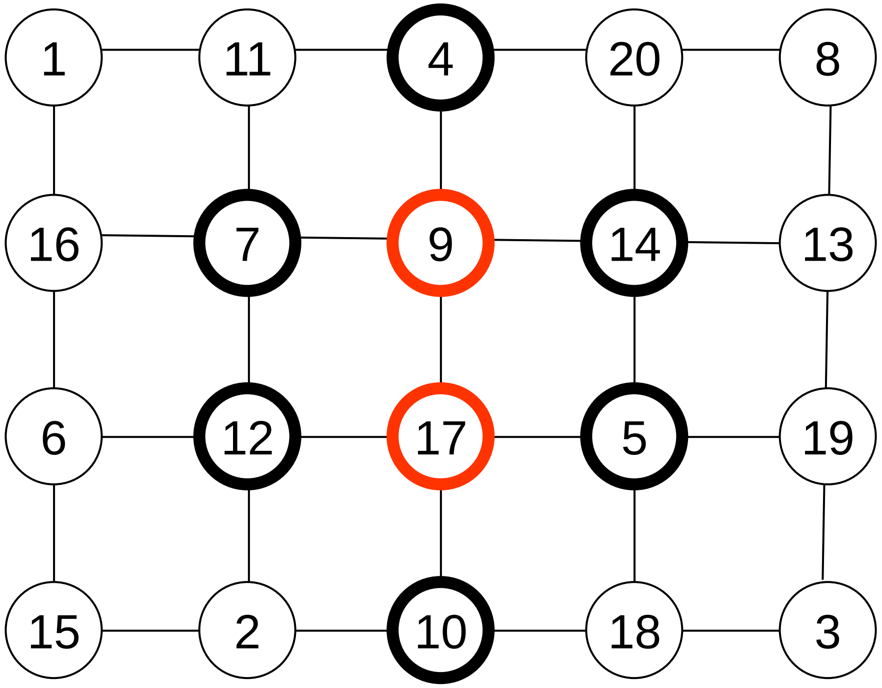


Universiteit Leiden

Hill-climbing in discrete spaces: Local Search

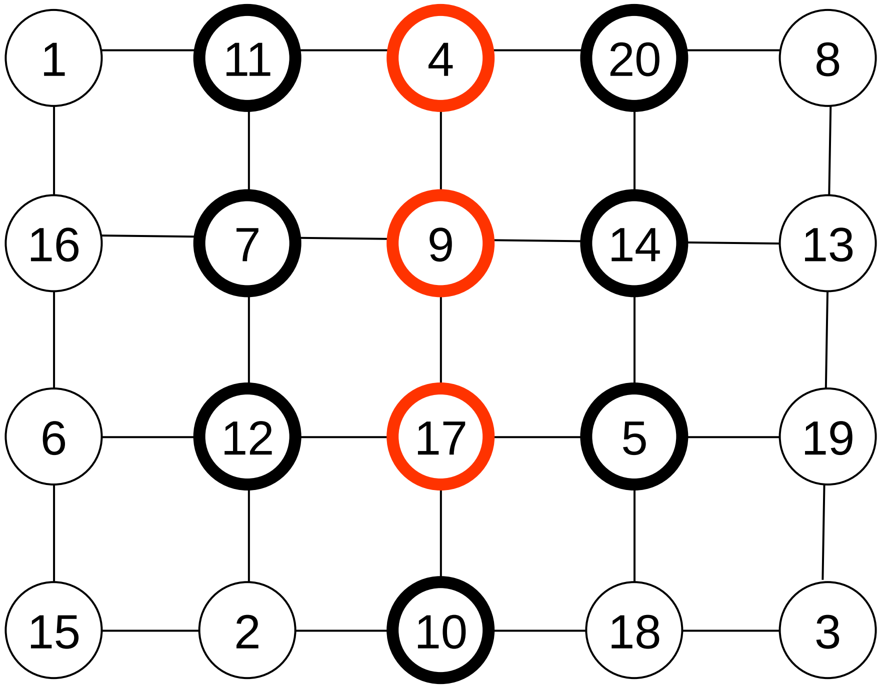


Hill-climbing in discrete spaces



Universiteit Leiden

Hill-climbing in discrete spaces

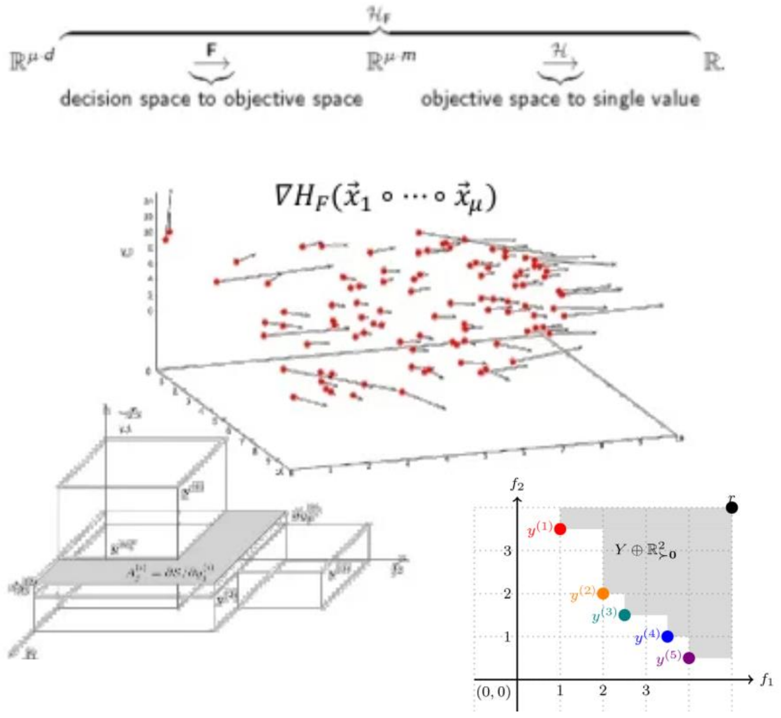


Universiteit Leiden

Going Multiobjective: Three Strategies

- Three strategies to apply hillclimbing method in Multi-Objective Optimization
  - Strategy 1: Repeated use of **Single-point method** (see lecture)
  - Strategy 2: **Memetic-algorithms** – Combine global population based search (e.g. NSGA-II_ with local search for the improvement of single points with single-point methods.
  - Strategy 3: **Set-Scalarization**: Treat the population as a single vector mapped to a Pareto front quality indicator (e.g. hypervolume indicator)

$$\psi: P = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\mu)}\} \subset_\mu \mathbb{R}^d \mapsto (\boldsymbol{x}^{(1)} \circ \cdots \circ \boldsymbol{x}^{(\mu)}) \in \mathbb{R}^{d\mu}$$

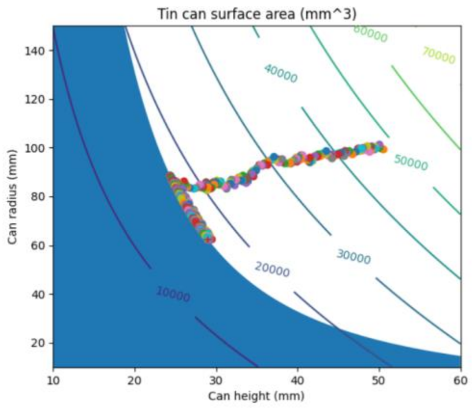

$\mathcal{H}_F$
$\mathbb{R}^{\mu \cdot d}$
$\mathbf{F}$
decision space to objective space
$\mathbb{R}^{\mu \cdot m}$
$\mathcal{H}$
objective space to single value
$\mathbb{R}$
$\nabla H_F(\vec{x}_1 \circ \cdots \circ \vec{x}_\mu)$
f_2
f_1
$y^{(1)}$
$y^{(2)}$
$y^{(3)}$
$y^{(4)}$
$y^{(5)}$
$Y \oplus \mathbb{R}^2_{\succ 0}$
$(0, 0)$

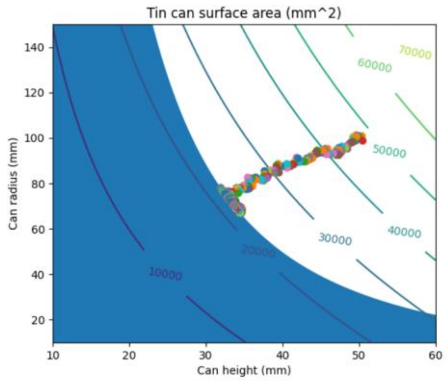
Set Scalarization: Idea is to perform hillclimbing on population vectors to maximize the set indicator (e.g. hypervolume indicator).

Computation of Pareto front with local search method

- Set the target volume to differer levels
- Minimize the surface area for each level
- Here: Local Search method is used
- Idea: Random variation of a point, select if improvement
- Add penalty in case of constraint violation

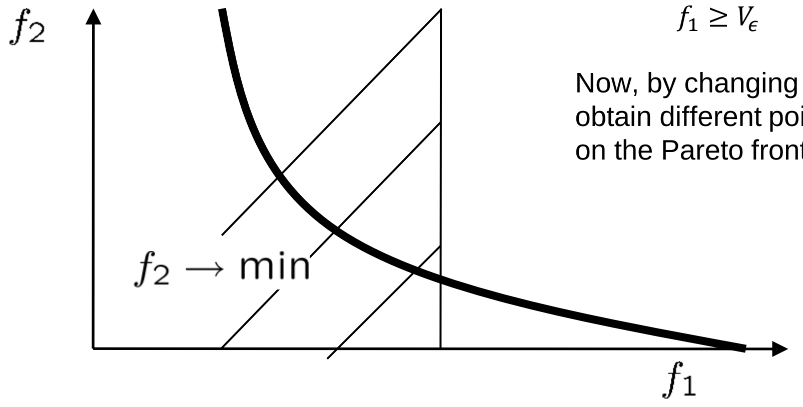
- Or: Use a scalarization method; see example in lecture on Single point method (linear weighting scalarization)





$f_1 \to max$ (e.g. volume)

$f_1 \geq V_\epsilon$



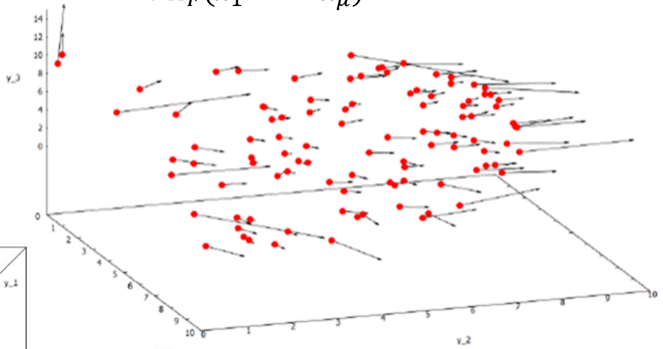
Now, by changing V_ϵ obtain different points on the Pareto front

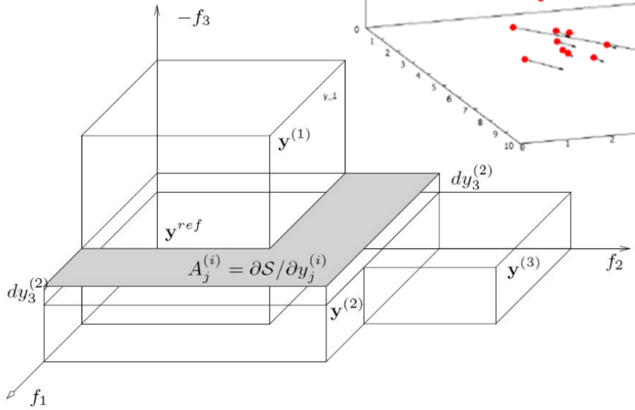
Universiteit Leiden

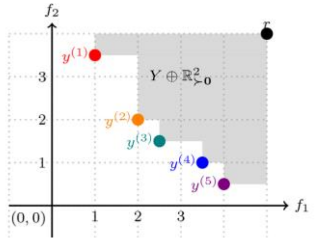
Set-Scalarization based Gradient & Newton Methods

$$\overbrace{\mathbb{R}^{\mu\cdot d} \underbrace{\xrightarrow{\mathsf{F}}}_{\text{decision space to objective space}} \mathbb{R}^{\mu\cdot m} \underbrace{\xrightarrow{\mathcal{H}}}_{\text{objective space to single value}} \mathbb{R}.}^{\mathcal{H}_\mathsf{F}}$$

$\nabla H_F(\vec{x}_1 \circ \cdots \circ \vec{x}_\mu)$







- Pareto front gradients proposed by Emmerich, Beume, Deutz 2007 for 2-D, Emmerich & Deutz 2014 for 3-D
- Generalization to N-D and efficient computation (**compute visible facets**) in $\Theta(nd + n \log n)$ optimal time by Emmerich & Deutz 2013
- Gradient Methods: Linear convergence speed (Emmerich, Beume 2007), (Wang, Emmerich, Bäck 2017)
- Multicriteria Hypervolume Newton's Method: Quadratic Convergence speed (Sosa, Wang, Schütze, Deutz, Emmerich, IEEE Transactions Cybernetics, 2019)

Emmerich, Michael, and André Deutz. "Time complexity and zeros of the hypervolume indicator gradient field." EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation III. Springer International Publishing, 2014. 169-193.
Sosa-Hernandez A., Wang H., Schütze O. and Emmerich M.: Set-based Newton's Method for Hypervolume Maximization, IEEE Transactions on Cybernetics, 2019

Universiteit Leiden

Summary

- In quadratic optimization and local optimization on convex differentiable problems Newton or Quasi-Newton methods yield precise solutions in small time (effort increases quadratically with dimension)
- For non-quadratic problems with as single optimum robust hillclimbing method like Hooke Jeeves and Simplex Method can be used
- Many other local search methods available (see Numerical Recipes etc.)
- Local search can be used also in discrete spaces and guarantees the convergence to a local optimum
- Three strategies for using local search in multicriteria optimization are Single-point methods, Memetic Algorithms, and Set-scalarization methods
- Set-Scalarization has been used in combination with Newton's method.

Universiteit Leiden