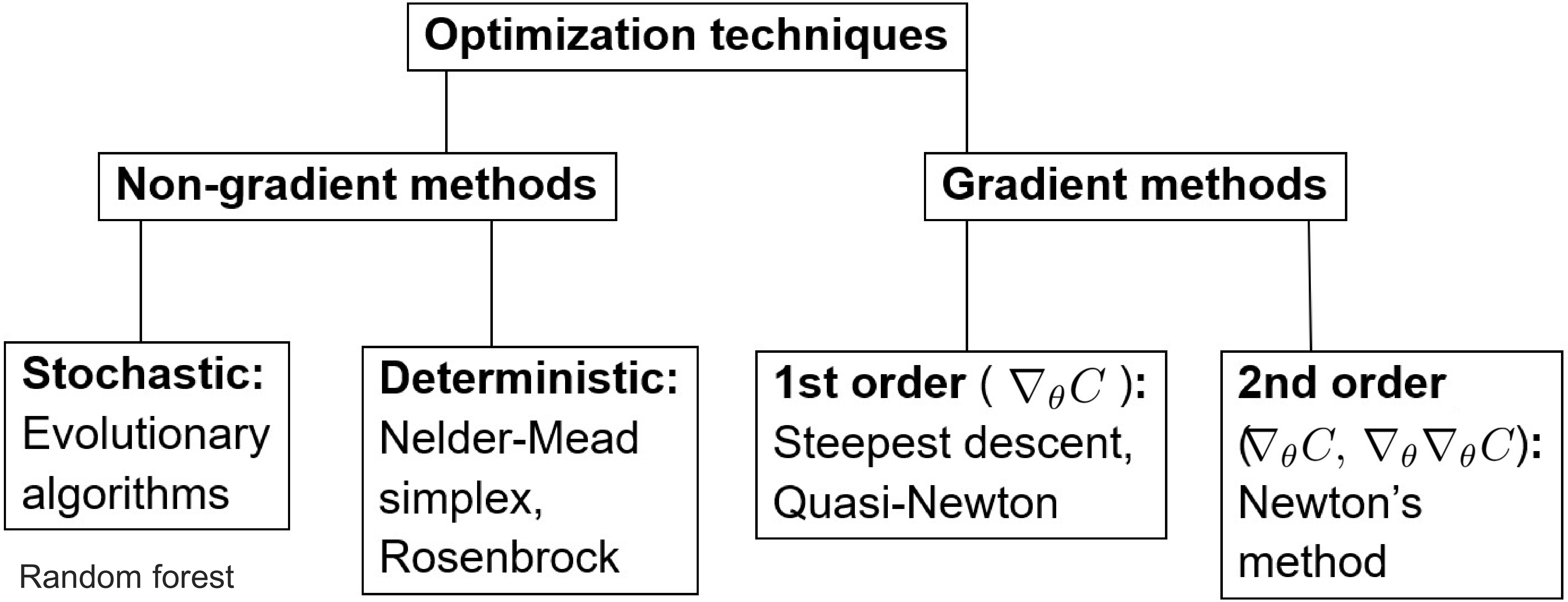
Planning of course:

- Dimensional reduction: singular value decomposition, principal component analysis, model reduction
- Regression, optimization, model selection
- Neural networks
- Physics informed neural networks, model discovery
- Projects


Optimization techniques
Non-gradient methods
Gradient methods
Stochastic:
Evolutionary algorithms
Deterministic:
Nelder-Mead simplex, Rosenbrock
1st order ($\nabla_\theta C$):
Steepest descent, Quasi-Newton
2nd order ($\nabla_\theta C$, $\nabla_\theta \nabla_\theta C$):
Newton's method

Random forest
…

Decision tree
…

Notation:

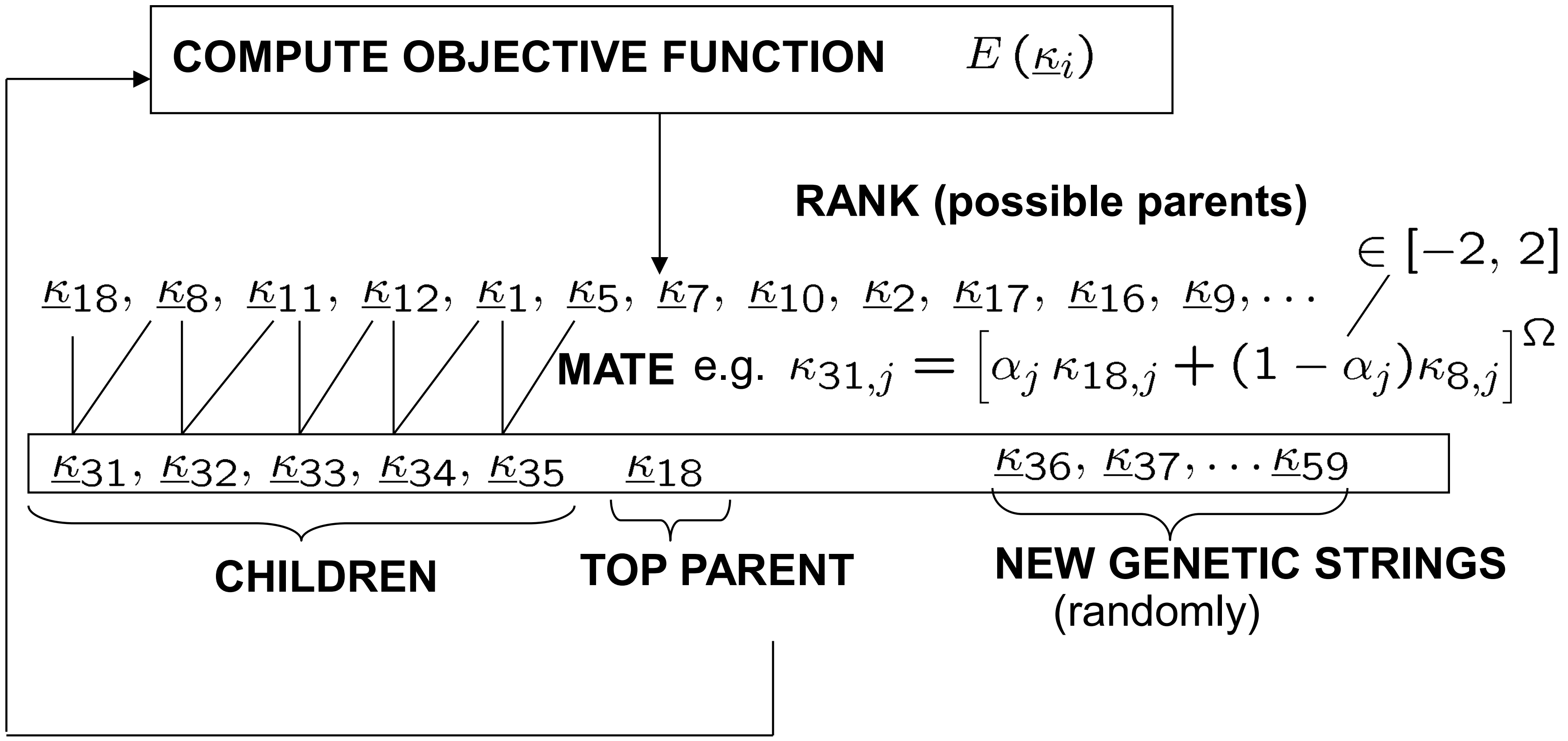
$\theta \rightarrow \kappa$

$C \rightarrow E$

Evolutionary algorithms, example

INITIATION: generate genetic strings (parameters sets) randomly

$$\underline{\kappa}_i \in \Omega \,, \quad i = 1, 2, \ldots, 30$$



COMPUTE OBJECTIVE FUNCTION $E\left(\underline{\kappa}_i\right)$
RANK (possible parents)
$\underline{\kappa}_{18},\ \underline{\kappa}_{8},\ \underline{\kappa}_{11},\ \underline{\kappa}_{12},\ \underline{\kappa}_{1},\ \underline{\kappa}_{5},\ \underline{\kappa}_{7},\ \underline{\kappa}_{10},\ \underline{\kappa}_{2},\ \underline{\kappa}_{17},\ \underline{\kappa}_{16},\ \underline{\kappa}_{9}, \ldots$
$\in [-2,\ 2]$
MATE e.g. $\kappa_{31,j} = \left[\alpha_j\, \kappa_{18,j} + (1-\alpha_j)\kappa_{8,j}\right]^{\Omega}$
$\underline{\kappa}_{31},\ \underline{\kappa}_{32},\ \underline{\kappa}_{33},\ \underline{\kappa}_{34},\ \underline{\kappa}_{35}$
$\underline{\kappa}_{18}$
$\underline{\kappa}_{36},\ \underline{\kappa}_{37}, \ldots \underline{\kappa}_{59}$
CHILDREN
TOP PARENT
NEW GENETIC STRINGS
(randomly)

Notation:

$\theta \to \kappa$

$C \to E$

Nelder – Mead simplex algorithm (1965)

- **Update idea:** *n*+1 points (desribing a simplex) in the parameter space:

$$\Omega = \left\{ \underline{\kappa} \in \mathcal{R}^n | \kappa_{\mathrm{min},i} \leq \kappa_i \leq \kappa_{\mathrm{max},i} \right\}$$

  is compared and the “worst” point is replaced.
- **Iteration steps:**
  1. Compute the objective function E for $\underline{\kappa}_1, \dots, \underline{\kappa}_{n+1}$
  2. Denote:

$$\underline{\kappa}_h = \mathrm{arg}\left(\max_{i=1,..,n+1} E(\underline{\kappa}_i) \right) \;,\; \underline{\kappa}_s = \mathrm{arg}\left(\max_{i=1,..,n+1\;,\;i\neq h} E(\underline{\kappa}_i) \right)$$

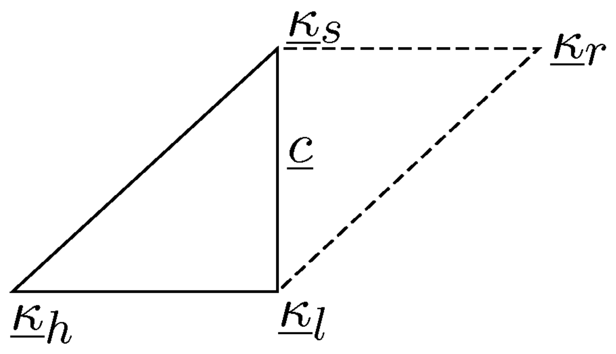
$$\underline{\kappa}_l = \mathrm{arg}\left(\min_{i=1,..,n+1} E(\underline{\kappa}_i) \right)$$

3. Compute the centroid (of all points except $\underline{\kappa}_h$):

$$\underline{c} = \frac{1}{n} \sum_{i=1\ ,\ i \neq h}^{n+1} \underline{\kappa}_i$$

4. Reflection of worst point through the centroid (α=1):

$$\underline{\kappa}_r = [\underline{c} + \alpha(\underline{c} - \underline{\kappa}_h)]^{\Omega} \quad , \quad \alpha > 0$$



5. If the reflection is succesful (better than the 2nd worst point), i.e.

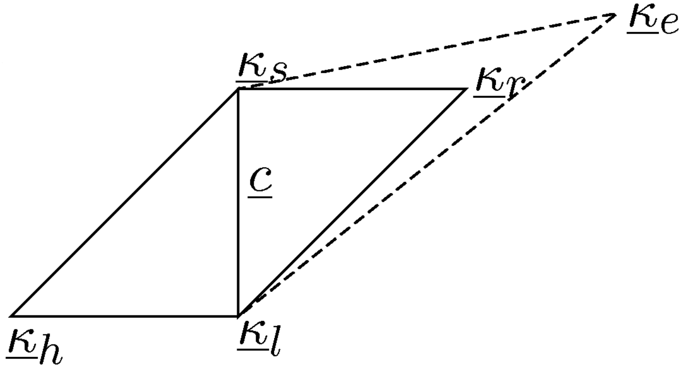
$$E(\underline{\kappa}_r) < E(\underline{\kappa}_s)$$

a) if $E(\underline{\kappa}_r) \geq E(\underline{\kappa}_l)$ then $\underline{\kappa}_r$ is accepted as the new point in the simplex $\boxed{\underline{\kappa}_h = \underline{\kappa}_r}$ goto 2

b) elseif ($\underline{\kappa}_r$ is even more sucessful) the simplex is expanded (γ=2)

$$\underline{\kappa}_e = [\underline{c} + \gamma(\underline{\kappa}_r - \underline{c})]^\Omega$$

$$\gamma > 1$$



If $E(\underline{\kappa}_e) < E(\underline{\kappa}_l)$ then $\boxed{\underline{\kappa}_h = \underline{\kappa}_e}$ goto 2

else $E(\underline{\kappa}_e) \geq E(\underline{\kappa}_l)$ then $\boxed{\underline{\kappa}_h = \underline{\kappa}_r}$ goto 2

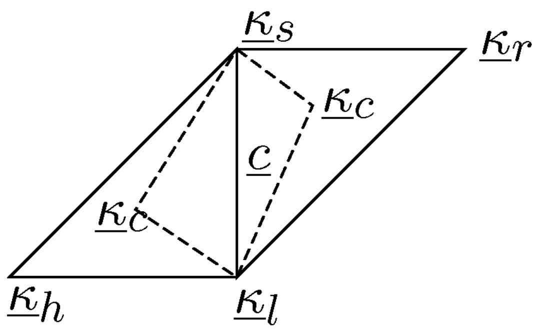
6. If the reflection *not* is succesful (worser than the 2nd worst point), i.e.

$$E(\underline{\kappa}_r) \geq E(\underline{\kappa}_s)$$

then perform contraction (β=0.5):

$$\underline{\kappa}_c = \begin{cases} \underline{c} + \beta(\underline{\kappa}_h - \underline{c}) \ , & \text{if } E(\underline{\kappa}_r) > E(\underline{\kappa}_h) \\ \underline{c} + \beta(\underline{\kappa}_r - \underline{c}) \ , & \text{else} \end{cases}$$

better than worst point

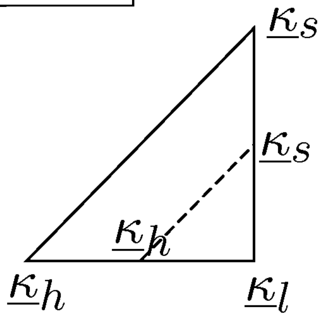


a) If $E(\underline{\kappa}_c) < \min(E(\underline{\kappa}_r) \, , \, E(\underline{\kappa}_h))$ then $\boxed{\underline{\kappa}_h = \underline{\kappa}_c}$ goto 2

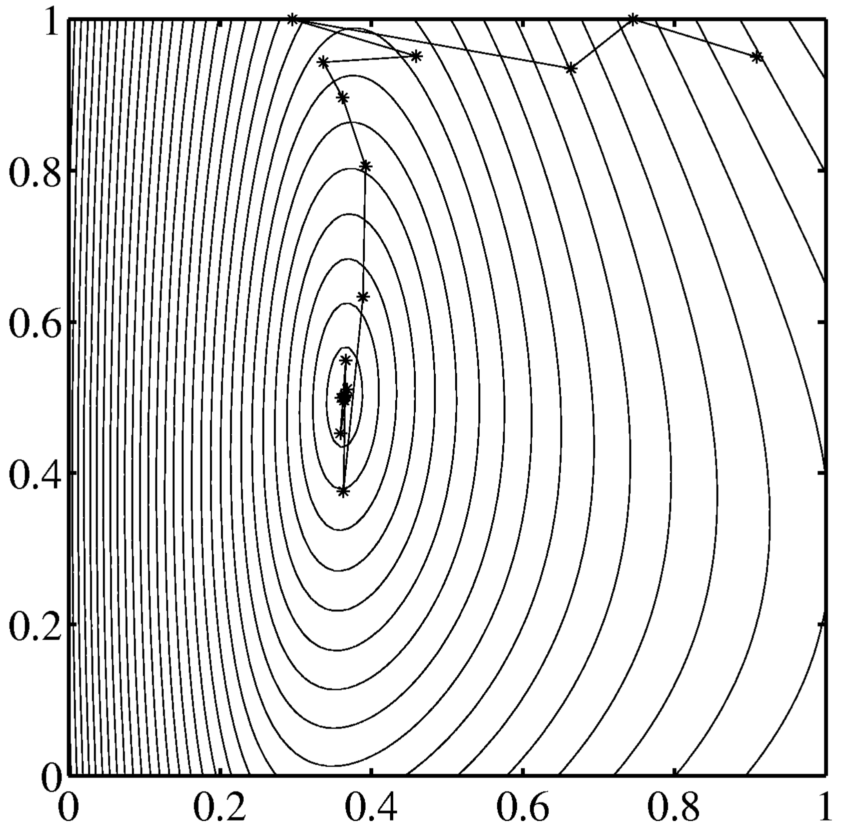
b) else the simplex is shrinked (δ=0.5)

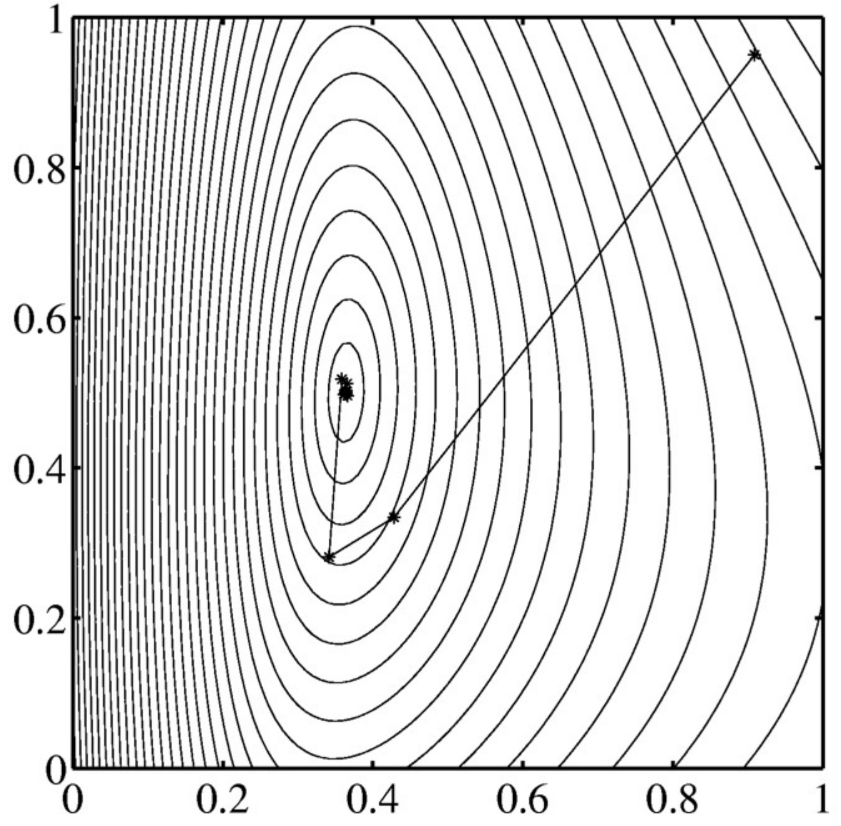
$$\boxed{\underline{\kappa}_i = \underline{\kappa}_l + \delta(\underline{\kappa}_i - \underline{\kappa}_l) \ , \ i = 1, ..., n+1}$$

goto 1

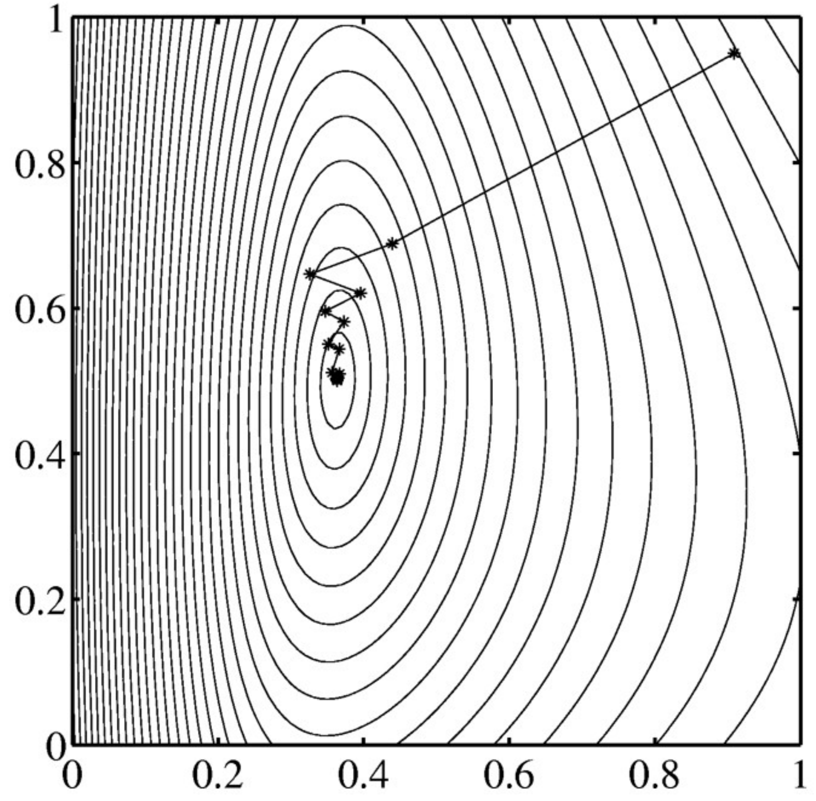


Nelder – Mead Simplex algorithm (1965)





Modified Newton (Hessian approximated by BFGS approximation)



Steepest/gradient descent

$$\underline{\theta}^{k+1} = \underline{\theta}^{k} - \alpha \, \nabla_{\theta} C(\underline{\theta}^{k})$$

$0 \leq \alpha \leq 1$ line search parameter/learning rate

For a ”small” neural network:

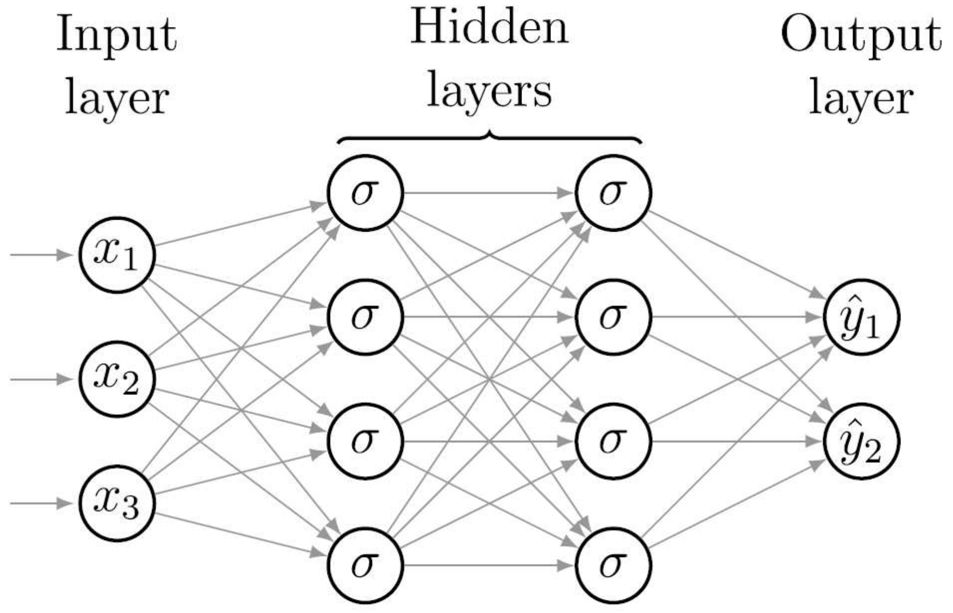


Fig. 3.1 A fully connected feed-forward neural network

Number of parameters: 4*(3+1)+4*(4+1)+2*(4+1)=16+20+10=46

Efficient gradient based methods are used.

https://arxiv.org/pdf/1609.04747

Gradient descent: $\underline{\theta}^{k+1} = \underline{\theta}^{k} - \alpha \, \nabla_\theta C(\underline{\theta}^{k})$

If you use all data points: full-batch gradient descent

$$C(\underline{\theta}) = \frac{1}{m} \sum_{i=1}^{m} C_i = \frac{1}{m} \sum_{i=1}^{m} \left(y_i - \hat{y}_i(x_i; \underline{\theta})\right)^2$$

Stochastic gradient

$$\underline{\theta}^{k+1} = \underline{\theta}^{k} - \alpha \, \frac{\partial C_i(\underline{\theta}^{k})}{\partial \underline{\theta}^{k}}$$

$$\underline{\theta}^{k+2} = \underline{\theta}^{k+1} - \alpha \, \frac{\partial C_{i+1}(\underline{\theta}^{k+1})}{\partial \underline{\theta}^{k+1}}$$

… gives better convergence than steepest gradient.

Mini-batch gradient descent, take a group of C_i and update $\underline{\theta}$:

$$\underline{\theta}^{k+1} = \underline{\theta}^{k} - \alpha \sum_{i=1}^{n_{batch}} \frac{\partial C_i(\underline{\theta}^k)}{\partial \underline{\theta}^k}$$

https://arxiv.org/pdf/1609.04747

Gradient descent: $$\underline{\theta}^{k+1} = \underline{\theta}^{k} - \alpha\,\nabla_\theta C(\underline{\theta}^{k})$$

Adagrad, individual learning rate (for each parameter *i*)

$$\theta_i^{k+1} = \theta_i^k - \alpha_i\,\frac{\partial C(\underline{\theta}_k)}{\partial \theta_i^k}$$

with $$\alpha_i = \frac{\alpha}{\sqrt{G_i + 10^{-8}}}$$

$$G_i = \left(\frac{\partial C}{\partial \theta_i^0}\right)^2 + \left(\frac{\partial C}{\partial \theta_i^1}\right)^2 + \ldots \left(\frac{\partial C}{\partial \theta_i^k}\right)^2$$

+ lower learning rate if gradient is steep

- learning rate decreases with no of iterations

Adam, uses ”momentum”

$$m_i^{k+1} = \beta_1\, m_i^k + (1-\beta_1)\,\frac{\partial C}{\partial \theta_i^k}, \quad \hat{m}_i^{k+1} = \frac{m_i^{k+1}}{1-\beta_1}$$

$$\theta_i^{k+1} = \theta_i^k - \frac{\alpha}{\sqrt{\hat{v}_i^{k+1}} + 10^{-8}}\,\hat{m}_i^k$$

$$v_i^{k+1} = \beta_2\, v_i^k + (1-\beta_2)\,\left(\frac{\partial C}{\partial \theta_i^k}\right)^2, \quad \hat{v}_i^{k+1} = \frac{v_i^{k+1}}{1-\beta_2}$$

Python packages (used in this course):

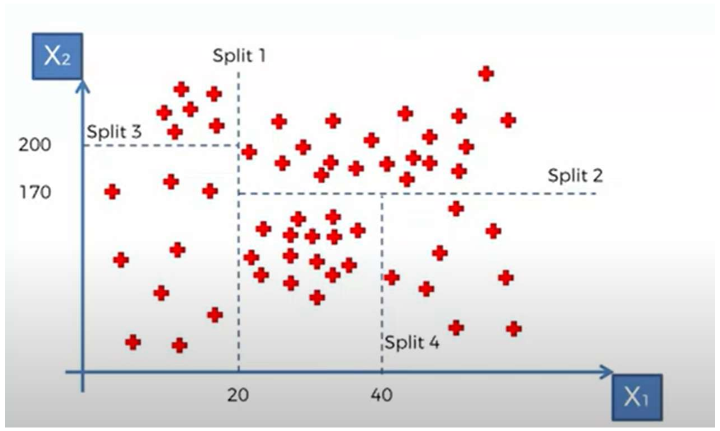
Optimization algorithms in SciPy:

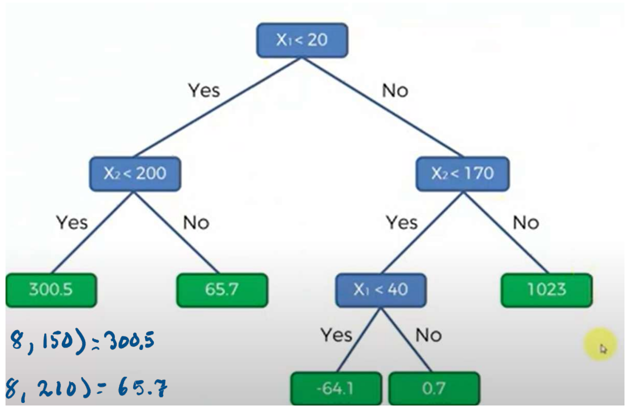
https://docs.scipy.org/doc/scipy/tutorial/optimize.html

Optimization algorithms in PyTorch:

https://pytorch.org/docs/stable/optim.html

Decision trees





Ex. $f(8,150) = 300.5$

$f(8,210) = 65.7$

Decision Tree Regression Clearly Explained

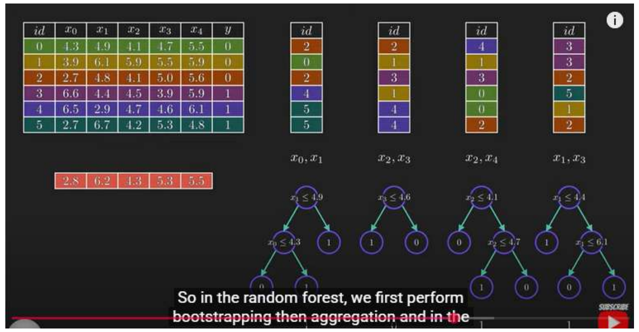
Splits are introduced to get most information

The green values are the average in each area

Random forest algorithms: use a random forest of decision trees

Random Forest Algorithm Clearly Explained!

scikit-learn package for ML



id	x_0	x_1	x_2	x_3	x_4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

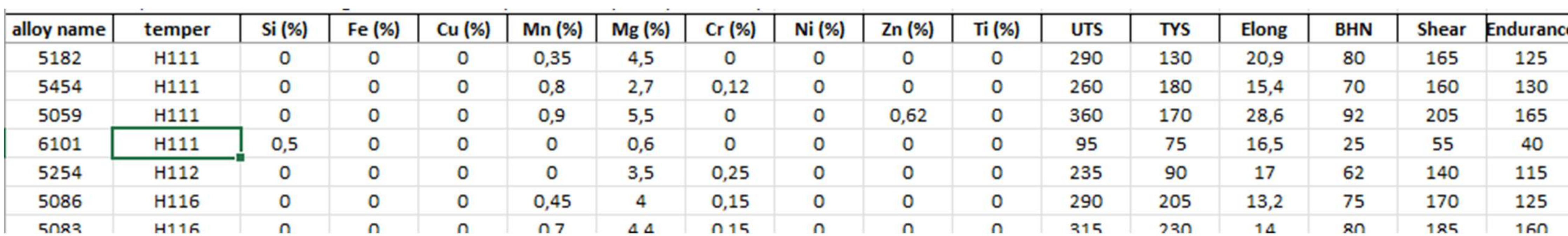
id	id	id	id
2	2	4	3
0	1	1	3
2	3	3	2
4	1	0	5
5	4	0	1
5	4	2	2

2.8	6.2	4.3	5.3	5.5

So in the random forest, we first perform bootstrapping then aggregation and in the

Example: random forest

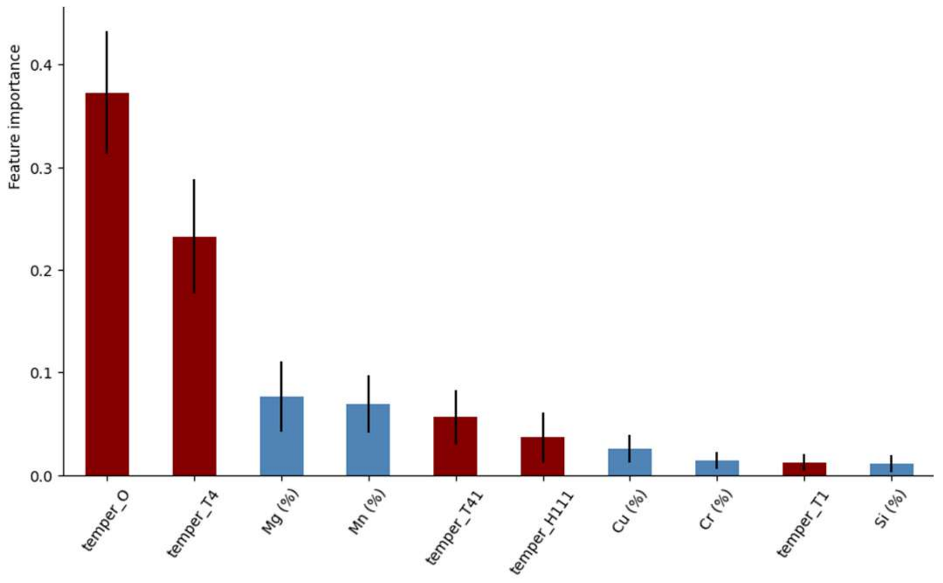
Alloy data:

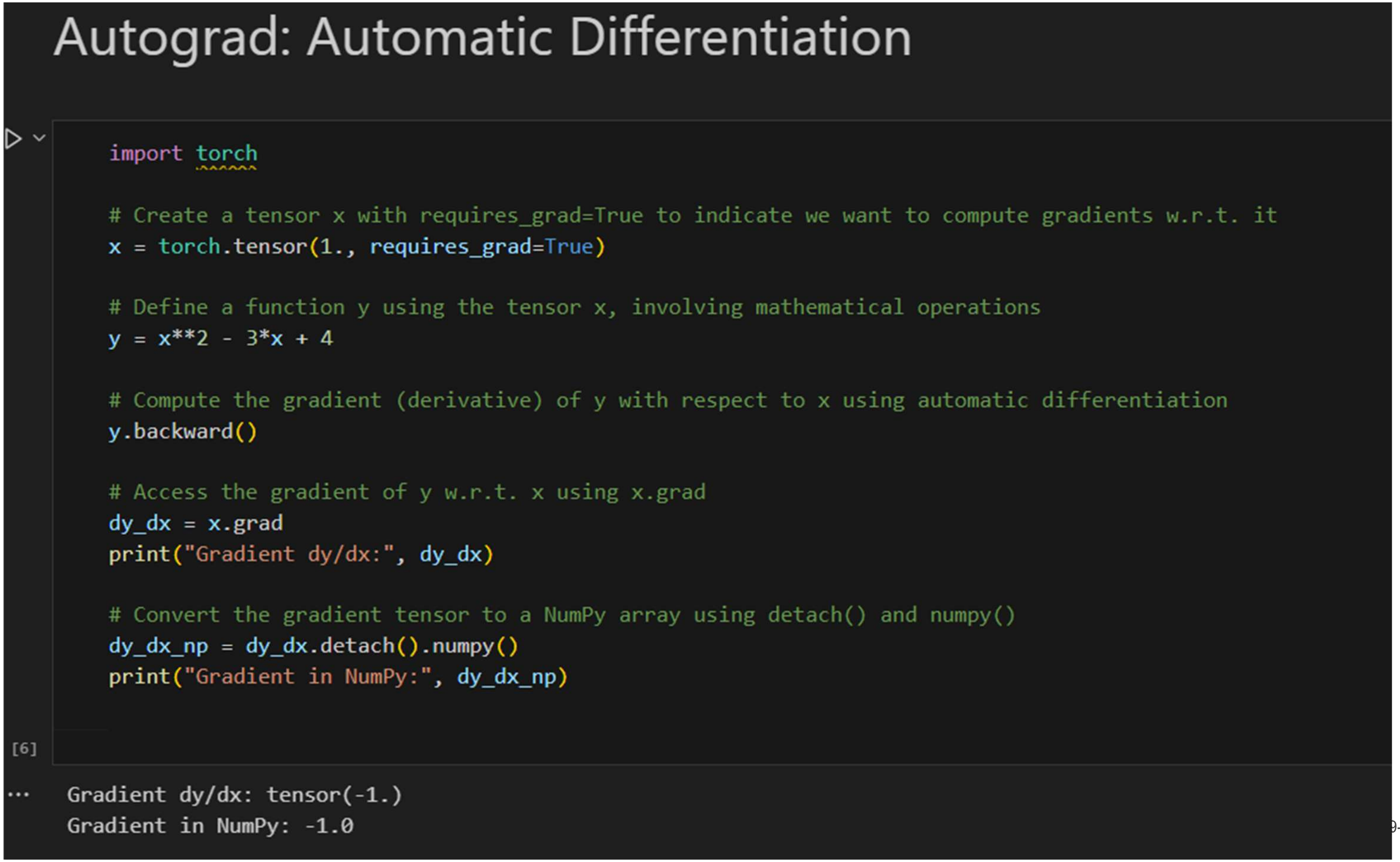


alloy name	temper	Si (%)	Fe (%)	Cu (%)	Mn (%)	Mg (%)	Cr (%)	Ni (%)	Zn (%)	Ti (%)	UTS	TYS	Elong	BHN	Shear	Enduranc
5182	H111	0	0	0	0,35	4,5	0	0	0	0	290	130	20,9	80	165	125
5454	H111	0	0	0	0,8	2,7	0,12	0	0	0	260	180	15,4	70	160	130
5059	H111	0	0	0	0,9	5,5	0	0	0,62	0	360	170	28,6	92	205	165
6101	H111	0,5	0	0	0	0,6	0	0	0	0	95	75	16,5	25	55	40
5254	H112	0	0	0	0	3,5	0,25	0	0	0	235	90	17	62	140	115
5086	H116	0	0	0	0,45	4	0,15	0	0	0	290	205	13,2	75	170	125
5083	H116	0	0	0	0,7	4,4	0,15	0	0	0	315	230	14	80	185	160

We can get:

Importance of heat treatment, chemical composition to different features





Autograd: Automatic Differentiation

```
import torch

# Create a tensor x with requires_grad=True to indicate we want to compute gradients w.r.t. it
x = torch.tensor(1., requires_grad=True)

# Define a function y using the tensor x, involving mathematical operations
y = x**2 - 3*x + 4

# Compute the gradient (derivative) of y with respect to x using automatic differentiation
y.backward()

# Access the gradient of y w.r.t. x using x.grad
dy_dx = x.grad
print("Gradient dy/dx:", dy_dx)

# Convert the gradient tensor to a NumPy array using detach() and numpy()
dy_dx_np = dy_dx.detach().numpy()
print("Gradient in NumPy:", dy_dx_np)
```

[6]

```
Gradient dy/dx: tensor(-1.)
Gradient in NumPy: -1.0
```