Numerical Optimization

Samuele Carli

AS-AI school
Entersys s.r.l.
carlisamuele@csspace.net

Outline

Problems

- AI
- Fitting of simulation parameters
- Linear and non-linear systems
- Simulation

$\longrightarrow$ any domain where there are measurements and uncertainty!

Errors, Machine Numbers, Iterative methods

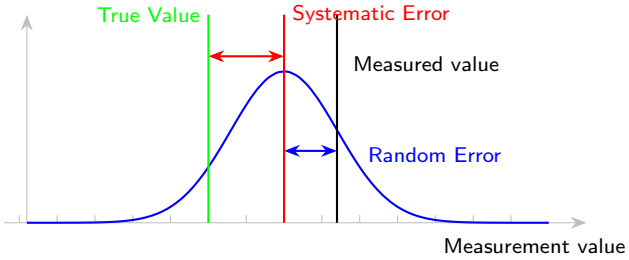
Error sources

- measurement
- method / approximation
- representation

Accuracy vs Precision

	Accurate	Not Accurate (Inaccurate)
Precise		
Not Precise (Imprecise)		

Measurement Error



Approximation / method errors

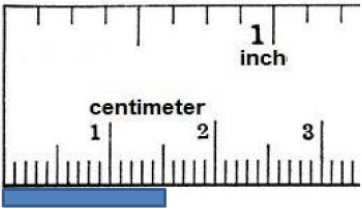
Taylor's approximation:

$$f(a+h) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} h^n \approx \sum_{n=0}^{k} \frac{f^{(n)}(a)}{n!} h^n + O(h^{k+1})$$

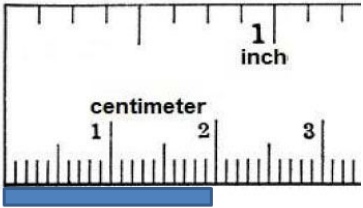
Newton's method:

$$x \text{ such that } f(x) = 0 \Longrightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Representation Error



15mm



19.5mm

Absolute - Relative

Absolute:

$$\Delta x \equiv x - \tilde{x}$$

Relative:

$$\epsilon_x \equiv \frac{\Delta x}{x} = \frac{x - \tilde{x}}{x}$$

Think of an error of 1 on $x = 10^{-9}$ or $x = 10^{9}$

Error analysis

$$y = f(x)$$

$$|\epsilon_y| \approx \kappa |\epsilon_x|$$

- well conditioned: $\kappa \approx 1$
- ill conditioned: $\kappa \gg 1$

Finite Arithmetic

Positional representation

$$n_k \dots n_0 . d_1 \dots d_l$$

- base 10: $n, d \in \{0, .., 9\} \rightarrow \sum_{i=0}^{k} n_i 10^i + \sum_{j=1}^{l} d_j 10^{-j}$
- base 2: $n, d \in \{0, 1\} \rightarrow \sum_{i=0}^{k} n_i 2^i + \sum_{j=1}^{l} d_j 2^{-j}$

Integers

- base 10: $n, d \in \{0, .., 9\} \rightarrow \sum_{i=0}^{k} n_i 10^i$
- base 2: $n, d \in \{0, 1\} \rightarrow \sum_{i=0}^{k} n_i 2^i$

16 bit integer:

- Unsigned: 0 to $\sum_{i=0}^{1} 62^i = 2^{17} - 1 = 131071$
- Signed: $-\sum_{i=0}^{15} 2^i = -2^{16} - 1 = -65535$ to $\sum_{i=0}^{15} 2^i = -2^{16} - 1 = 65535$

Signed integers

Zero is reprented twice! Using the 2-complement:

$$n = \begin{cases} \sum_{i=1}^{N} n_i 2^{N-i}, & \text{if } n_N = 0 \\ \sum_{i=1}^{N} n_i 2^{N-i} - b^N, & \text{if } n_N = 1 \end{cases}$$

we can store one number more so that the inteval becomes $[-2^N, 2^N - 1]$

Floating point representation

Much like scientific notation $\pm nEk = n10^k$

- sign s (1 bit)
- exponent $e = e_1, ..., e_m$ (8 bit in float32)
- fractional part $f = f_1, ..., f_s$ (23 bit in float32)
- exponent bias ν

$$x = se_1 ... e_m f_1 ... f_s$$

Floating point representation (2)

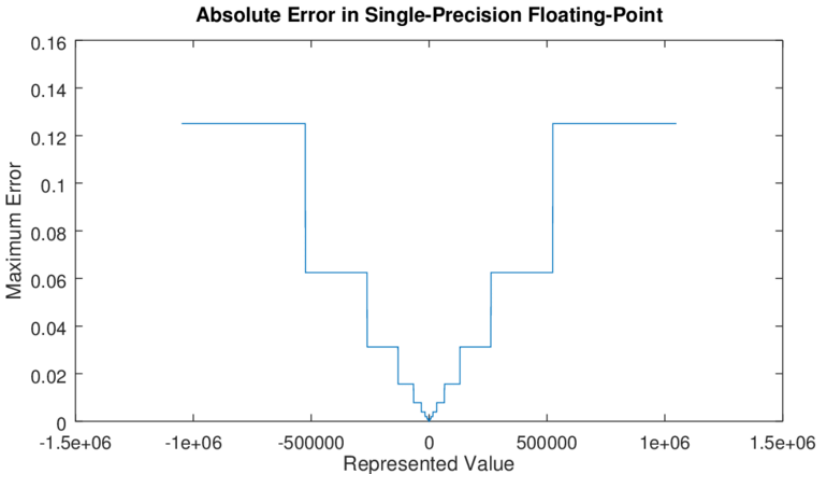
$$x = se_1 \ldots e_m f_1 \ldots f_s = (-1)^s \left(\sum_{i=1}^{s} f_i 2^{s-i} \right) 2^{r-\nu}, \quad r = \sum_{j=1}^{m} e_j 2^{m-j}$$

- finite set 2^n elements
- smallest representable number $N_m \approx 2^{-m+1}$
- biggest representable number $N_M \approx 2^{m+s}$
- $[N_m, N_M]$ not sampled uniformly: $|[0,1]| \approx |[1, N_M]|$
- real implementation has ± 0, plus combinations reserved for special $\pm\infty, NaN$ etc.
- normalization gives one extra bit

FP limits

- **Overflow** $|x| > N_M; \pm \inf$
- **Underflow** $|x| < N_m$

FP error


Absolute Error in Single-Precision Floating-Point
Maximum Error
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-1.5e+06
-1e+06
-500000
0
500000
1e+06
1.5e+06
Represented Value

Implicit base conversions!

Whatch out as this always happens! The problem:

$$
\begin{aligned}
1|_{10} &= 2^0 = 1|_2 \\
10|_{10} &= 10^1 = 2^3 + 2^1 = 1010|_2 \\
0.1|_{10} &= 2^{-4} + 2^{-5} + 2^{-8} + ... \\
&= 0.0001|_2 \ (=0.0625|_{10}) + 0.0000100110...|_2
\end{aligned}
$$

FP tests

Float16:

- sign,
- 5 bit exponent $\approx 2^6 - 1 = 31$, $\nu = 16 \rightarrow$ max exponent $2^6 - 1 - 16 = 15$
- 10 bit fraction $1.1111111111|_2 \approx 1.999|_{10}$
- expected max $\approx 2 \cdot 2^{15} = 65536$
- expected normalized min $\approx 2^{-16} \approx 1.52\ 10^{-5}$
- expected denormalized min $\approx 2^{-10}2^{-15} \approx 5.96\ 10^{-8}$

Implementation

```
import numpy as np
import pandas as pd
from numpy import float16 as f, float32 as f32

%precision 100
np.set_printoptions(precision=100)
pd.options.display.float_format = '{:.100f}'.format
```

```
# Expected max: 2^16 = 65536
print(2**15*2)
print(f(65536))
```

```
65536
inf
```

```
/tmp/ipykernel_2478194/3921099518.py:3: RuntimeWarning: overflow encountered in cast
  print(f(65536))
```

```
1  x = 1.999*2**15
2  print(x)
3  print(f(x))
```

```
1  65503.232
2  65500.0
```

```
#(1).11111 11111 |2 = (2^0) + 2^(-x) x=1,...,10
x = sum([2**-x for x in range(0,11)])
print(x)
print(x*2**15)
```

```
1.9990234375
65504.0
```

```python
# 65500?
print(f(65504))
print(f(65505))

print(f(65519))
print(f(65520))
```

```
65500.0
65500.0
65500.0
inf
```

```
/tmp/ipykernel_2478194/3696056160.py:6: RuntimeWarning: overflow encountered in cast
  print(f(65520))
```

```
# There is some approximation done by the display function, different for f32!
print(f32(f(65504)))
print(f32(f(65505)))

print(f32(f(65519)))
print(f32(f(65520)))
```

```
65504.0
65504.0
65504.0
inf
```

```
/tmp/ipykernel_2478194/3422968668.py:6: RuntimeWarning: overflow encountered in cast
  print(f32(f(65520)))
```

```
# when the exponent is maximum, the resolution is 32
print(2**15 * 2**-10)
print(2**16 - 32)
```

```
32.0
65504
```

```
#1.11111 11110  2**-10 * 2**15 = 2**5 = 32 --> approx @ [0-15][16-32]
x = sum([2**-x for x in range(0,10)])
print(x)
print(x*2**15)
```

```
1.998046875
65472.0
```

```
#1.11111 11110  2**-10 * 2**15 = 2**5 = 32 --> approx @ [0-15][16-32]
print(f32(f(65472)))
print(f32(f(65473)))
print()
#472+16 = 488, 472+32=504
print(f32(f(65488)))
print(f32(f(65489)))
print(f32(f(65490)))
```

```
65472.0
65472.0

65472.0
65504.0
65504.0
```

```
#1.11111 11111  + [0-16] -> 65504 + 16 = inf
print(f32(f(65504)))
#504+16 = 520
print(f32(f(65519)))
print(f32(f(65520)))
```

```
65504.0
65504.0
inf
```

```
/tmp/ipykernel_2478194/1849818127.py:5: RuntimeWarning: overflow encountered in cast
  print(f32(f(65520)))
```

```python
b = [0]
s = lambda b: sum([b[j]*2**-j for j in range(len(b))])

for i in range(1,200):
    if (s(b)+2**-i) < 0.1:
        b.append(1)
    else:
        b.append(0)

print(b)
```

```
[0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
↪  0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
1  s(b)
```

```
1  0.1000000000000000055511151231257827021181583404541015625000000000000000000000000000000000000000000000000000000000000000
```

```
1  0.1
```

```
1  0.1000000000000000055511151231257827021181583404541015625000000000000000000000000000000000000000000000000000
```

```
1  0.1+0.1
```

```
1  0.2000000000000000011102230246251565404236316680908203125000000000000000000000000000000000000000000000000000
```

```
# Attention for stopping conditions!
print(f(0.1)+f(0.1) == 0.2)
print(f(0.1)+f(0.1) == f(0.2))
print(f32(0.1)+f32(0.1) == 0.2)
print(0.1+0.1 == 0.2)
```

```
False
True
False
True
```

```
#1.00000 00000 * 2^-16 = 1.52 10^-5
print(f32(2**-16))
print(f(2**-16)) #significant digits
print(f32(f(2**-16)))
print(1/65536)
```

```
1.5258789e-05
1.526e-05
1.5258789e-05
1.52587890625e-05
```

```
#0. 00000 001 = 2^-16 = 2^-24 = 5.9 10^-8
print(f32(f(2**-24)))
print(f32(f(2**-25)))
print(f32(2**-25))
```

```
5.9604645e-08
0.0
2.9802322e-08
```

Pay attention to conversions!

- Integer to FP: always possible within FP precision
- FP to Integer: may not be representable, or change sign unexpectedly!
- Some conversions happen without you even knowing! Beware of print functions!

Finding Roots

Bisection method

$$y = f(x): \quad \bar{x} \in [a, b] \text{ such that } f(\bar{x}) = 0?$$

- f continuous on $[a, b]$
- $f(a)f(b) < 0 \Rightarrow \exists \bar{x} | f(\bar{x}) = 0$

Bisection method (2)

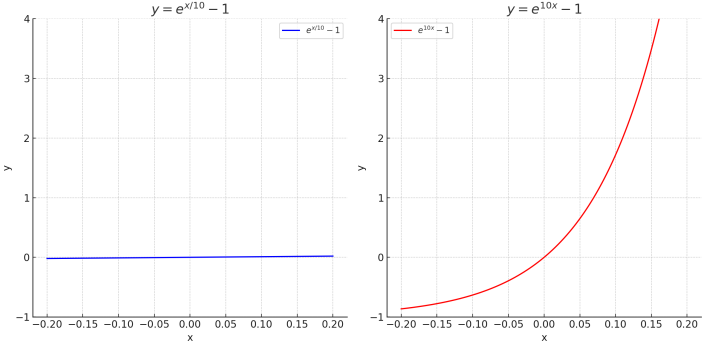
1. let $k = 0$; $[a_0, b_0] = [a, b]$;
2. iteration step: $c_{k+1} = a_k + \frac{1}{2}(b_k - a_k) \Longrightarrow [a_k, c_{k+1}], [c_{k+1}, b_k]$
3. iteration check: stop if $f(c_{k+1}) = 0$; return c_{k+1}
4. $[a_{k+1}, b_{k+1}] = [a_k, c_{k+1}]$ if $f(a_k)f(c_{k+1}) < 0$ else $[c_{k+1}, b_k]$
5. $k = k + 1$, restart from 2

Bisection method (3)

1. let $k = 0$; $[a_0, b_0] = [a, b]$; let $tolx$; $toly$
2. iteration step: $c_{k+1} = a_k + \frac{1}{2}(b_k - a_k) \Longrightarrow [a_k, c_{k+1}], [c_{k+1}, b_k]$
3. iteration check: stop if $|b_k - a_k| \leq tolx$; return c_{k+1}
4. iteration check: stop if $f(c_{k+1}) \leq toly$; return c_{k+1}
5. $[a_{k+1}, b_{k+1}] = [a_k, c_{k+1}]$ if $f(a_k)f(c_{k+1}) < 0$ else $[c_{k+1}, b_k]]$
6. $k = k + 1$, restart from 2

How to determine *toly*?

think $e^{x/10} - 1$ vs $e^{10x} - 1$



$$toly \approx f'(\bar{x}) tolx$$

$$f'(\bar{x}) \approx \frac{f(b_k) - f(a_k)}{b_k - a_k} \implies toly = \frac{f(b_k) - f(a_k)}{b_k - a_k} tolx$$

Conditioning

$f(x)$ can be seen as error on $\bar{x}$:

$$f(x) = f'(\bar{x})(x - \bar{x}) + \underbrace{f(\bar{x})}_{=0} \Longrightarrow (x - \bar{x}) \approx \frac{f(x)}{f'(\bar{x})}$$

The factor

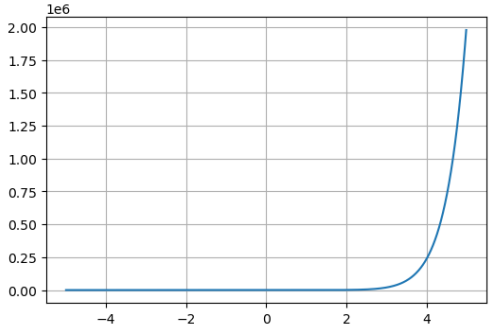
$$\kappa \approx \frac{1}{|f'(\bar{x})|}$$

can be seen as an amplification factor of the error on x over f.
When $f'(\bar{x}) \approx 0$, very small changes of f have dramatic effects on the error on x.
toly could become too small to be useful!

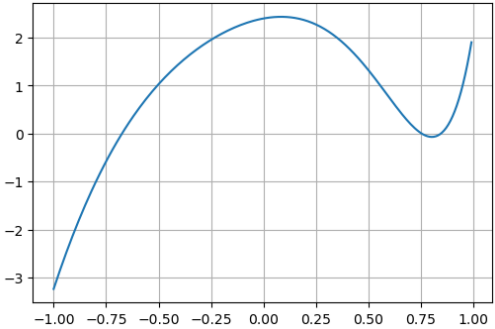
Implementation

```
1  import numpy as np
2  import math
3  from matplotlib import pyplot as plt
4  plt.rcParams['axes.grid'] = True
```

```
1  def f(x):
2      return 0.8*np.exp(x) * (6*x**5 - 3*x**4 +2*x**3- 5*x**2 - 2*x + 3)
3
4  x = np.arange(-5,5,0.01)
5  plt.plot(x, f(x));
```



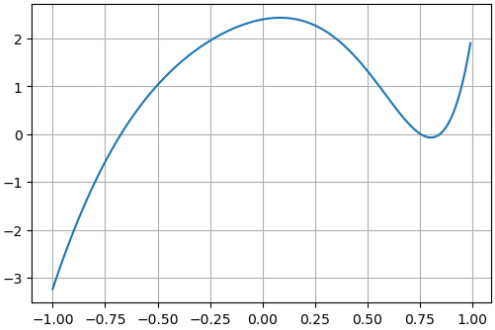
```
1  x = np.arange(-1,1,0.01)
2  plt.plot(x, f(x));
```



```python
def bisect(a, b, f, counter=0, maxiters=1000):
    c = a + (b-a)/2

    if counter>maxiters:
        return c

    if f(c) == 0:
        return c

    if f(a)*f(c) < 0:
        return bisect(a,c,f, counter=counter+1, maxiters=maxiters)
    else:
        return bisect(c,b,f, counter=counter+1, maxiters=maxiters)
```

```
a, b = -1, 1
x = bisect(a,b,f)
print(x)
print(f(x))

x = np.arange(-1,1,0.01)
plt.plot(x, f(x));
```

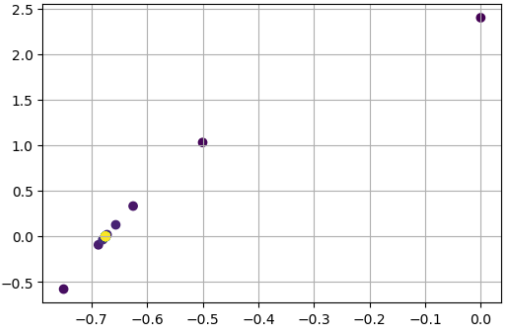
```
-0.6746113792482088
0.0
```



```python
def bisect_trace(a, b, f, counter=0, maxiters=1000, trace=None):
    if trace is None:
        trace = list()

    c = a + (b-a)/2

    trace.append([counter, a, b, c, f(c)])

    if counter>maxiters:
        return c, trace

    if f(c) == 0:
        return c, trace
    if f(a)*f(c) < 0:
        return bisect_trace(a,c,f, counter=counter+1, maxiters=maxiters, trace=trace)
    else:
        return bisect_trace(c,b,f, counter=counter+1, maxiters=maxiters, trace=trace)
```

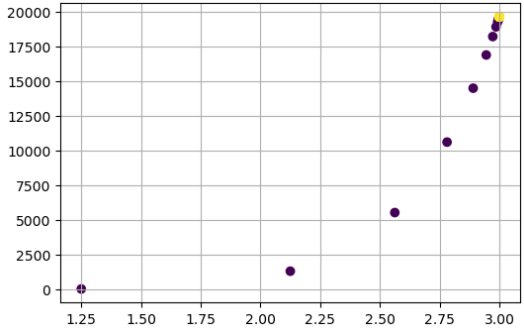
```
a, b = -1, 1
x, trace = bisect_trace(a,b,f)
print(x)
print(f(x))
print(trace[-1])

colors = np.linspace(0,1,len(trace))
plt.scatter([x[3] for x in trace], [x[4] for x in trace], c=colors, cmap='viridis');
```

```
-0.6746113792482088
0.0
[53, -0.6746113792482089, -0.6746113792482087, -0.6746113792482088, 0.0]
```

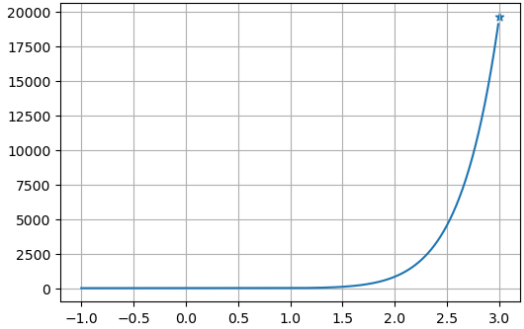


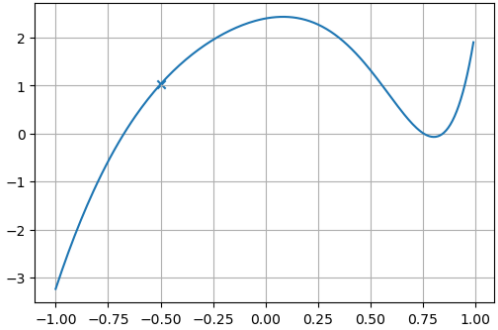
```
a, b = -0.5, 3
x, trace = bisect_trace(a,b,f)
print(x)
print(f(x))
print(trace[-1])

colors = np.linspace(0,1,len(trace))
plt.scatter([x[3] for x in trace], [x[4] for x in trace], c=colors, cmap='viridis');
```

```
3.0
19619.552466569716
[1001, 3.0, 3, 3.0, 19619.552466569716]
```

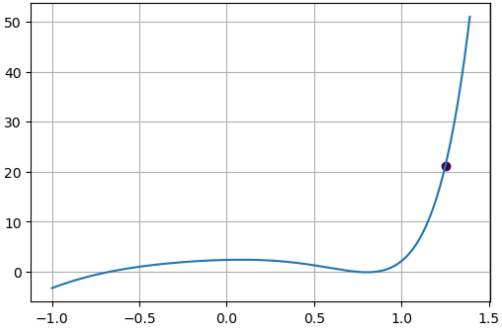


```
x = np.arange(-1,3,0.01)
plt.plot(x, f(x));
plt.scatter(b, f(b), marker='*');

plt.figure()
x = np.arange(-1,1,0.01)
plt.plot(x, f(x));
plt.scatter(a, f(a), marker='x');
```



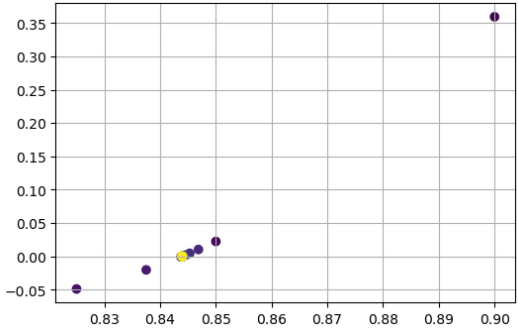

2
1
0
−1
−2
−3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

```
x = np.arange(-1,1.4,0.01)
m = 1
plt.plot(x, f(x));
colors = np.linspace(0,1,len(trace[:m]))
plt.scatter([x[3] for x in trace[:m]], [x[4] for x in trace[:m]], c=colors, cmap='viridis');
```

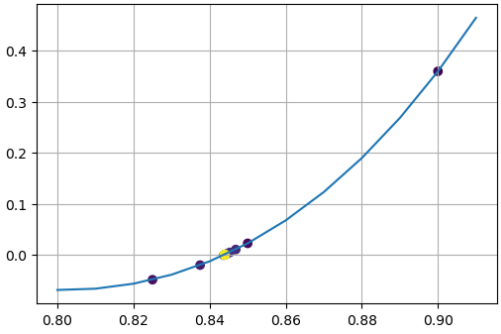


```
a, b = 0.8, 1
x, trace = bisect_trace(a,b,f)
print(x)
print(f(x))
print(trace[-1])

colors = np.linspace(0,1,len(trace))
plt.scatter([x[3] for x in trace], [x[4] for x in trace], c=colors, cmap='viridis');
```

```
0.8439655868551936
0.0
[44, 0.8439655868855188, 0.8439655868551994, 0.8439655868551936, 0.0]
```



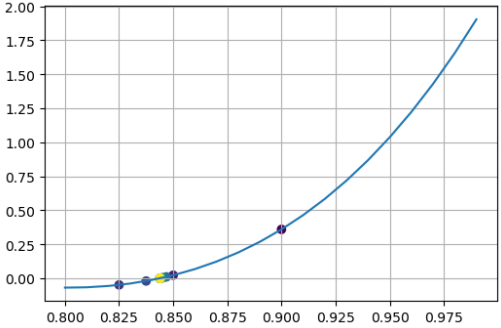
```
x = np.arange(0.8,0.92,0.01)
plt.plot(x, f(x));
colors = np.linspace(0,1,len(trace))
plt.scatter([x[3] for x in trace], [x[4] for x in trace], c=colors, cmap='viridis');
```



```python
def bisect_tol_trace(a, b, f, tolx=10**-3, toly=None, counter=0, maxiters=1000, trace=None):
    if trace is None:
        trace = list()

    if toly is None:
        toly = abs(tolx*(f(b)-f(a))/(b-a))

    c = a + (b-a)/2

    trace.append([counter, a, b, c, f(c)])

    if counter>maxiters:
        return c, trace

    if abs(c-a) <= tolx or abs(b-c) <= tolx:
        return c, trace

    if abs(f(c)) <= toly:
        return c, trace

    if f(a)*f(c) < 0:
        return bisect_tol_trace(a,c,f, tolx=tolx, toly=toly, counter=counter+1, maxiters=maxiters, trace=trace)
    else:
        return bisect_tol_trace(c,b,f, tolx=tolx, toly=toly, counter=counter+1, maxiters=maxiters, trace=trace)
```

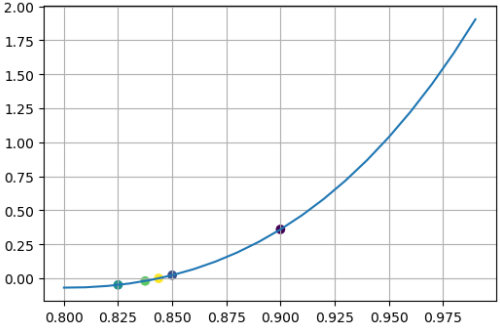
```
a, b = 0.8, 1
x, trace = bisect_tol_trace(a,b,f, tolx=10**-6)
print(x)
print(f(x))
print(trace[-1])

colors = np.linspace(0,1,len(trace))
plt.scatter([x[3] for x in trace], [x[4] for x in trace], c=colors, cmap='viridis');
x = np.arange(0.8,1,0.01)
plt.plot(x, f(x));
```

```
0.843963623046875
-6.724416240101983e-06
[14, 0.8439575195312501, 0.8439697265625, 0.843963623046875, -6.724416240101983e-06]
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.800
0.825
0.850
0.875
0.900
0.925
0.950
0.975

```
a, b = 0.8, 1
x, trace = bisect_tol_trace(a,b,f, tolx=10**-3)
print(x)
print(f(x))
print(trace[-1])

colors = np.linspace(0,1,len(trace))
plt.scatter([x[3] for x in trace], [x[4] for x in trace], c=colors, cmap='viridis');
x = np.arange(0.8,1,0.01)
plt.plot(x, f(x));
```

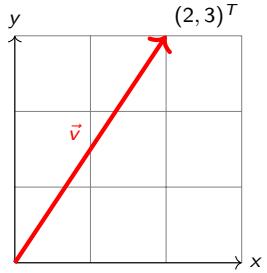
```
0.8437500000000001
-0.0007360523926994878
[4, 0.8375000000000001, 0.8500000000000001, 0.8437500000000001, -0.0007360523926994878]
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.800
0.825
0.850
0.875
0.900
0.925
0.950
0.975

Linear Algebra Recap

Scalar, Vector, Matrix

- scalar: one-dimensional number
- vector: $\mathbf{x} = (x_1, ..., x_n)^T = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$
- matrix: $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$



Norm

Length or magnitude of a vector (euclidean norm):

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \sqrt{\sum_i |x_i|^2}$$

In general, a function that holds:

- Triangle inequality: $f(x+y) \leq f(x) + f(y)$
- Linearity: $f(kx) = kf(x)$
- Positive definite: $f(x) = 0 \Rightarrow x = 0$

Other norms example:

- manhattan norm $\|\mathbf{x}\|_1 = \sum_i |x_i|$
- p-norm $\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p\right)^{1/p}$
- ∞-norm: $\max_i(|x_i|)$

Vector Operations

Scale, Sum, Hadamard Product, Scalar Product

$$c\mathbf{x} = \begin{pmatrix} cx_1 \\ \vdots \\ cx_n \end{pmatrix}, \quad \mathbf{x} + \mathbf{y} = \begin{pmatrix} x_1 + y_1 \\ \vdots \\ x_n + y_n \end{pmatrix}, \quad \mathbf{x} \circ \mathbf{y} = \begin{pmatrix} x_1 y_1 \\ \vdots \\ x_n y_n \end{pmatrix}, \quad \mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = \sum_i (x_i y_i)$$

Cross Product (only defined in $\mathbb{R}^3$):

$$\mathbf{x} \times \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \sin(\theta) \mathbf{n} = (x_2 y_3 - x_3 y_2)\mathbf{e_1} + (x_3 y_1 - x_1 y_3)\mathbf{e_2} + (x_1 y_2 - x_2 y_1)\mathbf{e_3}$$

where $\mathbf{n}$ is the unit vector perpendicular to the plane containing $\mathbf{x}, \mathbf{y}$, with direction such that $(\mathbf{x}, \mathbf{y}, \mathbf{n})$ is positively oriented (right hand rule)

Matrix Product

$$A \in M^{n \times m}, B \in M^{m \times p} \rightarrow AB \in M^{n \times p}$$

$$AB = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & & \vdots \\ b_{m1} & \cdots & b_{mp} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix} (\mathbf{b}_1, ..., \mathbf{b}_p) = \begin{pmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \cdots & \mathbf{a}_1 \cdot \mathbf{b}_p \\ \vdots & & \vdots \\ \mathbf{a}_n \cdot \mathbf{b}_1 & \cdots & \mathbf{a}_n \cdot \mathbf{b}_p \end{pmatrix}$$

Not commutative!

Watch out for bad notation! $\mathbf{a}_i \in \mathbb{R}^{1 \times m}, \mathbf{b}_j \in \mathbb{R}^{m \times 1}$

Linear (in)dependency

$$A\mathbf{x} = 0 \iff \mathbf{x} = \mathbf{0}$$

$$\rightarrow \exists A^{-1}A = \mathbf{I} = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$$

$$A\mathbf{x} = \mathbf{b} \rightarrow \mathbf{x} = A^{-1}\mathbf{b}$$

Function Approximation

Function approximation

Given $f : [a, b] \subset \mathbb{R} \to \mathbb{R}$ define $g : [a, b] \subset \mathbb{R} \to \mathbb{R}$ such that: $g([a, b]) \approx f([a, b])$

- f too expensive to compute
- f unknown, but can be measured on a grid like $a < x_0 < x_1 < ... < x_n < b$
- f is noisy or has random components

Polynomial interpolation

Given (x_i, f_i), $i = 0, 1, ..., n$, $f_i \equiv x_i$:
find a polynomial $p(x) \in \Pi$ such that $p(x_i) = f_i$

Canonical Base

Theorem ($\exists! p(x) \in \Pi_n$ such that $p(x_i) = f_i,\ i = 0, ..., n$)

$$p(x) = \sum_{k=0}^{n} a_k x^k \implies V\mathbf{a} = \mathbf{f}$$

where:

$$V = \begin{pmatrix} x_0^0 & x_0^1 & \cdots & x_0^n \\ x_1^0 & x_1^1 & \cdots & x_1^n \\ \vdots & \vdots & & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^n \end{pmatrix}, \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}, \mathbf{f} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}$$

V is a Vandermonde matrix: $\det(v) = \prod_{i>j}(x_i - x_j) \neq 0 \iff x_i \neq x_j \forall i, j$

$$\mathbf{a} = V^{-1}\mathbf{f}$$

Conditioning

Vandermonde matrices $\in M^{n \times n}$ have bad conditioning already for small n!

$$|p(x) - \tilde{p}(x)| = \kappa \max_i |f_i - \tilde{f}_i|, \quad \kappa >> 1$$

It's complicated, see http://arxiv.org/abs/1504.02118v3 and https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://osnadocs.ub.uni-osnabrueck.de/bitstream/urn:nbn:de:gbv:700-202103194121/1/thesis_nagel.pdf&ved=2ahUKEwjItvWklM2JAxXB3QIHHQgnD_MQFnoECBUQAQ&usg=AOvVaw3OFier08PagrHVx9c8XiKj to get an idea!

Lagrange form

$$L_{kn}(x) = \prod_{j=0,\ j\neq k}^{n-1} \frac{x - x_j}{x_k - x_j}, \quad k = 0, .., n-1$$

Observation

$$L_{kn}(x_i) = \delta_{ki}: \quad 1 \iff k = i,\ 0 \textit{ otherwise}$$

$$L_{kn} \equiv \Pi_n, \quad \textit{linearly independent}$$

$$p(x) = \sum_{k=0}^{n-1} f_k L_{kn}(x)$$

Problem Conditioning

Assumptions:

- $\{x_i\}$ can be usually chosen arbitrarily
- error on $\{x_i\}$ usually small
- $\{x_i\}$ usually common to many functions in real world (quantities measured at regular intervals)
- erorr on $\{f_i\}$ usually bigger

Consider:

$$p(x) = \sum_{k=0}^{n} f_k L_{kn}(x), \quad \tilde{p}(x) = \sum_{k=0}^{n} \tilde{f}_k L_{kn}(x)$$

Problem Conditioning (2)

$$
\begin{aligned}
|p(x) - \tilde{p}(x)| &= |\sum_{k=0}^{n} f_k L_{kn}(x) - \sum_{k=0}^{n} \tilde{f}_k L_{kn}(x)| \\
&= |\sum_{k=0}^{n} (f_k - \tilde{f}_k) L_{kn}(x)| \leq \sum_{k=0}^{n} |(f_k - \tilde{f}_k)| \cdot |L_{kn}(x)| \\
&\leq \left(\sum_{k=0}^{n} |L_{kn}(x)| \right) \max_k |f_k - \tilde{f}_k| = \Lambda_n \max_k |f_k - \tilde{f}_k|
\end{aligned}
$$

- Λ_n depends only on $[a, b]$ and $\{x_i\}$
- $\max_k |f_k - \tilde{f}_k|$ is an input error

Problem Conditioning (3)

- if $[a, b]$ can be mapped in $[c, d]$ with a linear transformation, Λ_n does not change for the two spaces
- $\Lambda_n \geq O(logn) \rightarrow \infty$ for $n \rightarrow \infty$
- Equidistant $\{x_i\}$ generate an approximately exponential sequence $\{\Lambda_n\}$ for $n \rightarrow \infty$

Piecewise interpolation

Define $q : [a, b] \subset \mathbb{R} \to \mathbb{R}$ such that:

$$q(x_i) = f(x_i) \tag{1}$$

$$q(x) = q_i(x) \in \Pi_k, \quad x_i \leq x \leq x_{i+1}, \ i = 0, ..., n$$

We also impose:

$$q_i^{(s)}(x_{i+1}) = q_{i+1}^{(s)}(x_{i+1}), \quad s = 0, .., k-1, \ i = 1, ..., n-1$$

- $q(x) \in \Pi_k$ over $[a, b]$ is called *spline* of degree k, interpolating f when subject to (1)
- $q(x) \in \mathcal{C}^k$
- $q'(x) \in \mathcal{C}^{k-1}$

Spline

Observation

$q_i(x) \in \Pi_k$ has $k+1$ degrees of freedom
Given a partition with $n+1$ points $x_0, ..., x_n$, the spline $q(x)$ has $k+n$ degrees of freedom

Each of the n polynomials q_i has $k+1$ degrees of freedom, for a total of $n(k+1)$ free coefficients. The k derivative continuity condition impose $k(n-1)$ parameters on the $n-1$ internal points $x_1, ..., x_{n-1}$, for a total of $k+n$ free parameters left.

- A spline of degree 1 (joint line) has $n+1$ free parameters: exactly the interpolation points!
- For degree > 1, one has to impose $k-1$ additional conditions.

Cubic splines

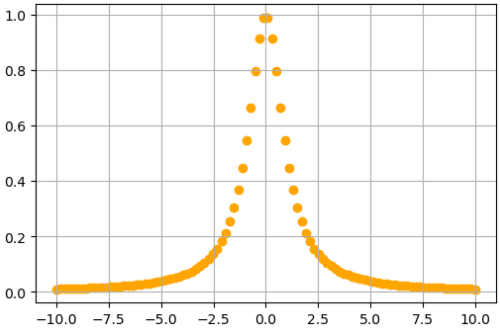
$q_i(x) \in \Pi_3$: $3 - 1 = 2$ additional conditions to impose, on $[a = x_0, x_1, ..., x_n = b]$

- Natural spline: $q''(a) = q''(b) = 0$
- Complete spline: $q'(a) = f'(a)$, $q'(b) = f'(b)$
- Periodic spline: $q'(a) = q'(b)$, $q''(a) = q''(b)$
- Not-a-knot: $q_0'''(x_1) = q_1'''(x_1)$, $q_{n-1}'''(x_{n-1}) = q_n'''(x_{n-1})$

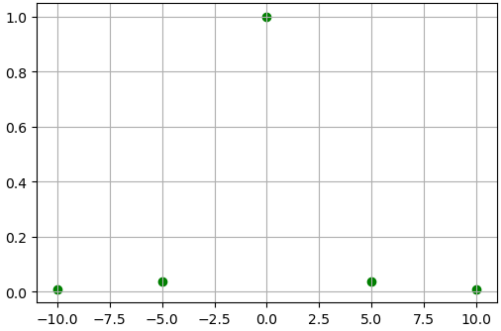
Implementation

```
1  import numpy as np
2  import math
3  from scipy import interpolate
4  from matplotlib import pyplot as plt
5  plt.rcParams['axes.grid'] = True
```

```
1  def f(x):
2      return 1/(1+x**2)
3
4  X = np.linspace(-10,10,100 )
5  Y = f(X)
6
7  plt.scatter(X,Y, color='orange');
```



```
1  X = np.linspace(-10,10,5 )
2  Y = f(X)
3
4  plt.scatter(X,Y, color='green');
```



```
1 def p(x, a):
2     return sum([a[k]*x**k for k in range(len(a))])
```

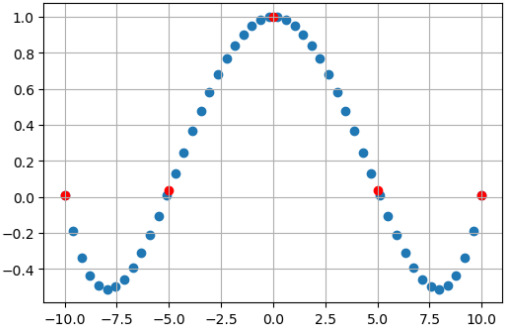
```
print(X)
V = np.array([X**i for i in range(len(X))] ).transpose()
print(V)
```

```
[-10.  -5.   0.   5.  10.]
[[ 1.00e+00 -1.00e+01  1.00e+02 -1.00e+03  1.00e+04]
 [ 1.00e+00 -5.00e+00  2.50e+01 -1.25e+02  6.25e+02]
 [ 1.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00]
 [ 1.00e+00  5.00e+00  2.50e+01  1.25e+02  6.25e+02]
 [ 1.00e+00  1.00e+01  1.00e+02  1.00e+03  1.00e+04]]
```

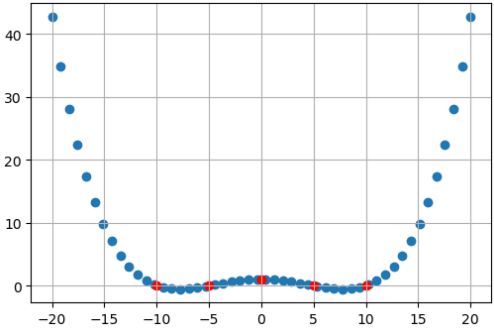
```
a = np.linalg.inv(V).dot(Y)
print(a)
```

```
[ 1.00000000e+00  9.53596960e-21 -4.79817212e-02 -3.55943931e-22
  3.80807312e-04]
```

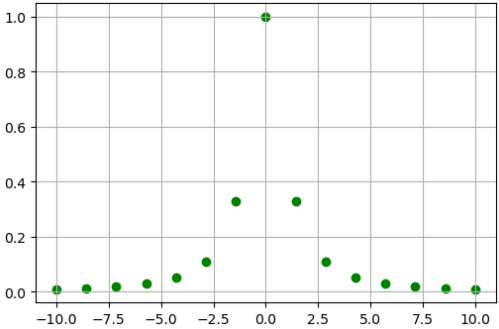
```
1  x = np.linspace(-10,10,50)
2  y = p(x, a)
3  plt.scatter(x,y);
4  plt.scatter(X,Y, color='red');
```



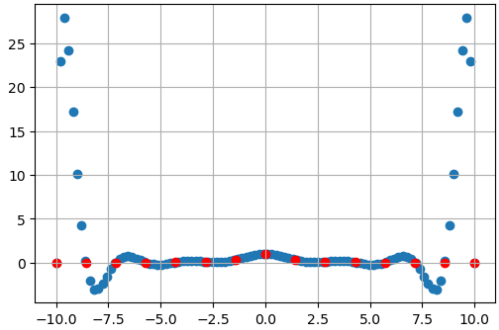
```
1  x = np.linspace(-20,20,50)
2  y = p(x, a)
3  plt.scatter(x,y);
4  plt.scatter(X,Y, color='red');
```



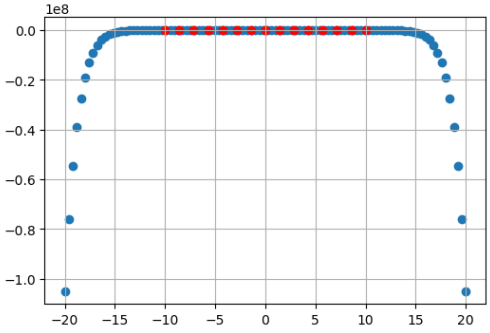
```
1  X = np.linspace(-10,10,15)
2  Y = f(X)
3
4  plt.scatter(X,Y, color='green');
```



```
V = np.array([X**i for i in range(len(X))] ).transpose()
a = np.linalg.inv(V).dot(Y)
x = np.linspace(-10,10,100)
y = p(x, a)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```



```
1  x = np.linspace(-20,20,100)
2  y = p(x, a)
3  plt.scatter(x,y);
4  plt.scatter(X,Y, color='red');
```

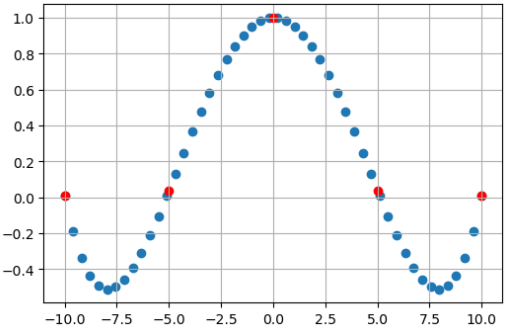


```python
def Lkn(Xi,k,x):
    x_k = Xi[k]
    X = list(Xi[:k])+list(Xi[k+1:])
    res = 1.
    for x_j in X:
        res *= (x-x_j)/(x_k-x_j)
    return res

```

```
X = np.linspace(-10, 10, 5)
x = np.linspace(-10,10, 50)

for k in range(0,5):
    Y = Lkn(X,k,x)
    plt.plot(x,Y);
```

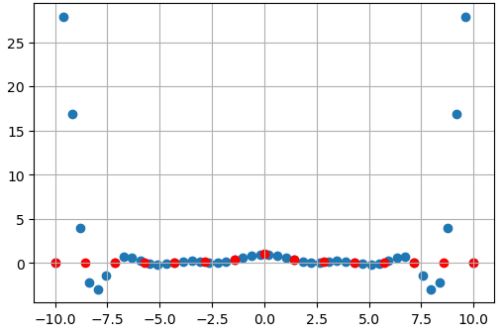
```
1  def pl(x,Xi,f):
2      return sum([f[k]*Lkn(Xi,k,x) for k in range(len(f))])
```

```
X = np.linspace(-10, 10, 5)
Y = f(X)

x = np.linspace(-10,10,50)
y = pl(x, X, Y)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```

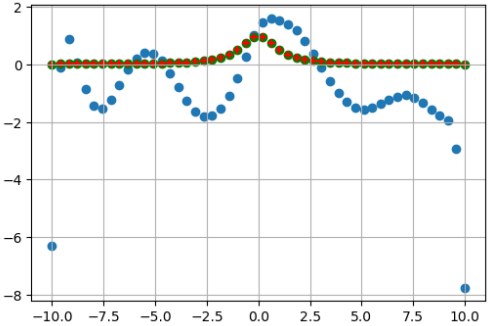


```
1  X = np.linspace(-10, 10, 15)
2  Y = f(X)
```

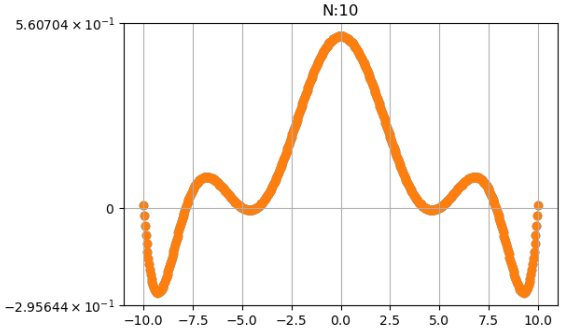
```
1  x = np.linspace(-10,10,50)
2  y = pl(x, X, Y)
3  plt.scatter(x,y);
4  plt.scatter(X,Y, color='red');
```

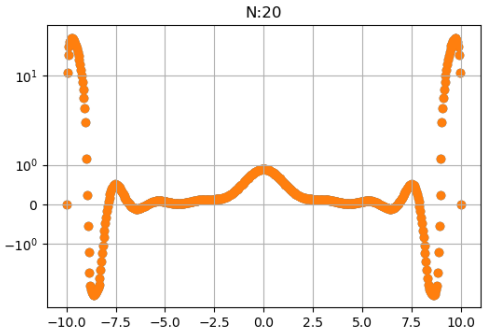


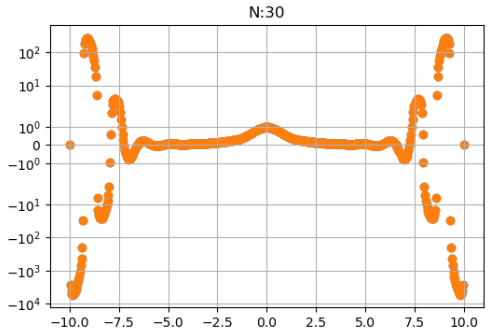
```
X = np.linspace(-10, 10, 50)
Y = f(X)

x = np.linspace(-10,10,50)

V = np.array([X**i for i in range(len(X))] ).transpose()
a = np.linalg.inv(V).dot(Y)
yp = p(x, a)

yl = pl(x, X, Y)

plt.scatter(x,yp);
plt.scatter(x,yl, color='green');
plt.scatter(X,Y, color='red', marker='+');
```

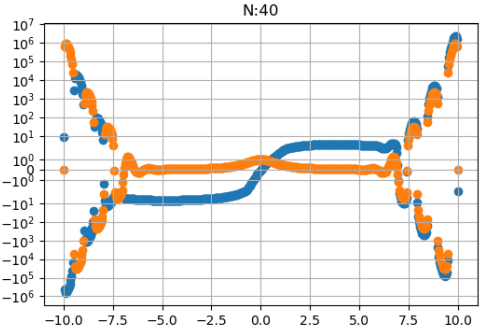

2
0
−2
−4
−6
−8
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

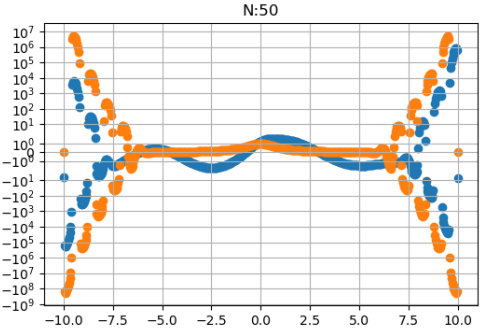
```
for n in range(10,100, 10):
    X = np.linspace(-10, 10, n)
    Y = f(X)

    x = np.linspace(-10,10,500)

    V = np.array([X**i for i in range(len(X))] ).transpose()
    a = np.linalg.inv(V).dot(Y)
    yp = p(x, a)

    yl = pl(x, X, Y)
    plt.figure()

    plt.scatter(x,yp);
    plt.scatter(x,yl);
    plt.yscale('symlog')
    plt.title('N:'+str(n))
```

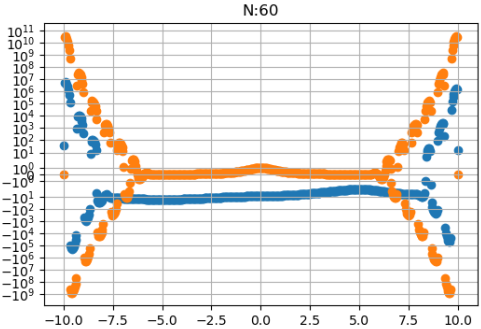

N:10
5.60704 × 10−1
0
−2.95644 × 10−1
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

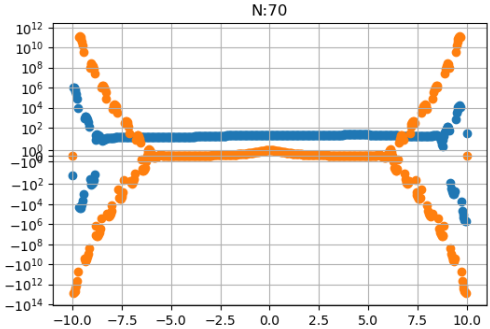

N:20
10^1
10^0
0
-10^0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

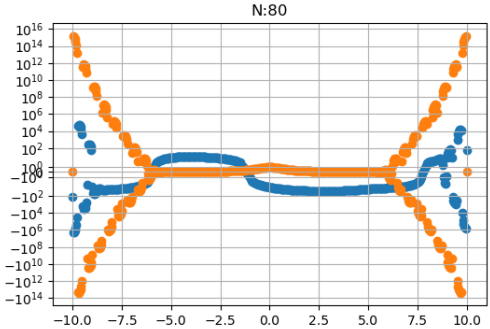

N:30
10^2
10^1
10^0
0
-10^0
-10^1
-10^2
-10^3
-10^4
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

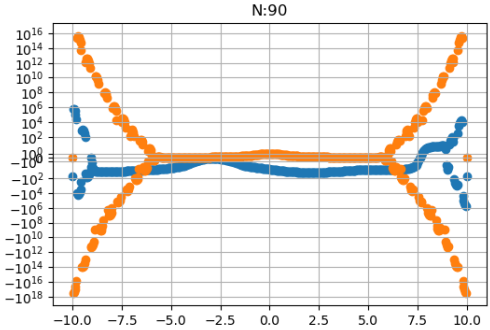

N:40
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
0
-10^0
-10^1
-10^2
-10^3
-10^4
-10^5
-10^6
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


N:50
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
-10^{0}
-10^{1}
-10^{2}
-10^{3}
-10^{4}
-10^{5}
-10^{6}
-10^{7}
-10^{8}
-10^{9}
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

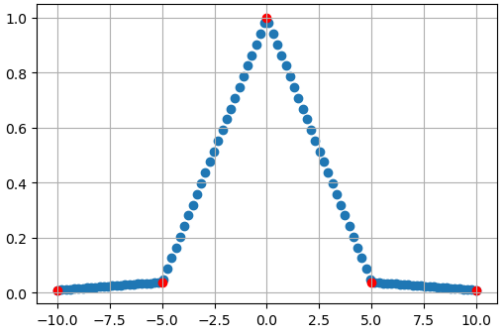

N:60
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
-10^{9}
-10^{1}
-10^{2}
-10^{3}
-10^{4}
-10^{5}
-10^{6}
-10^{7}
-10^{8}
-10^{9}
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0


N:70
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
-10^{0}
-10^{2}
-10^{4}
-10^{6}
-10^{8}
-10^{10}
-10^{12}
-10^{14}
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

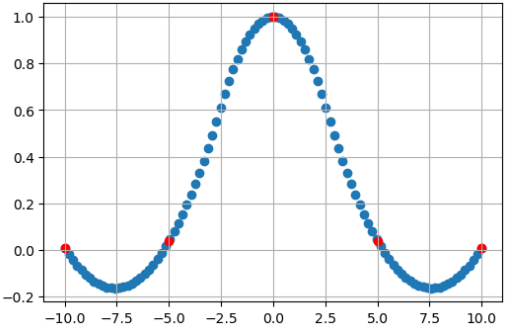

N:80
10^{16}
10^{14}
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
-10^{0}
-10^{2}
-10^{4}
-10^{6}
-10^{8}
-10^{10}
-10^{12}
-10^{14}
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


N:90
10^{16}
10^{14}
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
-10^{0}
-10^{2}
-10^{4}
-10^{6}
-10^{8}
-10^{10}
-10^{12}
-10^{14}
-10^{16}
-10^{18}
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

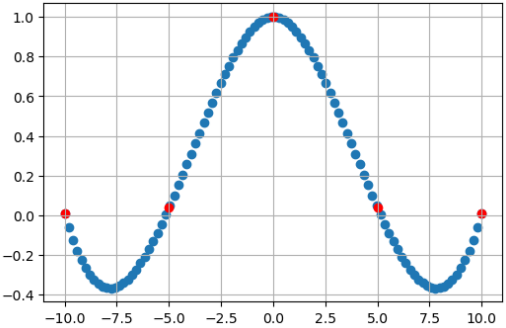
```
X = np.linspace(-10, 10, 5)
Y = f(X)

x = np.linspace(-10,10,100)
spline = interpolate.interp1d(X,Y, kind='linear')
y = spline(x)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```



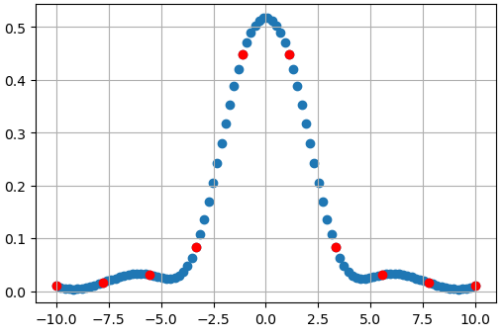
```
1  x = np.linspace(-10,10,100)
2  spline = interpolate.interp1d(X,Y, kind='quadratic')
3  y = spline(x)
4  plt.scatter(x,y);
5  plt.scatter(X,Y, color='red');
```



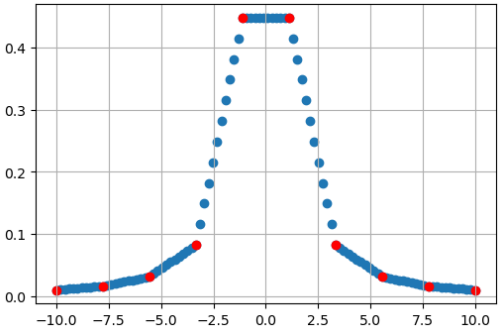
```
x = np.linspace(-10,10,100)
spline = interpolate.interp1d(X,Y, kind='cubic')
y = spline(x)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```



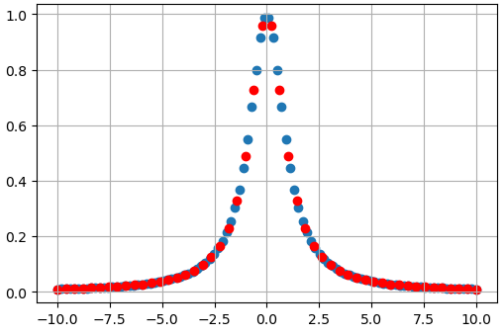
```
X = np.linspace(-10, 10, 10)
Y = f(X)
x = np.linspace(-10,10,100)
spline = interpolate.interp1d(X,Y, kind='cubic')
y = spline(x)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```



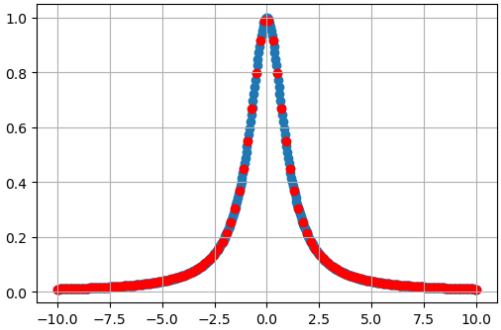
```
1  spline = interpolate.interp1d(X,Y, kind='linear')
2  y = spline(x)
3  plt.scatter(x,y);
4  plt.scatter(X,Y, color='red');
```



```
X = np.linspace(-10, 10, 50)
Y = f(X)
x = np.linspace(-10,10,100)
spline = interpolate.interp1d(X,Y, kind='cubic')
y = spline(x)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```



```
X = np.linspace(-10, 10, 100)
Y = f(X)
x = np.linspace(-10,10,500)
spline = interpolate.interp1d(X,Y, kind='cubic')
y = spline(x)
plt.scatter(x,y);
plt.scatter(X,Y, color='red');
```



Least squares approximation

Least square approximation

Data in the form of noisy measurements:

$$(x_i, y_i), \quad i = 0, ..., n$$

Let:

$$\mathbf{y} = (y_0, ..., y_n)^T, \ \mathbf{x} = (x_0, ..., x_n)^T, \ \mathbf{z} = f(\mathbf{x}) = (z_0, ..., z_n)^T$$

where f is the approximating function (be it polynomial, spline, etc.). We want to minimize:

$$\|y - z\|_2^2 = \sum_{i=0}^{n} |y_i - z_i|^2$$

Repeated measurements

Multiple measurements of the same point (example: $R = V/I$ multiple measures of I for each V)

$$(x_i^{[k]}, y_i^{[k]})$$

define:

$$\mathbf{x} = (x_0, ..., x_n)^T,$$

where $x_i \neq x_j$, and:

$$\mathbf{y} = \begin{pmatrix} \sum_{j=0}^{k} y_0^{[j]}/k \\ \vdots \\ \sum_{j=0}^{k} y_n^{[j]}/k \end{pmatrix}$$

Overdetermined Polynomial Approximation

Given $\mathbf{y} = (y_0, ..., y_n)^T$, $\mathbf{x} = (x_0, ..., x_n)^T$

$$p(x) \in \Pi_d = \sum_{k=0}^{d} a_k x^k \implies V\mathbf{a} = \mathbf{y}$$

where:

$$V = \begin{pmatrix} x_0^0 & x_0^1 & \cdots & x_0^d \\ x_1^0 & x_1^1 & \cdots & x_1^d \\ \vdots & \vdots & & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^d \end{pmatrix}, \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{pmatrix}, \mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

with $n >> d$. $V \in \mathbb{R}^{n \times d}$ is singular and not invertible!

$$V\mathbf{a} = \mathbf{y} \quad ?$$

Observation

V has maximum rank $(m+1)$: the first $m+1$ rows is a Vandermonde matrix, which is non-singular.

QR Decomposition

$$V\mathbf{a} = \mathbf{y} + \mathbf{r}$$

such that $\|r\|_2^2 = \|V\mathbf{a} - \mathbf{y}\|_2^2$ is minimized (least squares).

$$V = QR = Q\begin{pmatrix}\hat{R}\\ O\end{pmatrix}, \quad V \in \mathbb{R}^{n\times d}, Q \in \mathbb{R}^{n\times n}, R \in \mathbb{R}^{n\times d}$$

where Q orthogonal ($Q^TQ = QQ^T = I$), $\hat{R} \in \mathbb{R}^{d\times d}$ upper triangular and nonsingular.

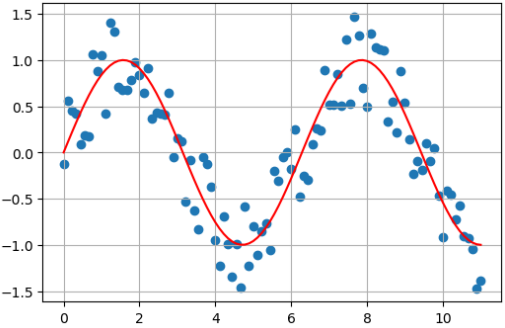
$$\|r\|_2^2 = \|V\mathbf{a} - \mathbf{y}\| = \|QR\mathbf{a} - \mathbf{y}\| = \|Q(R\mathbf{a} - Q^T\mathbf{y})\| = \|Q(R\mathbf{a} - \mathbf{g})\| = \left\|Q\begin{pmatrix}\hat{R}\\ O\end{pmatrix}\mathbf{a} - \begin{pmatrix}\mathbf{g}_u\\ \mathbf{g}_l\end{pmatrix}\right\|$$

$$\mathbf{a} = \hat{R}^{-1}\mathbf{g}_u$$

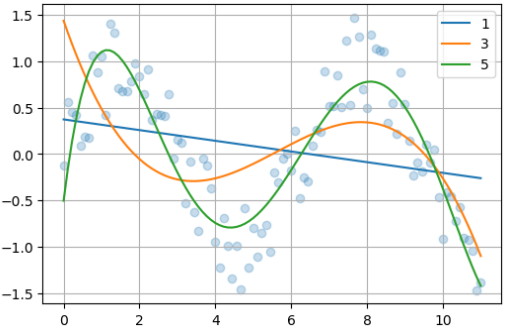
Implementation

```
1  import numpy as np
2  import math
3  from scipy import interpolate, optimize
4  from matplotlib import pyplot as plt
5  plt.rcParams['axes.grid'] = True
```

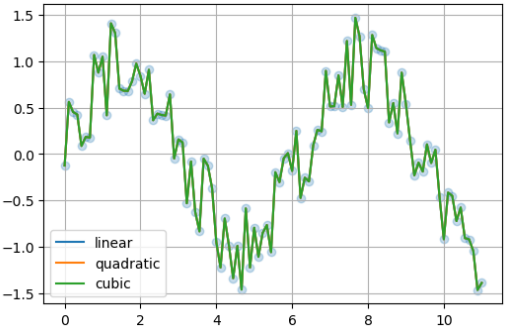
```
1  np.random.seed(42)
2  Xmin, Xmax = 0, 3.5*np.pi
3  X = np.linspace(Xmin, Xmax, 100)
4  Y = np.sin(X) + np.random.random(len(X))-0.5
5  plt.scatter(X,Y);
6  plt.plot(X, np.sin(X), color='red');
```



```
f1 = np.polynomial.Polynomial.fit(X,Y,1)
f3 = np.polynomial.Polynomial.fit(X,Y,3)
f5 = np.polynomial.Polynomial.fit(X,Y,5)
plt.scatter(X,Y, alpha=0.25)
plt.plot(X,f1(X), label='1')
plt.plot(X,f3(X), label='3')
plt.plot(X,f5(X), label='5')
plt.legend();
```

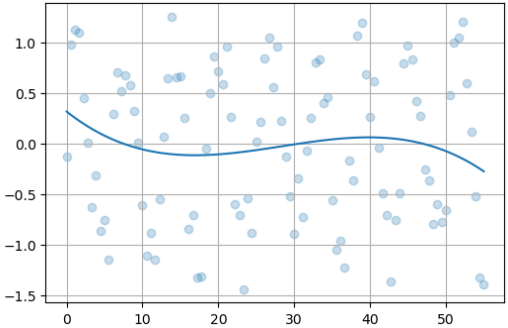


```
1  flin = interpolate.interp1d(X,Y, kind='linear')
2  fquad = interpolate.interp1d(X,Y, kind='quadratic')
3  fcub = interpolate.interp1d(X,Y, kind='cubic')
4
5  plt.scatter(X,Y, alpha=0.25)
6  plt.plot(X,flin(X), label='linear')
7  plt.plot(X,fquad(X), label='quadratic')
8  plt.plot(X,fcub(X), label='cubic')
9  plt.legend();
```



```python
def curve(x, a,b,c,d):
    return a*x**3+b*x**2+c*x+d
popt, pcov = optimize.curve_fit(curve, X,Y)
print(popt)
print(pcov)

plt.scatter(X,Y, alpha=0.25)
plt.plot(X, curve(X, *popt));
```

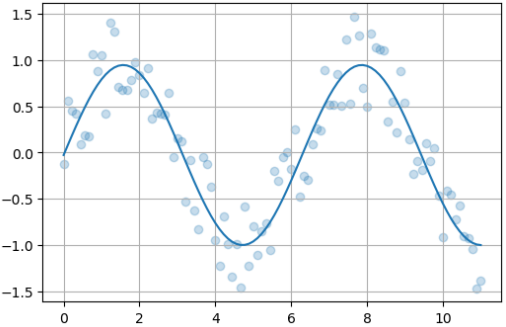
```
[-2.94751365e-05  2.50738081e-03 -5.95113827e-02  3.16497965e-01]
[[ 5.60069341e-10 -4.61871304e-08  1.01061274e-06 -4.51339126e-06]
 [-4.61871304e-08  3.91984072e-06 -8.94410056e-05  4.27525473e-04]
 [ 1.01061274e-06 -8.94410056e-05  2.18170101e-03 -1.18122707e-02]
 [-4.51339126e-06  4.27525473e-04 -1.18122707e-02  8.70448263e-02]]
```


1.0
0.5
0.0
−0.5
−1.0
−1.5
0
10
20
30
40
50

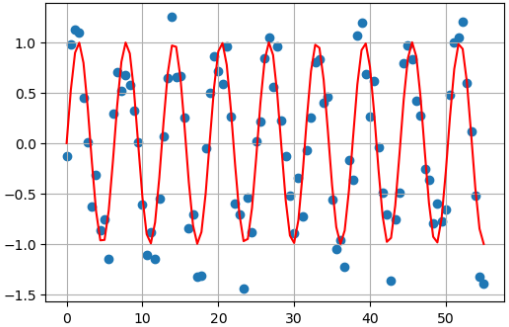
```
import inspect
inspect.signature(curve).parameters
```

```
mappingproxy({'x': <Parameter "x">,
              'a': <Parameter "a">,
              'b': <Parameter "b">,
              'c': <Parameter "c">,
              'd': <Parameter "d">})
```

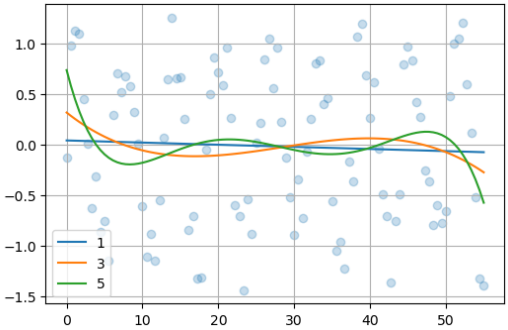
```
1  def curve(x, a,b,c):
2      return  a*np.sin(b*x)+c
3  popt, pcov = optimize.curve_fit(curve, X,Y)
4
5  plt.scatter(X,Y, alpha=0.25)
6  plt.plot(X, curve(X, *popt));
```



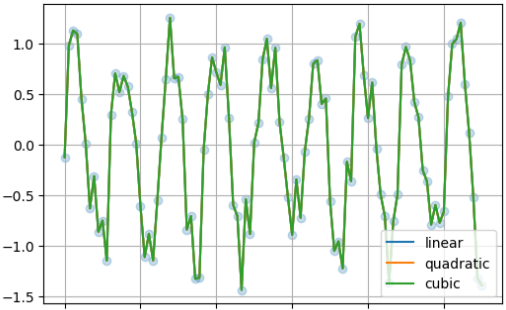
```
1  np.random.seed(42)
2  Xmin, Xmax = 0, 5*3.5*np.pi
3  X = np.linspace(Xmin, Xmax, 100)
4  Y = np.sin(X) + np.random.random(len(X))-0.5
5  plt.scatter(X,Y);
6  plt.plot(X, np.sin(X), color='red');
```



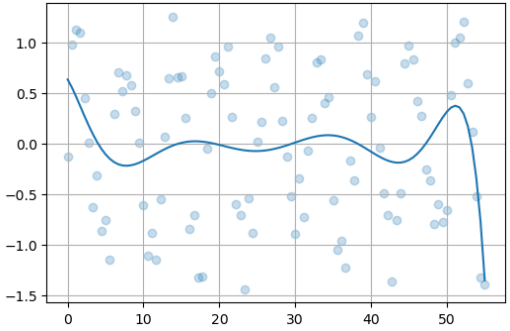
```
f1 = np.polynomial.Polynomial.fit(X,Y,1)
f3 = np.polynomial.Polynomial.fit(X,Y,3)
f5 = np.polynomial.Polynomial.fit(X,Y,5)
plt.scatter(X,Y, alpha=0.25)
plt.plot(X,f1(X), label='1')
plt.plot(X,f3(X), label='3')
plt.plot(X,f5(X), label='5')
plt.legend();
```



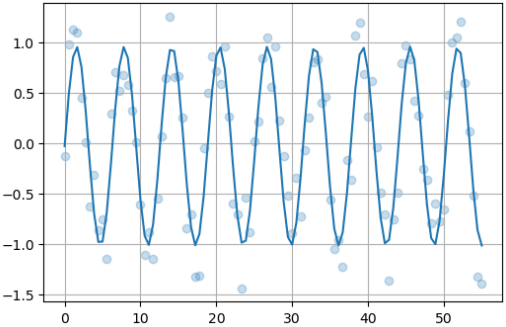
```
1  flin = interpolate.interp1d(X,Y, kind='linear')
2  fquad = interpolate.interp1d(X,Y, kind='quadratic')
3  fcub = interpolate.interp1d(X,Y, kind='cubic')
4
5  plt.scatter(X,Y, alpha=0.25)
6  plt.plot(X,flin(X), label='linear')
7  plt.plot(X,fquad(X), label='quadratic')
8  plt.plot(X,fcub(X), label='cubic')
9  plt.legend();
```



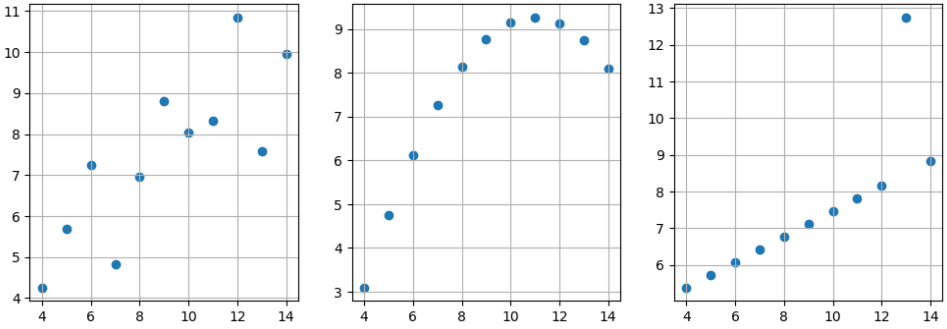
```
1  def curve(x, a,b,c,d,e,f,g,h,i):
2      params = [a,b,c,d,e,f,g,h,i]
3      terms = [params[i]*x**i for i in range(len(params))]
4      return sum(terms)
5  popt, pcov = optimize.curve_fit(curve, X,Y)
6
7  plt.scatter(X,Y, alpha=0.25)
8  plt.plot(X, curve(X, *popt));
```



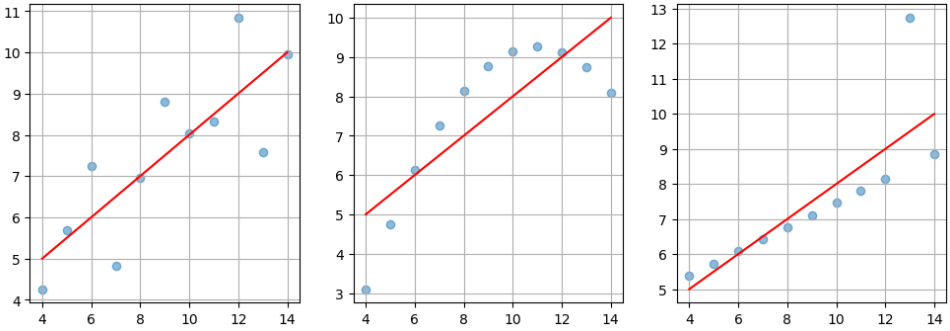
```
1  def curve(x, a,b,c):
2      return  a*np.sin(b*x)+c
3  popt, pcov = optimize.curve_fit(curve, X,Y)
4
5  plt.scatter(X,Y, alpha=0.25)
6  plt.plot(X, curve(X, *popt));
```



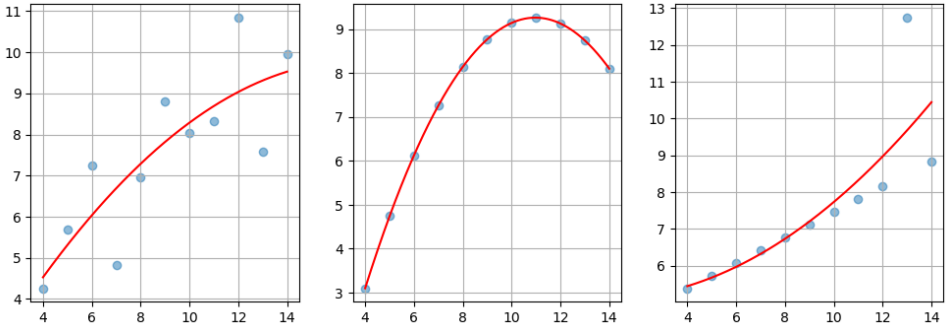
```
x = [10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5]
y1 = [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68]
y2 = [9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74]
y3 = [7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73]

fbase = np.linspace(min(x), max(x), 1000)
fig, (ax1,ax2,ax3) = plt.subplots(1,3)
fig.set_size_inches(12,4)
for ax,y in [(ax1,y1),(ax2,y2),(ax3,y3)]:
    ax.scatter(x,y)
```



```
f1 = np.polynomial.Polynomial.fit(x,y1,1)
f2 = np.polynomial.Polynomial.fit(x,y2,1)
f3 = np.polynomial.Polynomial.fit(x,y3,1)
fbase = np.linspace(min(x), max(x), 1000)
fig, (ax1,ax2,ax3) = plt.subplots(1,3)
fig.set_size_inches(12,4)
for ax,f,y in [(ax1,f1,y1),(ax2,f2,y2),(ax3,f3,y3)]:
    ax.scatter(x,y, alpha=0.5, label='Data')
    ax.plot(fbase,f(fbase), color='r')
```



```
f1 = np.polynomial.Polynomial.fit(x,y1,2)
f2 = np.polynomial.Polynomial.fit(x,y2,2)
f3 = np.polynomial.Polynomial.fit(x,y3,2)
fbase = np.linspace(min(x), max(x), 1000)
fig, (ax1,ax2,ax3) = plt.subplots(1,3)
fig.set_size_inches(12,4)
for ax,f,y in [(ax1,f1,y1),(ax2,f2,y2),(ax3,f3,y3)]:
    ax.scatter(x,y, alpha=0.5, label='Data')
    ax.plot(fbase,f(fbase), color='r')
```



Optimization

Definition

Given $\mathbf{x} \in \mathbb{C}^n$, $f : \mathbb{C}^n \to \mathbb{R}$, $c_i : \mathbb{C} \to \{T, F\}$

Definition

$$\min_{\mathbf{x} \in \mathbb{C}^n} f(\mathbf{x}), \quad \text{subject to } c_i(x_i) = T$$

Example

$$\mathbf{x} \in \mathbb{R}^2, \quad f(\mathbf{x}) = \left[\mathbf{x} - \begin{pmatrix} k_1 \\ k_2 \end{pmatrix}\right]^{2/3}, \quad \text{subject to } c(\mathbf{x}) = \begin{pmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_k(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} -x_1 + x_2^2 \geq 0 \\ \vdots \\ \|\mathbf{x}\|_2^2 < 0.5 \end{pmatrix}$$

Example: Distribution problem

- k factories F_i produce each p_i tons of product
- n users (receivers) R_i need each b_i tons of product
- the cost of the product from F_i to R_j is c_{ij}
- the amount of product moved from F_i to R_j is x_{ij}

$$\min \sum_{ij}(c_{ij}x_{ij}) \text{ subject to } \begin{pmatrix} \sum_{j=1}^{n} x_{ij} \leq p_i, & i = 1, ..., k \\ \sum_{i=1}^{k} x_{ij} \geq b_j, & j = 1, ..., n \\ x_{ij} \geq 0 \end{pmatrix}; \quad \left(\text{or } \min \sum_{ij}(c_{ij}\sqrt{\delta + x_{ij}}), \delta > 0 \right)$$

$$\mathbf{x} \in \mathbb{R}^{kn} = \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \\ x_{1,1} \\ \vdots \\ x_{k,n} \end{pmatrix}, \quad c(\mathbf{x}) = \begin{pmatrix} \sum_{j=1}^{n} x_{1j} \leq p_1 \\ \sum_{j=1}^{n} x_{2j} \leq p_2 \\ \vdots \\ \sum_{i=0}^{k} x_{i1} \geq b_1 \\ \sum_{i=0}^{k} x_{i2} \geq b_2 \\ \vdots \\ x_{1,1} \geq 0 \\ x_{1,2} \geq 0 \\ \vdots \end{pmatrix}$$

Other Examples

Portfolio optimization:

- variables: amounts invested in different assets
- constraints: budget, investment limits per assets, minimum return
- objectives: overall risk, return variance...

Electronic circuits component placement:

- variables: components size, routing
- constraints: manufacturing limits, board size, assembly time, manual labor requirements
- objectives: uniform heat dissipation, reliability, cost...

Machine Learning:

- variables: model parameters
- constraints: parameter limits, data, computational cost, time...
- objectives: prediction accuracy, classification accuracy, generalization...

Continuous vs Discrete optimization

- Distribution problem (could be both)
- ML tasks (usually continuous)
- Number of power plants to build
- In which city to place a factory
- Mixed problems (some integer variables, some continuous)
- Discrete problems are generally more difficult to solve, there are special techniques

  When to approximate? During optimization or after? Depends on the problem...

Constrained vs Unconstrained optimization

- There are *always* constraints!
- ... but they are costly to check!
- ... but sometimes they don't affect the solution, so the can be "safely" ignored
- ... but sometimes they can be reformulated as part of the cost function!
- linear cost and constraint much easier to deal with
- nonlinear constraints typical of physical world, costly
- usually worth understanding which constraints can be dropped or represented in the cost function

Global vs Local optimization

Definition (Global optimization)

$$\min f(\mathbf{x}) \text{ where } \mathbf{x} \in \text{Domain}$$

Definition (Local optimization)

$$\min f(\mathbf{x}) \text{ where } \mathbf{x} \in \text{Neighborhood of starting point within the Domain}$$

- Generally, global optimization requirese infinite checks (depending on properties of f)
- Global optimization is approximated with multiple local optimizations
- Convex problems surely admit a global minimum

Stochastic vs Deterministic optimization

Method:

- Deterministic methods always obtain the same exact result given the same conditions
- Stochastic methods use a (pseudo)random component (Genetic algorithm, montecarlo etc.) (seed!)

Problem:

- Some problems have stochastic components (economic model depending on future demand)
- Instead of "best guess", model probability into the model!
  - know scenarios with associated probabilities
  - produce expected performance with associated probability

Convexity

Definition (Convex set)

$$S \in \mathbb{R}^n \text{ such that } \forall x, y \in S,\ \alpha x + (1-\alpha)y \in S, \forall \alpha \in [0,1]$$

- A convex set is such that a straight line segment between any two points in S are also in S.
- Example: the unit sphere $\{\mathbf{y} \in \mathbb{R}^n | \|\mathbf{y}\|_2 \leq 1\}$
- Example: a polyhedron $\{\mathbf{x} \in \mathbb{R}^n | A\mathbf{x} = \mathbf{b}, C\mathbf{x} \leq \mathbf{d}\}$

Definition (Convex function)

$f : S \to \mathbb{R}$ is convex if S is convex and

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y})) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}), \quad \forall \alpha \in [0,1]$$

- Example: $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + k$
- Example: $f(\mathbf{x}) = \mathbf{x}^T H \mathbf{x}$ with H symmetric positive semidefinite

Multivariate Calculus Recap

Partial Derivatives

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\frac{\partial f}{\partial x_k} = \frac{\partial}{\partial x_k} f = f_{x_k}(\mathbf{x}) = D_{x_k} f = \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{e_k}) - f(\mathbf{x})}{h}$$

when $\exists f_{k_k}(\mathbf{x}) \neq \pm\infty$, and where $\mathbf{e}_k$ is the k-th unit versor.

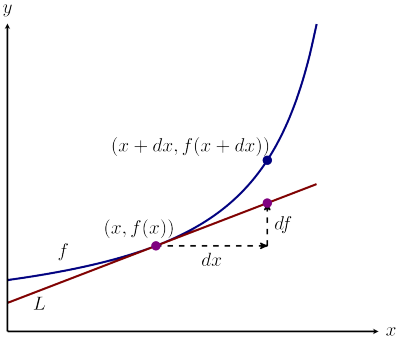
Gradient:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

First order approximation

$$f(x+h) = f(x) + f'(x)h + O(h^2)$$

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{y} + O(\|\mathbf{y}\|)$$



Second Derivative

If $\nabla f(\mathbf{x})$ are still all derivable with respect to all components of $\mathbf{x}$:

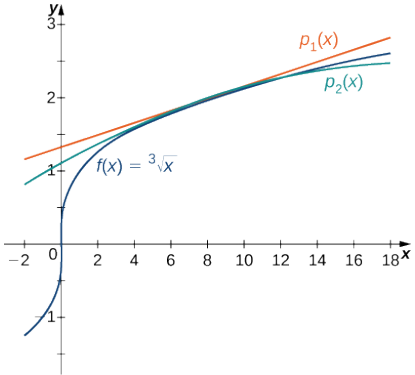
$$\nabla^2 f(\mathbf{x}) = \nabla(\nabla f(\mathbf{x})^T) = \nabla \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \cdots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right) = \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_n} \end{pmatrix}$$

$\nabla^2 f(\mathbf{x})$ is called *Hessian*, and is a symmetric matrix

Second order approximation

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3)$$

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{y} + \frac{1}{2}\mathbf{y}^T \nabla^2 f(\mathbf{x})\mathbf{y} + O(\|\mathbf{y}\|^2)$$

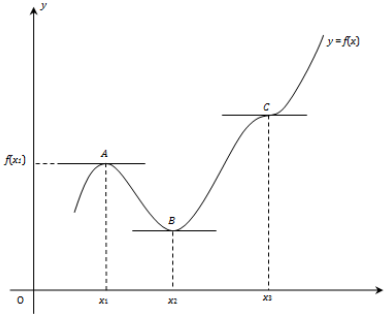


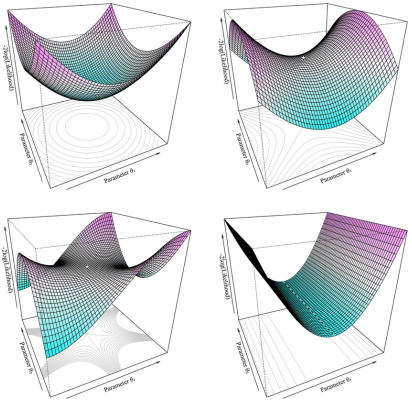
Stationary points

Stationary point $\bar{\mathbf{x}}$ of $f : \mathbb{R}^n \to \mathbb{R}$ is such that:

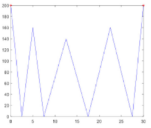
$$\nabla f(\bar{\mathbf{x}}) = 0$$

- candidates for local/global max/min!

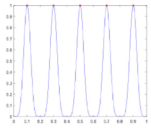




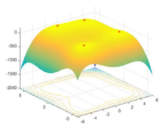
Function Examples, local and global maxima



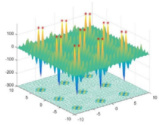
(a) F1



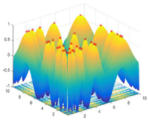
(b) F2



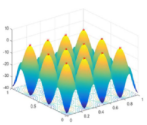
(c) F4



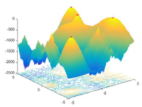
(d) F6



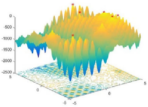
(e) F7



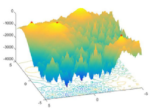
(f) F10



(g) F11



(h) F12



(i) F13

Definitions

$f : S \to \mathbb{R}$

Definition (Global Minimizer)

$\bar{\mathbf{x}}$ for which $f(\bar{\mathbf{x}}) \leq f(\mathbf{x}) \; \forall \mathbf{x} \in S$

Definition (Local Minimizer)

$\bar{\mathbf{x}}$ for which $\exists \mathcal{N}(\bar{\mathbf{x}}) \mid f(\bar{\mathbf{x}}) \leq f(\mathbf{x}) \; \forall \mathbf{x} \in \mathcal{N}(\bar{\mathbf{x}})$

Definition (Strict Local Minimizer)

$\bar{\mathbf{x}}$ for which $\exists \mathcal{N}(\bar{\mathbf{x}}) \mid f(\bar{\mathbf{x}}) < f(\mathbf{x}) \; \forall \mathbf{x} \in \mathcal{N}(\bar{\mathbf{x}})$ with $\mathbf{x} \neq \bar{\mathbf{x}}$

Recognizing local minimum

Theorem (First order necessary conditions)

If $\bar{\mathbf{x}}$ is a local minimizer and f is continuously differentiable in an open neighbour $\mathcal{N}(\bar{\mathbf{x}})$, then $\nabla f(\bar{\mathbf{x}}) = 0$

Theorem (Second order necessary condition)

If $\bar{\mathbf{x}}$ is a local minimizer and $\nabla^2 f$ exists and is continuously differentiable in an open neighbourhood $\mathcal{N}(\bar{\mathbf{x}})$, then $\nabla f(\bar{\mathbf{x}}) = 0$ and $\nabla^2 f(\bar{\mathbf{x}})$ is positive semidefinite

Theorem (Second order sufficient condition)

Suppose $\nabla^2 f$ continuous in an open Neighborhood $\mathcal{N}(\bar{\mathbf{x}})$ and that $\nabla f(\bar{\mathbf{x}})$ and $\nabla^2 f(\bar{\mathbf{x}})$ is positive definite. Then $\bar{\mathbf{x}}$ is a strict local minimizer of f.

Theorem (Convexity)

If f is convex, any local minimizer $\bar{\mathbf{x}}$ is also a global minimizer for f.
If f is convex and differentiable, any stationary point $\bar{\mathbf{x}}$ is also a global minimizer of f.

Examples of convex/concave functions

On $\mathbb{R}$:

$$ax + b,\ e^{ax},\ |x|^{\alpha},\ x\log x,\quad \log x$$

On $\mathbb{R}^n$

$$\mathbf{a}^T\mathbf{x} + \mathbf{b},\ \|\mathbf{x}\|_p$$

Examples of operations that presenve convexity:

- nonnegative weighted sum: $\sum \alpha_i f_i$ is convex if f_i are all convex and $\alpha_i > 0$
- composition: $f(A\mathbf{x} + \mathbf{b})$ is convex if f is convex
- ...

The study of convexity can greatly improve optimization performance!

Optimization strategies overview

Line Search vs Trust Region

Line Search:

- Choose a direction $\mathbf{p_k}$ and look for a point with lower value:

$$\mathbf{x_{k+1}} = \min_{\alpha>0} f(\mathbf{x}_k + \alpha \mathbf{p}_k)$$

- exact solution for $\mathbf{p}$ usually too expensive/impossible
- use a loose approximation of $\mathbf{p}$ after a few trials and try again

Trust Region:

- build a model function m_k to approximate/simplify f over trust-region $\mathcal{N}_k = \mathcal{N}(\mathbf{x}_k)$
- solve (approximate):

$$\mathbf{x_{k+1}} = \mathbf{x}_k + \mathbf{p} \text{ where } min_{\mathbf{p}} m_k(\mathbf{x_k} + \mathbf{p})$$

- if f does not decrease enough ($f(\mathbf{x_{k+1}}) - f(\mathbf{x_k}) > -\delta$) shrink the trust region and try again
- usually the trust region is a sphere of given radius
- usually the model m is a polynomial approximation of low (2, 3) degree

Derivative-free optimization

And what if the f is not differentiable? Or has stochastic components, noise etc.?

Nelder-Mead simplex-reflection method:

- Define a simplex trust region where to search for a minimum
- At each iteration, replace the vertex with the worst f value with a better one
- If impossible, replace all vertices but the best ones with some closer to the best one
- Stopping conditions on simplex size

Genetic algorithms:

- Completely stochastic approach
- Consider each dimension of $\mathbf{R}^n$ as a "gene"
- Generate a population of vectors in the search space
- Combine vectors of the population, choosing the best individuals in each generation
- Stopping condition on lack of improvement after some generations

Gradient Descent

$$\mathbf{x}_{k+1} = \mathbf{x_n} - \eta \nabla f(\mathbf{x}_k) \tag{2}$$

- Use derivative (gradient) to move in the correct direction
- Only works on differentiable and convex functions!

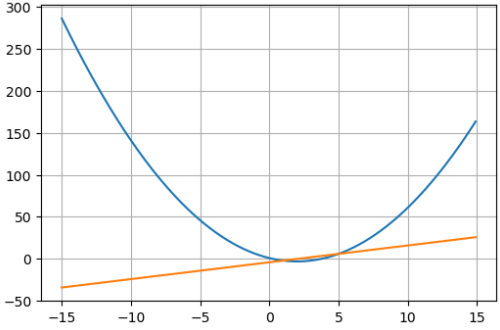
Steps:

1. Check for differentiability and convexity
2. Iterate (2) until:
   - maximum iterations reached
   - tolx (and/or tolf) reached

```python
import numpy as np
from matplotlib import pyplot as plt
plt.rcParams['axes.grid'] = True
```

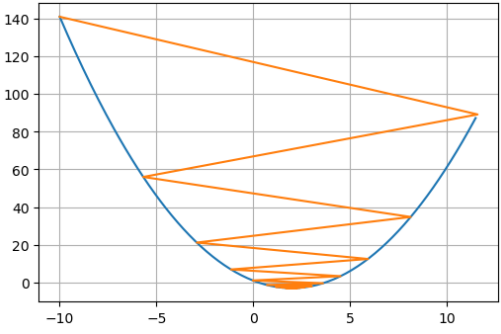
```python
def gradient_descent(f, f_grad, x_0, maxiter=1000, eta=0.9, tolx=10**-10):
    counter = 0
    x_history = [x_0]

    while counter < maxiter:
        counter += 1
        x = x_history[-1]
        new_x = x - eta * f_grad(x)
        x_history.append(new_x)
        if abs(np.min(new_x-x)) < tolx:
            break

    return new_x, np.array(x_history)
```

```
1  def f(x):
2      return x**2 - 4*x + 1
3
4  def f_grad(x):
5      return 2*x - 4
6
7  x_base = np.arange(-15, 15, 0.1)
8  plt.plot(x_base, f(x_base));
9  plt.plot(x_base, f_grad(x_base));
```



```
x_min, x_history = gradient_descent(f, f_grad, -10, eta=0.9)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.1)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
1.9999999999559648
119
```



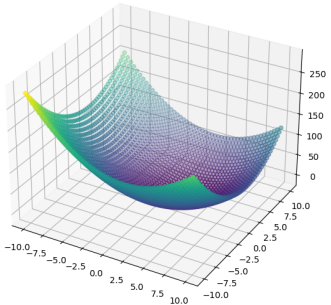
```python
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
print(X.shape)
print(X.ravel().shape)
x_base = np.vstack([X.ravel(), Y.ravel()]).T
print(x_base.shape)
```

```
(50, 50)
(2500,)
(2500, 2)
```

```python
# m points (rows) times n dimensions (columns), same polynomial on all dimensions
def f(x):
    return np.sum(x**2 - 4*x + 1, axis=1)

# gradient is a column vector! A row per dimension, a column per point
def f_grad(x):
    return (2*x - 4).T

```

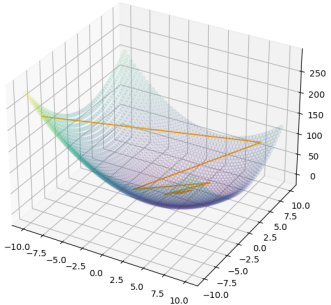
```
Z = f(x_base)
print(Z.shape)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```

```
(2500,)
```

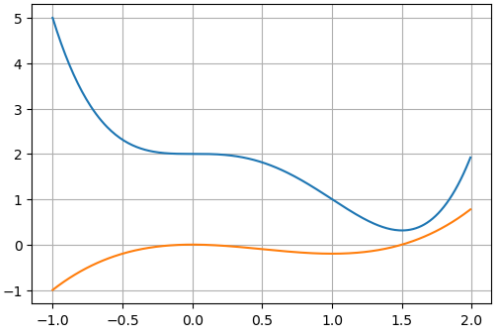


```
x_min, x_history = gradient_descent(f, f_grad, np.array([-8,-10]) , eta=0.8)
print(x_min)
print(len(x_history))

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history), color='orange', alpha=1)
```

```
[2. 2.]
53
```

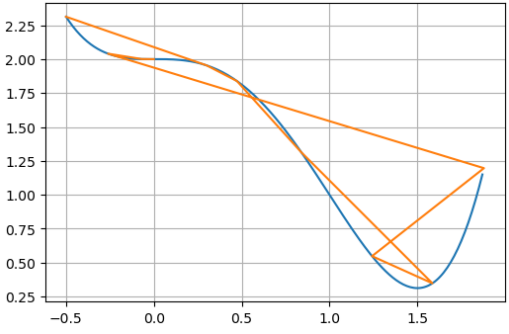


```
1  def f(x):
2      return x**4 - 2*x**3 + 2
3
4  def f_grad(x):
5      return 4*x**3 - 6*x**2
6
7  x_base = np.arange(-1, 2, 0.01)
8  plt.plot(x_base, f(x_base));
9  plt.plot(x_base, 0.1*f_grad(x_base));
```



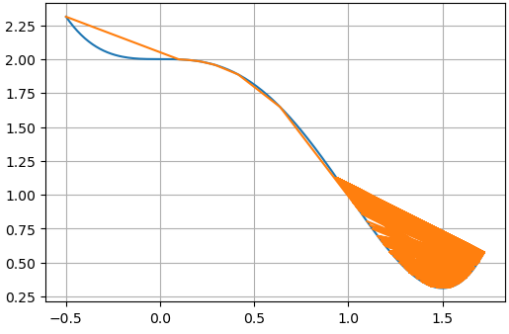
```
x_min, x_history = gradient_descent(f, f_grad, -0.5, eta=0.4)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.01)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
-0.00041472877781818996
1001
```



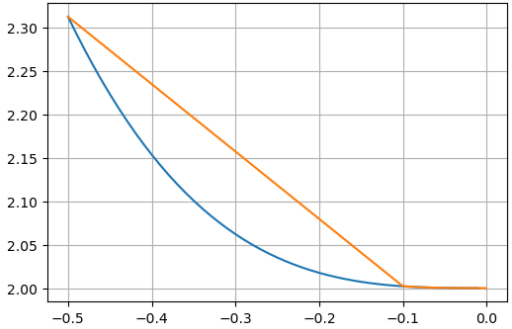
```
x_min, x_history = gradient_descent(f, f_grad, -0.5, eta=0.3)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.01)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
1.7202062388426052
1001
```



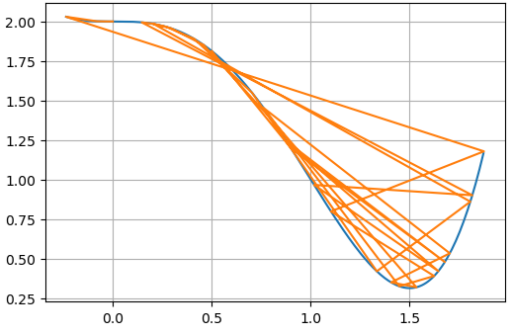
```
x_min, x_history = gradient_descent(f, f_grad, -0.5, eta=0.2)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.01)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
-0.0008211225683724356
1001
```



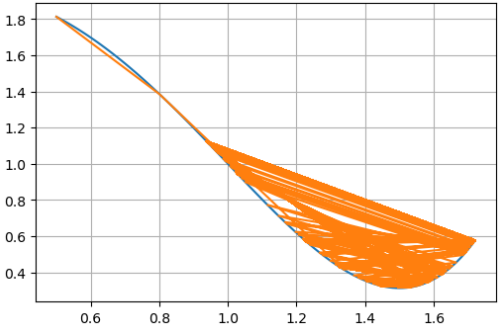
```
x_min, x_history = gradient_descent(f, f_grad, 0.5, eta=0.4)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.01)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
-0.0004230136377711251
1001
```



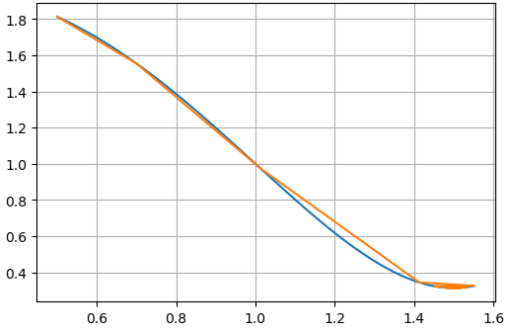
```
x_min, x_history = gradient_descent(f, f_grad, 0.5, eta=0.3)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.01)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
0.9382490585492951
1001
```



```
x_min, x_history = gradient_descent(f, f_grad, 0.5, eta=0.2)
print(x_min)
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.01)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
1.5000000000412663
99
```



Newton's method

$$\mathbf{x}_{k+1} = \mathbf{x_n} - \nabla^2 f(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k) \tag{3}$$

- Also only works on differentiable and convex functions!
- Additionally uses curvature information
- For convex function can converge to the global optimum with quadratic rate
- hessian (especially the inverse!) very expensive to compute

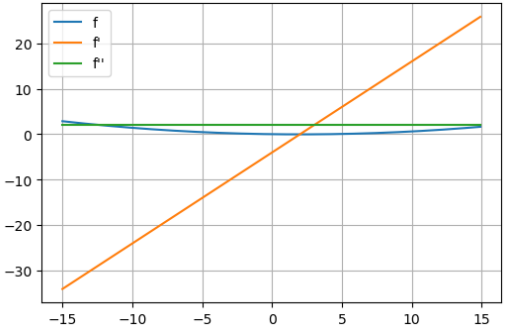
Steps:

1. Check for double differentiability and convexity
2. Iterate (3) until:
   - maximum iterations reached
   - tolx (and/or tolf) reached

```
1  import numpy as np
2  from scipy import optimize
3  from matplotlib import pyplot as plt
4  plt.rcParams['axes.grid'] = True
```

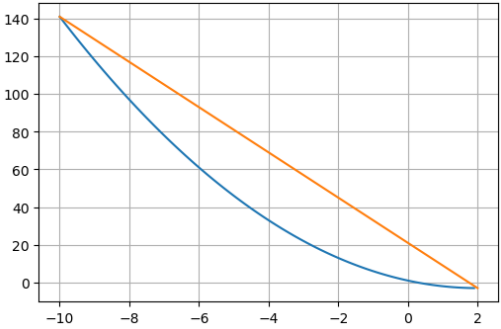
```python
# 1-dimensional example
def newton(f, f_grad, f_hessian, x_0, maxiter=1000, eta=0.9, tolx=10**-10):
    counter = 0
    x_history = [x_0]

    while counter < maxiter:
        counter += 1
        x = x_history[-1]
        new_x = x - f_grad(x)/f_hessian(x)
        x_history.append(new_x)
        if abs(np.min(new_x-x)) < tolx:
            break

    return new_x, np.array(x_history)
```

```python
def f(x):
    return x**2 - 4*x + 1

def f_grad(x):
    return 2*x - 4

def f_hessian(x):
    return 2 + 0*x

x_base = np.arange(-15, 15, 0.1)
plt.plot(x_base, 0.01*f(x_base), label='f');
plt.plot(x_base, f_grad(x_base), label='f\'');
plt.plot(x_base, f_hessian(x_base), label='f\'\'');
plt.legend();
```

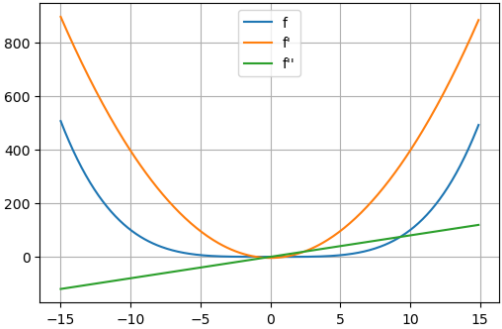

f
f'
f''
20
10
0
−10
−20
−30
−15
−10
−5
0
5
10
15

```
x_min, x_history = newton(f, f_grad, f_hessian, -10)
print(x_min, f(x_min))
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.1)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

```
2.0 -3.0
3
```

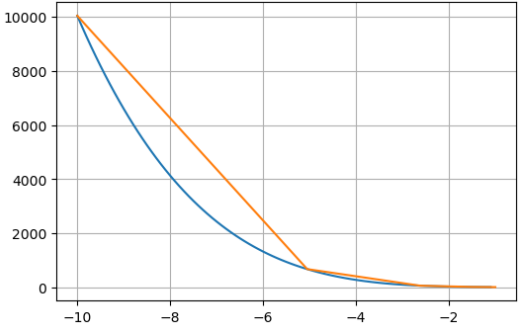


```python
def f(x):
    return x**4 - 4*x + 1

def f_grad(x):
    return 4*x**2 - 4

def f_hessian(x):
    return 8*x

x_base = np.arange(-15, 15, 0.1)
plt.plot(x_base, 0.01*f(x_base), label='f');
plt.plot(x_base, f_grad(x_base), label='f\'');
plt.plot(x_base, f_hessian(x_base), label='f\'\'');
plt.legend();
```


f
f'
f''
800
600
400
200
0
−15
−10
−5
0
5
10
15

```python
x_min, x_history = newton(f, f_grad, f_hessian, -10)
print(x_min, f(x_min))
print(len(x_history))
x_base = np.arange(min(x_history), max(x_history), 0.1)
plt.plot(x_base, f(x_base));
plt.plot(x_history, f(np.array(x_history)));
```

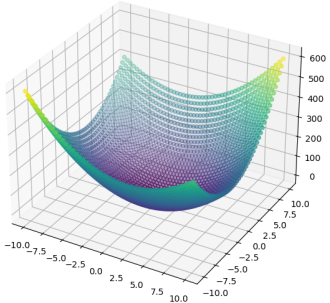
```
-1.0 6.0
9
```



```python
# vectorial implementation
def newton(f, f_grad, f_hessian, x_0, maxiter=1000, eta=0.9, tolx=10**-10):
    counter = 0
    x_history = [x_0]

    while counter < maxiter:
        counter += 1
        x = x_history[-1]
        new_x = x - np.linalg.inv(f_hessian(x)).dot(f_grad(x))
        x_history.append(new_x)
        if abs(np.min(new_x-x)) < tolx:
            break

    return new_x, np.array(x_history)
```

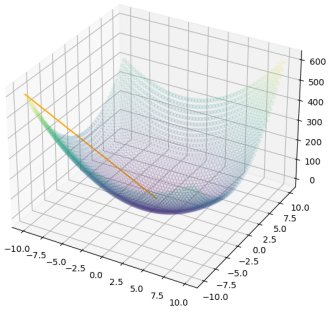
```python
# m points (rows) times n dimensions (columns)
# f = 2 x^2 + xy + 3y^2
def f(x: np.array):
    assert x.shape[1] == 2
    return 2*x[:,0]**2 + x[:,0]*x[:,1] + 3*x[:,1]**2

# gradient is a column vector! A row per dimension, a column per point
def f_grad(x: np.array):
    assert x.shape == (2,)
    return np.array([ 4*x[0] + x[1] , 6*x[1] + x[0] ]).T


# not-vectorized implementation for simplicity
# assumes x is a vector size n (dimension)
def f_hessian(x):
    assert x.shape == (2,)
    gt = f_grad(x).T
    h = np.array( [[  4 ,  1],
                   [ 1 ,  6]] )
    return h
```

```python
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
x_base = np.vstack([X.ravel(), Y.ravel()]).T

Z = f(x_base)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



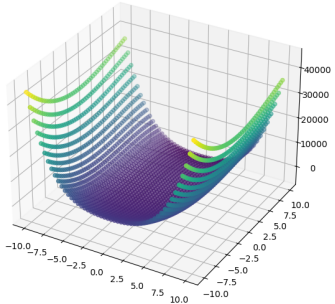
```
x_min, x_history = newton(f, f_grad, f_hessian, np.array([-10,-10]))
print(x_min, f(np.array([x_min]).reshape((1,2))))
print(len(x_history))
print(x_history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history), color='orange', alpha=1)
```

```
[0. 0.] [0.]
3
[[-1.00000000e+01 -1.00000000e+01]
 [ 0.00000000e+00 -1.77635684e-15]
 [ 0.00000000e+00  0.00000000e+00]]
```


600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

```python
# f = 3x^4 - 4y**3 + x**2y**2
def f(x: np.array):
    assert x.shape[1] == 2  # m points (rows) times n dimensions (columns)
    return 3*x[:,0]**4 - 4*x[:,1]**3 + x[:,0]**2 * x[:,1]**2

# gradient is a column vector! A row per dimension, a column per point
def f_grad(x: np.array):
    assert x.shape == (2,) # one point at a time!
    return np.array([ 12*x[0]**3 + 2*x[0]*x[1]**2 , -12*x[1]**2 + 2*x[0]**2*x[1] ]).T


# not-vectorized implementation for simplicity
# assumes x is a vector size n (dimension)
def f_hessian(x):
    assert x.shape == (2,)  # one point at a time!
    h = np.array( [[  36*x[0]**2 + 2*x[1]**2 , 4*x[0]*x[1] ],
                   [ 4*x[1]*x[0] , -24*x[1] + 2*x[0]**2]] )
    return h
```

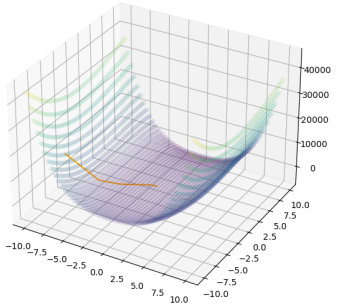
```python
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
x_base = np.vstack([X.ravel(), Y.ravel()]).T

Z = f(x_base)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



```python
x_min, x_history = newton(f, f_grad, f_hessian, np.array([-8,-6]), tolx=10**-12)
print(x_min, f(np.array([x_min]).reshape((1,2))))
print(len(x_history))
print(x_history[0::5])
```

```
[-9.84218546e-08 -5.29568876e-13] [2.81505855e-28]
46
[[-8.00000000e+00 -6.00000000e+00]
 [-1.08231914e+00 -3.23864028e-01]
 [-1.43234065e-01 -1.34383219e-02]
 [-1.88717096e-02 -4.84817559e-04]
 [-2.48526003e-03 -1.63310003e-05]
 [-3.27277880e-04 -5.31284851e-07]
 [-4.30983284e-05 -1.69700479e-08]
 [-5.67550009e-06 -5.36724698e-10]
 [-7.47390958e-07 -1.68841912e-11]
 [-9.84218546e-08 -5.29568876e-13]]
```

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history), color='orange', alpha=1)
```



```
x_min, x_history = newton(f, f_grad, f_hessian, np.array([-8,-6]), tolx=10**-32)
print(x_min, f(np.array([x_min]).reshape((1,2))))
print(len(x_history))
print(x_history[0::5])
```

```
[-2.35000316e-19 -7.20977671e-33] [9.14945112e-75]
112
[[-8.00000000e+00 -6.00000000e+00]
 [-1.08231914e+00 -3.23864028e-01]
 [-1.43234065e-01 -1.34383219e-02]
 [-1.88717096e-02 -4.84817559e-04]
 [-2.48526003e-03 -1.63310003e-05]
 [-3.27277880e-04 -5.31284851e-07]
 [-4.30983284e-05 -1.69700479e-08]
 [-5.67550009e-06 -5.36724698e-10]
 [-7.47390958e-07 -1.68841912e-11]
 [-9.84218546e-08 -5.29568876e-13]
 [-1.29609027e-08 -1.65826676e-14]
 [-1.70678554e-09 -5.18792059e-16]
 [-2.24761881e-10 -1.62223772e-17]
 [-2.95982724e-11 -5.07124907e-19]
 [-3.89771489e-12 -1.58506991e-20]
 [-5.13279327e-13 -4.95387170e-22]
 [-6.75923393e-14 -1.54817651e-23]
 [-8.90104880e-15 -4.83821044e-25]
 [-1.17215457e-15 -1.51196831e-26]
 [-1.54357804e-16 -4.72494875e-28]
 [-2.03269536e-17 -1.47655477e-29]
 [-2.67680048e-18 -4.61424802e-31]
 [-3.52500474e-19 -1.44195500e-32]]
```

```python
# This is as proof-test for the next section!
def f(x: np.array):
    assert x.shape == (2,)  # one point at a time!
    return 3*x[0]**4 - 4*x[1]**3 + x[0]**2 * x[1]**2

x_min, x_history = newton(f, f_grad, f_hessian, np.array([-8,-6]), tolx=10**-12)
print(x_min, f(x_min))
print(len(x_history))
print(x_history[0::5])
```

```
[-9.84218546e-08 -5.29568876e-13] 2.815058553998554e-28
46
[[-8.00000000e+00 -6.00000000e+00]
 [-1.08231914e+00 -3.23864028e-01]
 [-1.43234065e-01 -1.34383219e-02]
 [-1.88717096e-02 -4.84817559e-04]
 [-2.48526003e-03 -1.63310003e-05]
 [-3.27277880e-04 -5.31284851e-07]
 [-4.30983284e-05 -1.69700479e-08]
 [-5.67550009e-06 -5.36724698e-10]
 [-7.47390958e-07 -1.68841912e-11]
 [-9.84218546e-08 -5.29568876e-13]]
```

Newton Method with approximate derivatives

What if f is unknown? If not noisy/stochastic:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \quad f''(x) \approx \frac{f'(x+h) - f'(x)}{h} = \frac{f(x+2h) + f(x)}{h^2}$$

$$\nabla f(\mathbf{x}) = \frac{1}{h} \begin{pmatrix} f(\mathbf{x} + h\mathbf{e}_1) - f(\mathbf{x}) \\ \vdots \\ f(\mathbf{x} + h\mathbf{e}_n) - f(\mathbf{x}) \end{pmatrix}, \quad \nabla^2 f(x) = \frac{1}{h^2} \left(f(\mathbf{x} + h\mathbf{e_i} + h\mathbf{e_j}) - f(\mathbf{x} + h\mathbf{e_j}) - f(\mathbf{x} + h\mathbf{e_i}) + f(\mathbf{x}) \right)_{ij}$$

- That's *a lot* of function evaluations at each step! ($2 \times n$ for ∇, $4 \times n^2$ for ∇^2 minus change)
- How to choose h? Watch for numerical issues!
- What if $\nabla^2 f(\mathbf{x})$ is singular or not defined?

Gradient descent with approximation (very simplified BFGS)

Idea is still:

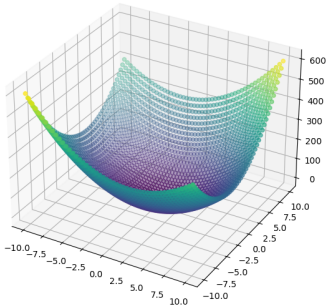
$$\mathbf{x}_{k+1} = \mathbf{x_n} - \eta \nabla f(\mathbf{x}_k)$$

But:

- the gradient ∇f is an approximation
- the step η is computed instead of fixed
- the search for η is done on a polynomial approximation of f

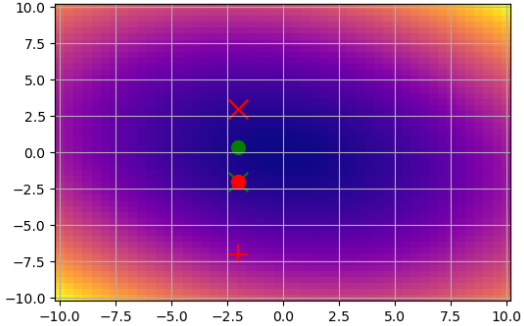
```
1  import numpy as np
2  from scipy import optimize
3  from matplotlib import pyplot as plt
4  plt.rcParams['axes.grid'] = True
```

```python
def f(x):
    return 2*x[0]**2 + x[0]*x[1] + 3*x[1]**2

x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
x_base = np.vstack([X.ravel(), Y.ravel()])

Z = f(x_base)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```

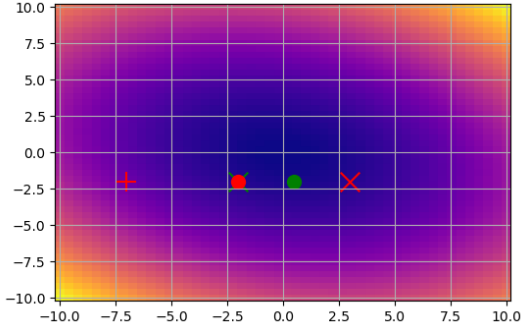

600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

```python
def find_eta(f, p, x, min_eta=-1, max_eta=1):
    etas = np.array([min_eta, (max_eta+min_eta)/2 , max_eta])
    x_0 = x + etas[0]*p
    x_1 = x + etas[1]*p
    x_2 = x + etas[2]*p

    X = np.vstack([x_0, x_1, x_2]).T
    Y = f(X)

    coeffs = np.polyfit(etas, Y, 2)  # ax^n +bx_n-1 +...
    #np.polyval(coeffs, x)
    # p = ax^2+bx+c
    # p' = 2ax + b
    # p'' = 2a
    if coeffs[0] > 0: #convex parabola
        eta = -coeffs[1]/(2*coeffs[0]) #-b/2a
        return max(min_eta, min(max_eta, eta))
    else: #concave parabola
        if Y[0] < Y[2]:
            return min_eta
        else:
            return max_eta
```

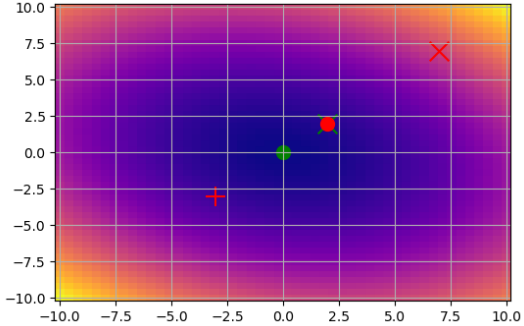
```
x0 = np.array([-2,-2])
p = np.array([0,1])
min_eta, max_eta = -5, 5
eta = find_eta(f, p, x0, min_eta=min_eta, max_eta=max_eta)
x = x0 + eta*p

z = Z.reshape((50,50))
plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto');
plt.scatter(*x0, marker='x', color='green', s=200);
plt.scatter(*(x0+min_eta*p), marker='+', color='red', s=200);
plt.scatter(*(x0+max_eta*p), marker='x', color='red', s=200);
plt.scatter(*(x0+((max_eta+min_eta)/2)*p), marker='o', color='red', s=100);
plt.scatter(*x, marker='o', color='green', s=100);
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

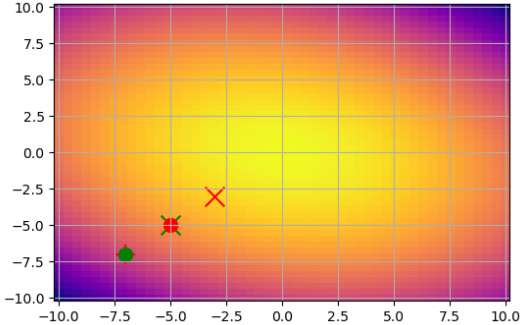
```
x0 = np.array([-2,-2])
p = np.array([1,0])
min_eta, max_eta = -5, 5
eta = find_eta(f, p, x0, min_eta=min_eta, max_eta=max_eta)
x = x0 + eta*p

z = Z.reshape((50,50))
plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto');
plt.scatter(*x0, marker='x', color='green', s=200);
plt.scatter(*(x0+min_eta*p), marker='+', color='red', s=200);
plt.scatter(*(x0+max_eta*p), marker='x', color='red', s=200);
plt.scatter(*(x0+((max_eta+min_eta)/2)*p), marker='o', color='red', s=100);
plt.scatter(*x, marker='o', color='green', s=100);
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

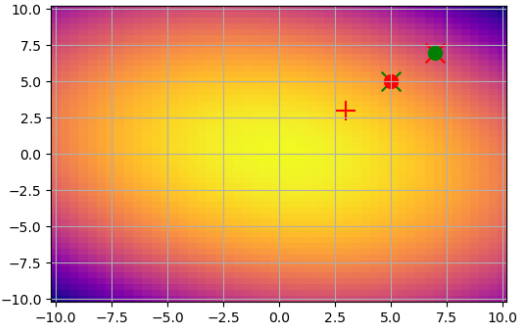
```
x0 = np.array([2,2])
p = np.array([1,1])
min_eta, max_eta = -5,5
eta = find_eta(f, p, x0, min_eta=min_eta, max_eta=max_eta)
x = x0 + eta*p

z = Z.reshape((50,50))
plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto');
plt.scatter(*x0, marker='x', color='green', s=200);
plt.scatter(*(x0+min_eta*p), marker='+', color='red', s=200);
plt.scatter(*(x0+max_eta*p), marker='x', color='red', s=200);
plt.scatter(*(x0+((max_eta+min_eta)/2)*p), marker='o', color='red', s=100);
plt.scatter(*x, marker='o', color='green', s=100);
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

```
g = lambda x: -f(x)
x0 = np.array([-5,-5])
p = np.array([1,1])
min_eta, max_eta = -2,2
eta = find_eta(g, p, x0, min_eta=min_eta, max_eta=max_eta)
x = x0 + eta*p

plt.pcolormesh(X, Y, -z, cmap='plasma', shading='auto');
plt.scatter(*x0, marker='x', color='green', s=200);
plt.scatter(*(x0+min_eta*p), marker='+', color='red', s=200);
plt.scatter(*(x0+max_eta*p), marker='x', color='red', s=200);
plt.scatter(*(x0+((max_eta+min_eta)/2)*p), marker='o', color='red', s=100);
plt.scatter(*x, marker='o', color='green', s=100);
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

```
x0 = np.array([5,5])
eta = find_eta(g, p, x0, min_eta=min_eta, max_eta=max_eta)
x = x0 + eta*p

plt.pcolormesh(X, Y, -z, cmap='plasma', shading='auto');
plt.scatter(*x0, marker='x', color='green', s=200);
plt.scatter(*(x0+min_eta*p), marker='+', color='red', s=200);
plt.scatter(*(x0+max_eta*p), marker='x', color='red', s=200);
plt.scatter(*(x0+((max_eta+min_eta)/2)*p), marker='o', color='red', s=100);
plt.scatter(*x, marker='o', color='green', s=100);
```



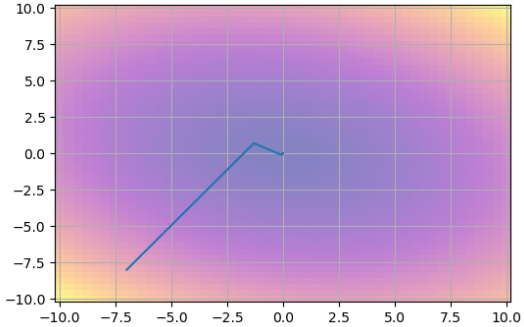
```python
def f_grad_approx(f, x_0, h=1e-1):
    n = len(x_0)
    grad = np.zeros(n)

    for i in range(n):
        h_i = np.eye(1,n,i).flatten() * h
        xl, xr = x_0-h_i, x_0+h_i
        grad[i] = (f(xr)-f(xl))/(2*h)

    return grad
```

```python
def simplified_BFGS(f, x_0, maxiter=1000, tolx=1e-10, eta=1):
    counter = 0
    x_history = [x_0]
    f_grad = f_grad_approx(f, x_0)

    while counter < maxiter:
        counter += 1
        x = x_history[-1]
        p = f_grad_approx(f, x)
        neta = find_eta(f, p, x, -eta, eta)
        new_x = x + neta*p
        x_history.append(new_x)
        if abs(np.min(new_x-x)) < tolx:
            break

    return new_x, np.array(x_history)
```

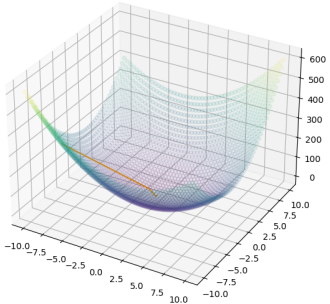
```
x_0 = np.array([-7, -8])
sol, x_history = simplified_BFGS(f, x_0)
print (sol,  len(x_history))
```

```
[-2.94289276e-12  1.60009673e-12] 14
```

```
1  plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
2  plt.plot(x_history[:,0], x_history[:,1]);
```

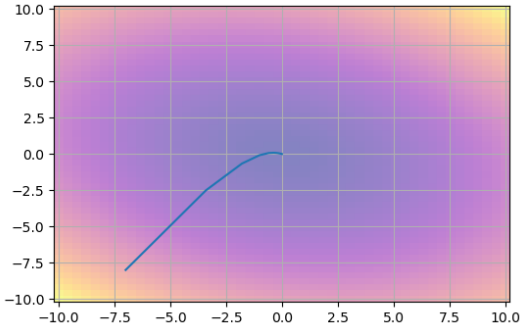


```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

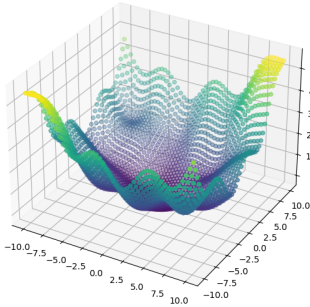


```
x_0 = np.array([-7, -8])
sol, x_history = simplified_BFGS(f, x_0, eta=0.1)
print (sol,  len(x_history))
plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

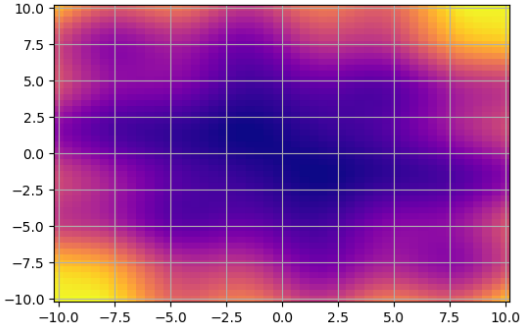
```
[-2.94630697e-10  1.22040026e-10] 53
```



```python
def f(x):
    return 2*x[0]**2 + x[0]*x[1] + 3*x[1]**2 + 5*x[0]*np.sin(x[1]) + 5*x[1]*np.sin(x[0])

x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
x_base = np.vstack([X.ravel(), Y.ravel()])

Z = f(x_base)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```

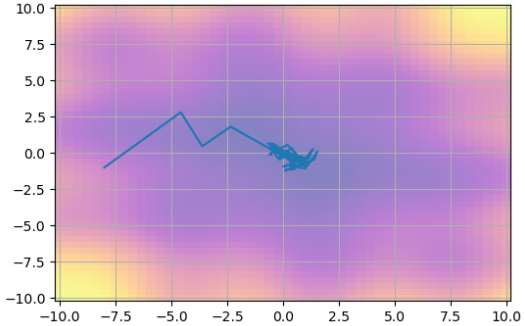

500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

```
1  z = Z.reshape((50,50))
2  plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto');
```

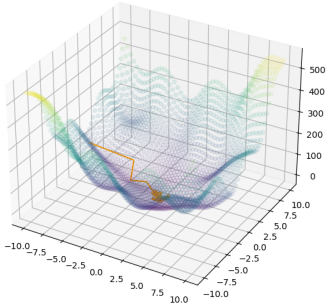


```
x_0 = np.array([-8, -1])
sol, x_history = simplified_BFGS(f, x_0, eta=1)
print (sol, len(x_history))

plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

```
[-0.53012055 -0.10230653] 1001
```

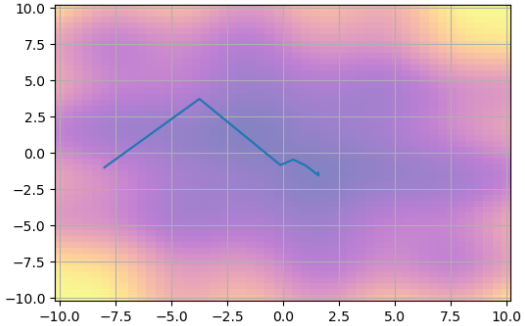


```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

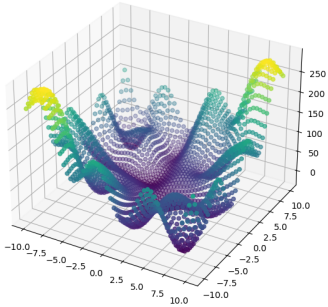


```
x_0 = np.array([-8, -1])
sol, x_history = simplified_BFGS(f, x_0, eta=0.2)
print (sol, len(x_history))

plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

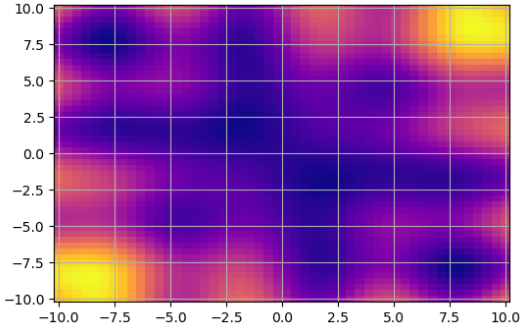
```
[ 1.56843658 -1.3635669 ] 18
```



```python
def f(x):
    return x[0]**2 + x[0]*x[1] + x[1]**2 + 5*x[0]*np.sin(x[1]) + 5*x[1]*np.sin(x[0])

x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
x_base = np.vstack([X.ravel(), Y.ravel()])

Z = f(x_base)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```

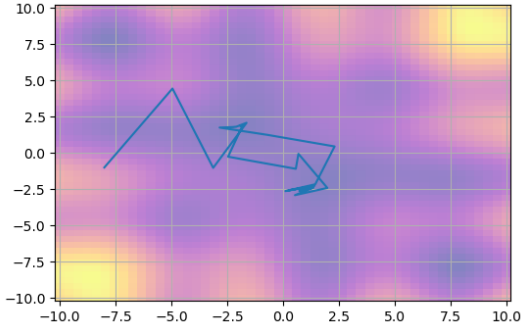

250
200
150
100
50
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

```
1  z = Z.reshape((50,50))
2  plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto');
```

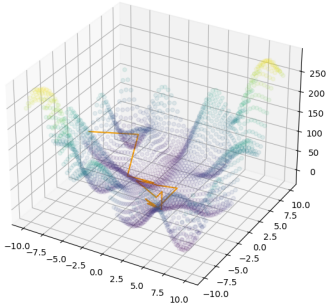


```
x_0 = np.array([-8, -1])
sol, x_history = simplified_BFGS(f, x_0, eta=1)
print (sol, len(x_history))

plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

```
[-1.88211303  1.88159848] 37
```

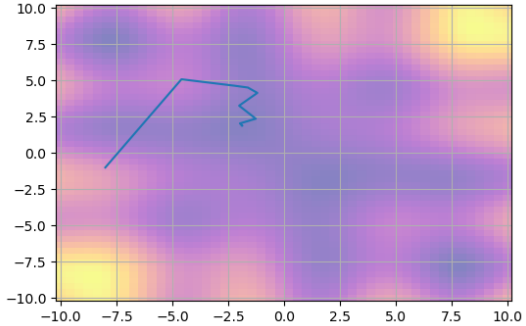


```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```



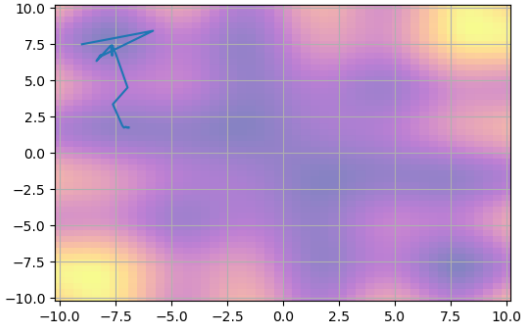
```
x_0 = np.array([-8, -1])
sol, x_history = simplified_BFGS(f, x_0, eta=0.2)
print (sol, len(x_history))

plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

```
[-1.88162442  1.88186874] 16
```



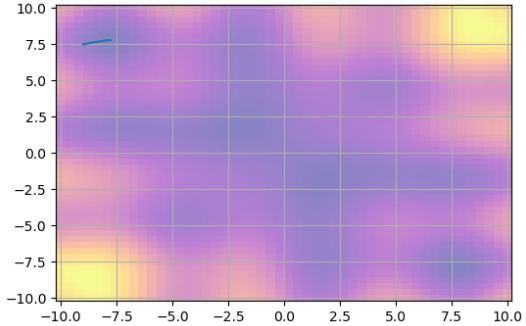
```
x_0 = np.array([-9, 7.5])
sol, x_history = simplified_BFGS(f, x_0, eta=0.2)
print (sol, len(x_history))

plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

```
[-6.90361457  1.75472133] 35
```



```
x_0 = np.array([-9, 7.5])
sol, x_history = simplified_BFGS(f, x_0, eta=0.1)
print (sol, len(x_history))

plt.pcolormesh(X, Y, z, cmap='plasma', shading='auto', alpha=0.5);
plt.plot(x_history[:,0], x_history[:,1]);
```

```
[-7.78210924  7.78203928] 10
```



Check existing implementations!

BFGS: quasi-Newton method of Broyden, Fletcher, Goldfarb, and Shanno

- Only uses first derivative if available, otherwise approximates also hessian

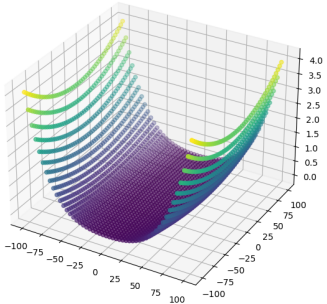
```
scipy.optimize.minimize(method='BFGS')
```

Newotn-CG, trust-constr (previous slide)...

```
1  import numpy as np
2  from scipy import optimize
3  from matplotlib import pyplot as plt
4  plt.rcParams['axes.grid'] = True
```

```
1  def f(x: np.array):
2      return 3*x[0]**4 - 4*x[1]**3 + x[0]**2 * x[1]**2
```

```python
x = np.linspace(-100, 100, 50)
y = np.linspace(-100, 100, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()

Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



```
test = f(np.vstack([X,Y]))
all(test == Z)
```

```
True
```

```python
def callback_generator():
    history = list()
    def cb(*args, **kwargs):
        history.append([args, kwargs])
    return cb, history
```

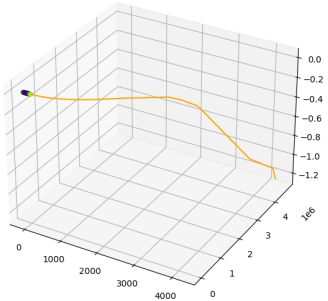
```python
cb, history = callback_generator()
optimize.minimize(f, np.array([-8, -6]), method='BFGS', tol=10**-10, callback=cb)
```

```
  message: Desired error not necessarily achieved due to precision loss.
  success: False
   status: 2
      fun: -1.2312628309191098e+19
        x: [ 4.264e+03  4.686e+06]
      nit: 90
      jac: [ 1.873e+17 -9.236e+13]
hess_inv: [[ 4.365e-15  1.030e-11]
           [ 1.030e-11  2.660e-08]]
     nfev: 479
     njev: 157
```

```python
print(history[0])

def read_history(h):
    return np.vstack([x[0][0] for x in h])


x_history = read_history(history)
print(x_history.shape)
print(x_history[::5])
```

```
[(array([-7.00572815, -5.82245146]),), {}]
(90, 2)
[[-7.00572815e+00 -5.82245146e+00]
 [-3.83689016e+00 -1.42354575e+00]
 [-9.66002658e-01 -5.24334772e-02]
 [ 4.98246331e-01  6.00531775e-01]
 [ 3.78208333e+00  7.20881908e+00]
 [ 1.22295746e+01  4.69636169e+01]
 [ 2.17992812e+01  1.33926259e+02]
 [ 3.55087499e+01  3.38853261e+02]
 [ 5.68003645e+01  8.48109193e+02]
 [ 8.97359251e+01  2.09705773e+03]
 [ 1.40918828e+02  5.15101363e+03]
 [ 2.20566369e+02  1.25988544e+04]
 [ 3.44931966e+02  3.07884602e+04]
 [ 5.37380333e+02  7.47560833e+04]
 [ 8.44288255e+02  1.84304594e+05]
 [ 1.30353260e+03  4.40090200e+05]
 [ 2.01695604e+03  1.04711598e+06]
 [ 3.40349872e+03  3.02328899e+06]]
```

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

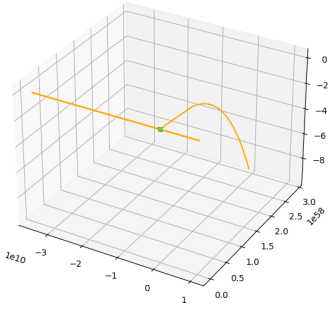


```
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='trust-constr', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

```
/usr/lib/python3/dist-packages/scipy/optimize/_trustregion_constr/qp_subproblem.py:114: RuntimeWarning: overflow encountered in scalar multiply
  discriminant = b*b - 4*a*c
```

```
message: The maximum number of function evaluations is exceeded.
          success: False
           status: 0
              fun: -9.518247911038625e+175
                x: [ 1.054e+06  2.876e+58]
              nit: 1000
             nfev: 2256
             njev: 752
             nhev: 0
         cg_niter: 1390
     cg_stop_cond: 2
             grad: [ 0.000e+00 -9.928e+117]
  lagrangian_grad: [ 0.000e+00 -9.928e+117]
           constr: []
              jac: []
      constr_nfev: []
      constr_njev: []
```

```
      constr_nhev: []
                v: []
           method: equality_constrained_sqp
       optimality: 9.927637997582089e+117
  constr_violation: 0
    execution_time: 0.5644679069519043
        tr_radius: 9.882283524957568e+54
    constr_penalty: 1.0
             niter: 1000
```

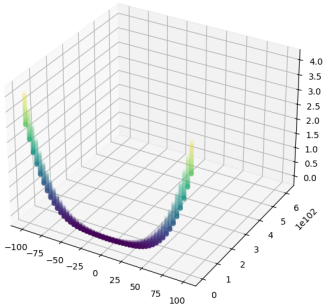


```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='powell', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

```
message: Optimization terminated successfully.
 success: True
  status: 0
     fun: -inf
       x: [ 5.791e-09  6.156e+102]
     nit: 1
   direc: [[ 1.000e+00  0.000e+00]
           [ 0.000e+00  1.000e+00]]
    nfev: 552
```

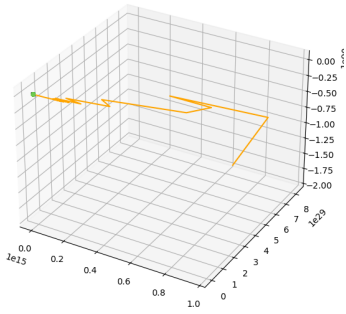
```
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2973: RuntimeWarning: overflow encountered in scalar multiply
  w = xb - ((xb - xc) * tmp2 - (xb - xa) * tmp1) / denom
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2972: RuntimeWarning: overflow encountered in scalar multiply
  denom = 2.0 * val
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2973: RuntimeWarning: invalid value encountered in scalar divide
  w = xb - ((xb - xc) * tmp2 - (xb - xa) * tmp1) / denom
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2966: RuntimeWarning: overflow encountered in scalar multiply
  tmp1 = (xb - xa) * (fb - fc)
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2967: RuntimeWarning: overflow encountered in scalar multiply
  tmp2 = (xb - xc) * (fb - fa)
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2968: RuntimeWarning: invalid value encountered in scalar subtract
  val = tmp2 - tmp1
/tmp/ipykernel_110575/2187872282.py:2: RuntimeWarning: overflow encountered in scalar multiply
  return 3*x[0]**4 - 4*x[1]**3 + x[0]**2 * x[1]**2
/tmp/ipykernel_110575/2187872282.py:2: RuntimeWarning: overflow encountered in scalar power
  return 3*x[0]**4 - 4*x[1]**3 + x[0]**2 * x[1]**2
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2472: RuntimeWarning: invalid value encountered in scalar subtract
  tmp1 = (x - w) * (fx - fv)
/usr/lib/python3/dist-packages/scipy/optimize/_optimize.py:2473: RuntimeWarning: invalid value encountered in scalar subtract
  tmp2 = (x - v) * (fx - fw)
/tmp/ipykernel_110575/2187872282.py:2: RuntimeWarning: overflow encountered in power
  return 3*x[0]**4 - 4*x[1]**3 + x[0]**2 * x[1]**2
```

```
/usr/lib/python3/dist-packages/mpl_toolkits/mplot3d/proj3d.py:177: RuntimeWarning: invalid value encountered in
↪  divide
  txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```



```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='nelder-mead', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

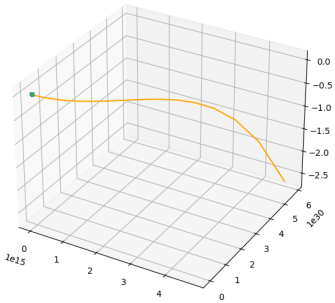
```
message: Maximum number of function evaluations has been exceeded.
       success: False
        status: 1
           fun: -1.9143830178334263e+90
             x: [ 6.344e+14  8.172e+29]
           nit: 209
          nfev: 400
 final_simplex: (array([[ 6.344e+14,  8.172e+29],
                        [ 9.600e+14,  6.085e+29],
                        [ 5.592e+14,  3.393e+29]]), array([-1.914e+90, -5.600e+89, -1.203e+89]))
```


0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
1e29
0
1
2
3
4
5
6
7
8
1e13
0.0
0.2
0.4
0.6
0.8
1.0

```python
def f_grad(x: np.array):
    return np.array([ 12*x[0]**3 + 2*x[0]*x[1]**2 , -12*x[1]**2 + 2*x[0]**2*x[1] ]).T

def f_hessian(x):
    h = np.array( [[  36*x[0]**2 + 2*x[1]**2 , 4*x[0]*x[1] ],
                   [ 4*x[1]*x[0] , -24*x[1] + 2*x[0]**2]] )
    return h
```

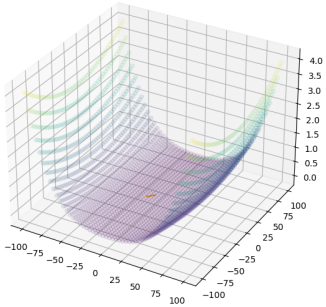
```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='BFGS', jac=f_grad, tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

```
message: Maximum number of iterations has been exceeded.
  success: False
   status: 1
      fun: -2.6246515245425888e+91
        x: [ 4.738e+15  5.807e+30]
      nit: 400
      jac: [ 3.196e+77 -1.439e+62]
 hess_inv: [[ 4.890e-62  1.208e-46]
            [ 1.208e-46  2.997e-31]]
     nfev: 430
     njev: 430
```

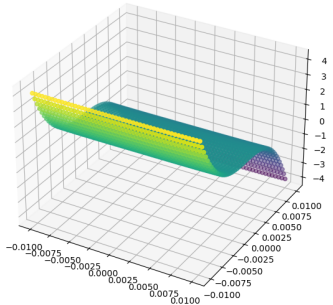

0.0
−0.5
−1.0
−1.5
−2.0
−2.5
1e20
6
5
4
3
2
1
0
1e15
0
1
2
3
4

```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='trust-exact', jac=f_grad, hess=f_hessian, tol=10**-10,
↪  callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(np.vstack([x,y])), color='orange', alpha=1)
```

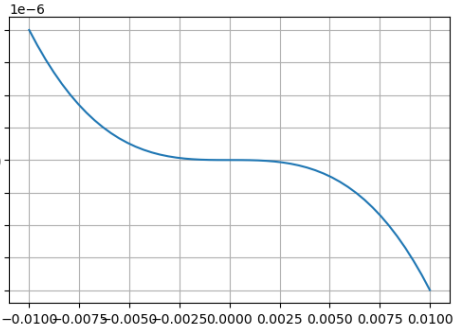
```
message: Optimization terminated successfully.
 success: True
  status: 0
     fun: 1.3086566627333613e-15
       x: [-1.445e-04 -1.771e-07]
     nit: 28
     jac: [-3.622e-11 -3.838e-13]
    nfev: 29
    njev: 29
    nhev: 29
    hess: [[ 7.519e-07  1.024e-10]
           [ 1.024e-10  4.292e-06]]
```


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−100
−75
−50
−25
0
25
50
75
100
−100
−75
−50
−25
0
25
50
75
100

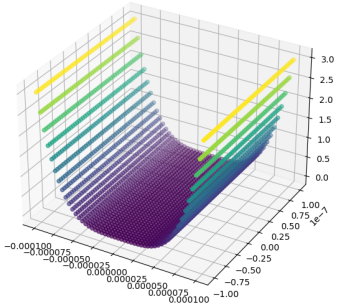
```
x = np.linspace(-0.01, 0.01, 50)
y = np.linspace(-0.01, 0.01, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()

Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



```
1  plt.plot(y, f([y*0,y]));
```

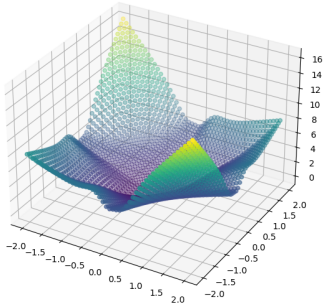


```python
x = np.linspace(-10**-4, 10**-4, 50)
y = np.linspace(-10**-7, 10**-7, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()

Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```

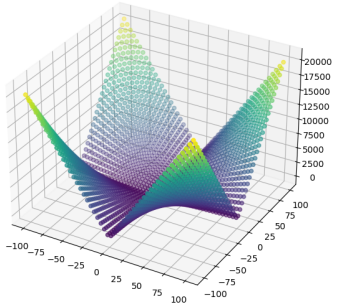


```python
def f(x: np.array):
    return (abs(x[0]*x[1]) + abs(x[0]-x[1])) * (2 + np.sin(x[0]*x[1])/(1+(x[0]*x[1])**4))
```

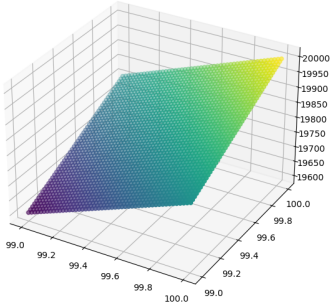
```python
x = np.linspace(-2, 2, 50)
y = np.linspace(-2, 2, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



```
x = np.linspace(-100, 100, 50)
y = np.linspace(-100, 100, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



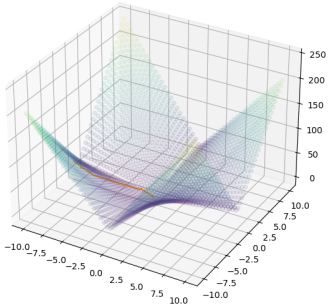
```
x = np.linspace(99, 100, 50)
y = np.linspace(99, 100, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



```
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]
```

```
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='BFGS', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

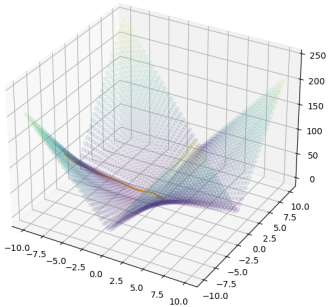
```
message: Desired error not necessarily achieved due to precision loss.
  success: False
   status: 2
      fun: 0.1298088831229093
        x: [-2.509e-01 -2.509e-01]
      nit: 6
      jac: [ 1.263e+00  1.530e+00]
 hess_inv: [[ 1.629e-01  6.334e-02]
            [ 6.334e-02  2.462e-02]]
     nfev: 192
     njev: 60
```


250
200
150
100
50
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='trust-constr', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

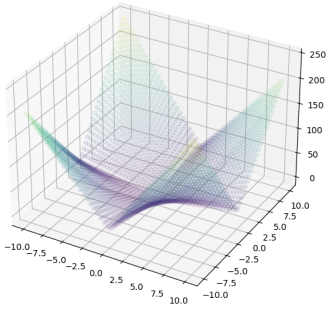
```
message: The maximum number of function evaluations is exceeded.
          success: False
           status: 0
              fun: 0.00606691171570428
                x: [ 5.504e-02  5.504e-02]
              nit: 1000
            nfev: 2991
            njev: 997
            nhev: 0
         cg_niter: 1011
     cg_stop_cond: 2
             grad: [ 2.113e+00  2.113e+00]
  lagrangian_grad: [ 2.113e+00  2.113e+00]
           constr: []
              jac: []
      constr_nfev: []
      constr_njev: []
      constr_nhev: []
                v: []
           method: equality_constrained_sqp
```

```
      optimality: 2.113432572747115
constr_violation: 0
  execution_time: 0.6807045936584473
       tr_radius: 2.0866498231384016e-08
  constr_penalty: 1.0
           niter: 1000
```



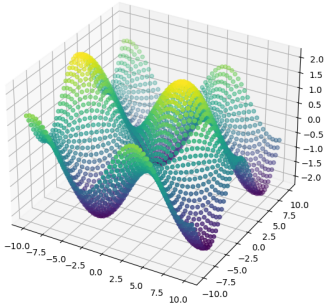
```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='powell', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

```
message: Optimization terminated successfully.
 success: True
  status: 0
     fun: 0.16697595553808425
       x: [-2.833e-01 -2.833e-01]
     nit: 3
   direc: [[ 0.000e+00  1.000e+00]
           [ 2.555e-12  0.000e+00]]
    nfev: 255
```

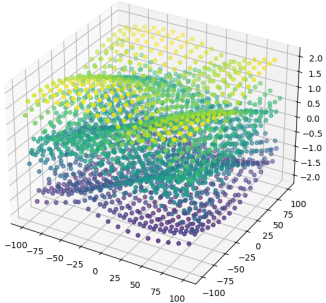

250
200
150
100
50
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

```
1  def f(x: np.array):
2      return np.sin(x[0]/2)+np.cos(x[1]/2)
```

```python
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



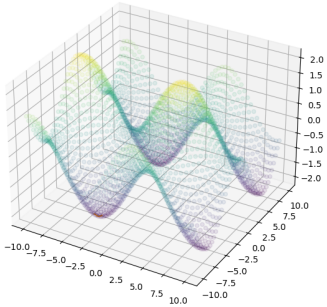
```
x = np.linspace(-100, 100, 50)
y = np.linspace(-100, 100, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```



```
x = np.linspace(-10, 10, 50)
y = np.linspace(-10, 10, 50)
X, Y = np.meshgrid(x, y)
X = X.ravel()
Y = Y.ravel()
Z = [f([X[i],Y[i]]) for i in range(len(X))]
```

```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='BFGS', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

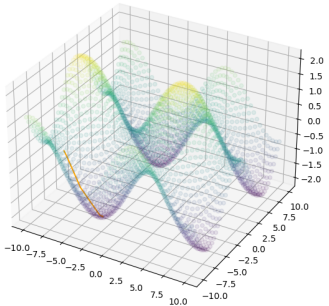
```
message: Optimization terminated successfully.
  success: True
   status: 0
      fun: -2.0
        x: [-3.142e+00 -6.283e+00]
      nit: 8
      jac: [ 0.000e+00  0.000e+00]
 hess_inv: [[ 4.182e+00  2.863e-01]
            [ 2.863e-01  3.716e+00]]
     nfev: 42
     njev: 14
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='trust-constr', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

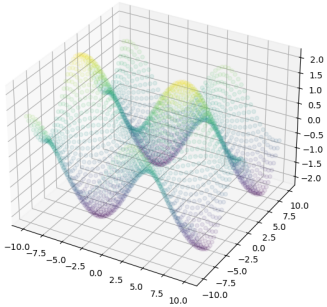
```
message: `gtol` termination condition is satisfied.
          success: True
           status: 1
              fun: -1.9999999999999996
                x: [-3.142e+00 -6.283e+00]
              nit: 8
            nfev: 24
            njev: 8
            nhev: 0
         cg_niter: 7
     cg_stop_cond: 4
             grad: [-0.000e+00 -0.000e+00]
  lagrangian_grad: [-0.000e+00 -0.000e+00]
           constr: []
              jac: []
      constr_nfev: []
      constr_njev: []
      constr_nhev: []
                v: []
           method: equality_constrained_sqp
```

```
      optimality: 0.0
constr_violation: 0
  execution_time: 0.0069081783294677734
       tr_radius: 16.583940593587347
  constr_penalty: 1.0
           niter: 8
```



```python
cb, history = callback_generator()
print(optimize.minimize(f, np.array([-8, -6]), method='powell', tol=10**-10, callback=cb))

x_history = read_history(history)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis', alpha=0.1)
x,y = x_history[:,0], x_history[:,1]
surf = ax.plot3D(x, y, f(x_history.T), color='orange', alpha=1)
```

```
message: Optimization terminated successfully.
 success: True
  status: 0
     fun: -2.0
       x: [-3.142e+00 -6.283e+00]
     nit: 2
   direc: [[ 1.000e+00  0.000e+00]
           [ 0.000e+00  1.000e+00]]
    nfev: 73
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

Moral of the story...

Checking for convexity is really important...
... but sometimes really difficult.
Make sure the functions you are optimizing are at least limited!

Nelder-Mead

Derivative-free, constrained, trust-region base approach

- Define a polyhedron (simplex) as domain ($n+1$ points in $\mathbb{R}^n$), the volume is the trust region
- Replace the worst vertex (highest f value) of the polyhedron with a "better" one
- until the trust region "radius" is within the required *tolx* tolerance, or *tolf* condition met

Let the simplex S have vertices $\mathbf{x_0}, ..., \mathbf{x_n}$. S is nondegenerate if:

- there are never more than 3 coplanar vertices
- the matrix $V(S) \in \mathcal{M}^{n \times n} = [\mathbf{x_n} - \mathbf{x_0}, \mathbf{x_{n-1}} - \mathbf{x_0}, ..., \mathbf{x_1} - \mathbf{x_0}]$ is nonsingular

Nelder-Mead vertex operations

$S = \mathbf{x_0}, ..., \mathbf{x_n}$, ordered such that $f(\mathbf{x_0}) \leq f(\mathbf{x_1}) \leq \dots f(\mathbf{x_n})$.
The centroid is :

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=0}^{n} \mathbf{x_i}$$

which is used to define the points along the "worst" direction:

$$\bar{x}(t) = \bar{\mathbf{x}} + t(\mathbf{x_n} - \mathbf{x})$$

- REFLECT: $\mathbf{x_n} = \bar{x}(-1)$
- EXPAND: $\mathbf{x_n} = \bar{x}(-2)$
- CONTRACT (inside): $\mathbf{x_n} = \bar{x}(-1/2)$
- CONTRACT (outside): $\mathbf{x_n} = \bar{x}(1/2)$
- SHRINK: $\mathbf{x_i} = (1/2)(\mathbf{x_0} + \mathbf{x_i})$ (in direction of $\mathbf{x_0}$)

Nelder-Mead crawl

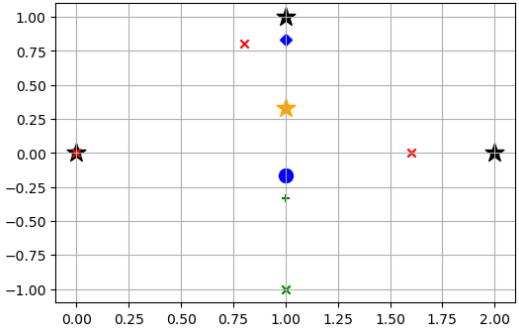
1. Given starting point $\mathbf{x_0}$, generate $S = \{\mathbf{x_i}\}$, $\mathbf{x}_i = x_0 + \alpha \mathbf{e}_i$
2. Check: if simplex too small or degenerate, end on best point
3. Compute centroid $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=0}^{n} \mathbf{x_i}$
4. Sort vertices $f(\mathbf{x_0}) \leq f(\mathbf{x_1}) \leq \ldots f(\mathbf{x_n})$, select worst $\mathbf{x}_n$
5. Try REFLECT, EXPAND, CONTRACT in, CONTRACT out. If new $\mathbf{x}$ is better than old $\mathbf{x}_n$ and within constraints, restart from (2) (prefer reflect/expand to contract!)
6. SHRINK and restart from (2)

Implementation

```python
import numpy as np
from scipy import optimize
from matplotlib import pyplot as plt
plt.rcParams['axes.grid'] = True
```

```python
# m vectors of n dimensions, m x n
def centroid(X):
    return 1/(X.shape[0]) * X.sum(axis=0)

def modify_x(x, centroid, operation):
    t = {'REFLECT': -1,
         'EXPAND': -2,
         'CONTRACT_IN': -0.75,
         'CONTRACT_OUT': 0.75}
    return centroid + t[operation]*(x-centroid)

def shrink_S(sorted_X, centroid):
    return (sorted_X + sorted_X[0])/1.25
```

```
X = np.array([[0 , 0], [2, 0], [1,1]])
plt.scatter(X[:,0], X[:,1], marker='*', s=200, color='black')

C = centroid(X)
plt.scatter(C[0], C[1], marker='*', s=200, color='orange')

RX = modify_x(X[2], C, 'REFLECT')
plt.scatter(RX[0], RX[1], marker='+', color='green');

RX = modify_x(X[2], C, 'EXPAND')
plt.scatter(RX[0], RX[1], marker='x', color='green');

RX = modify_x(X[2], C, 'CONTRACT_IN')
plt.scatter(RX[0], RX[1], marker='o', s=100, color='blue');

RX = modify_x(X[2], C, 'CONTRACT_OUT')
plt.scatter(RX[0], RX[1], marker='D', color='blue');

NX = shrink_S(X, C)
plt.scatter(NX[:,0], NX[:,1], marker='x', color='red');
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

```python
def sort_x(X, f):
    f_vals = np.apply_along_axis(f, axis=1, arr=X)
    return X[np.argsort(f_vals)]

def testf(x):
    return np.sum(x, axis=0)

X =  np.array([[5,4],[2,2],[3,7],[1,1],[8,4],[0,-1]])
sX = sort_x(X, testf)

print(testf(X.T))
print(testf(sX.T))
```

```
[ 9  4 10  2 12 -1]
[-1  2  4  9 10 12]
```

```python
def simplex_too_small(X, told):
    maxdist = 0
    for x in X:
        for y in X:
            d = np.sqrt(np.sum((x-y)**2))
            maxdist = max(maxdist, d)
    return maxdist <= told

X = np.array([[0,0],[2,0],[1,1]])
print(simplex_too_small(X, 2))
print(simplex_too_small(X, 1.9))
```

```
True
False
```

```python
def is_degenerate(X):
    return (np.linalg.matrix_rank(X[1:]-X[0]) < (len(X)-1))

X = np.array([[0,0],[2,0],[1,1]])
print(is_degenerate(X))
X = np.array([[0,0],[0.5,0.5],[1,1]])
print(is_degenerate(X))
```

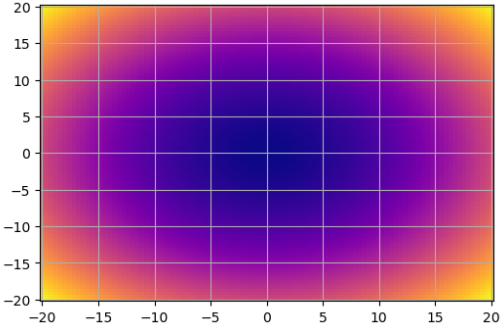
```
False
True
```

```python
def nelder_mead(f, x_0,  told=10**-10, maxiters=1000, callback=False):
    n = x_0.shape[0] # assume x_0 n-dimentional vector
    X = 5 * np.eye(n) * x_0 + x_0
    X = np.vstack([X, x_0])

    counter = 0
    while not simplex_too_small(X,told) and counter < maxiters:
        counter += 1
        callback(X)
        X = sort_x(X, f)
        C = centroid(X)

        f_worst = f(X[-1])

        x_reflect = modify_x(X[-1], C, 'REFLECT')
        f_x_reflect = f(x_reflect)

        if f_x_reflect <= f_worst:
            x_expand = modify_x(X[-1], C, 'EXPAND')
            f_x_expand = f(x_expand)
            if f_x_expand <= f_x_reflect:
                X[-1] = x_expand
            else:
                X[-1] = x_reflect
        else:
            x_contract_in = modify_x(X[-1], C, 'CONTRACT_IN')
            f_x_contract_in = f(x_contract_in)
            x_contract_out = modify_x(X[-1], C, 'CONTRACT_OUT')
            f_x_contract_out = f(x_contract_out)

            if f_x_contract_in <= f_worst:
```

```
32                  X[-1] = x_contract_in
33              elif f_x_contract_out <= f_worst:
34                  X[-1] = x_contract_out
35              else:
36                  X = shrink_S(X, C)
37
38
39      return sort_x(X, f)[0]
```

```python
def callback_generator():
    history = list()
    def cb(*args, **kwargs):
        history.append([args, kwargs])
    return cb, history

def read_history(h):
    return np.array([h[0][0] for h in history])

```

```python
def f(x):
    return np.sum(x**2, axis=0)

x = np.linspace(-20,20,100)
y = np.linspace(-20,20,100)
X,Y = np.meshgrid(x,y)
z = f(np.vstack([X.ravel(),Y.ravel()])).reshape((100,100))

plt.pcolormesh(x, y, z, cmap='plasma', shading='auto');
```



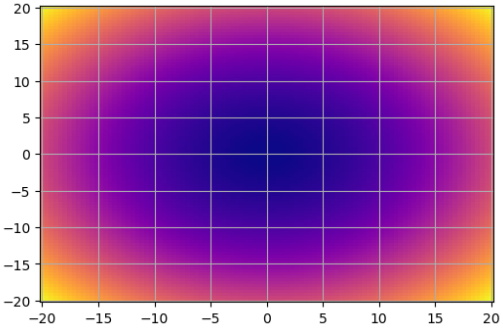
```python
cb, history = callback_generator()
sol = nelder_mead(f, np.array([-15,-15]), callback=cb)

print(sol, f(sol))
print(len(history))
print(read_history(history))
```

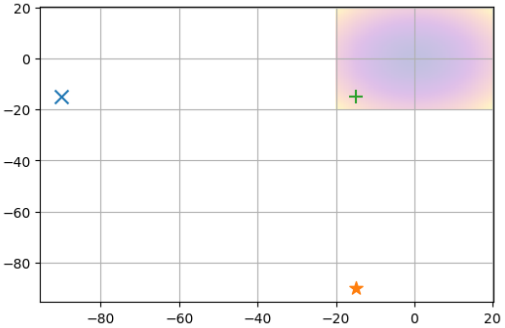
```
[ 1.28111797e-09 -6.25620062e-10] 2.032663716633584e-18
457
[[[-9.00000000e+01 -1.50000000e+01]
  [-1.50000000e+01 -9.00000000e+01]
  [-1.50000000e+01 -1.50000000e+01]]

 [[-1.50000000e+01 -1.50000000e+01]
  [-9.00000000e+01 -1.50000000e+01]
  [-6.50000000e+01  1.00000000e+01]]

 [[-1.50000000e+01 -1.50000000e+01]
  [-6.50000000e+01  1.00000000e+01]
  [ 1.00000000e+01  1.00000000e+01]]

 ...

 [[ 1.28283365e-09 -6.75181284e-10]
  [ 1.33215250e-09 -5.77558722e-10]
  [ 1.30663524e-09 -6.01589392e-10]]

 [[ 1.30663524e-09 -6.01589392e-10]
  [ 1.28283365e-09 -6.75181284e-10]
  [ 1.25731639e-09 -6.99211954e-10]]
```

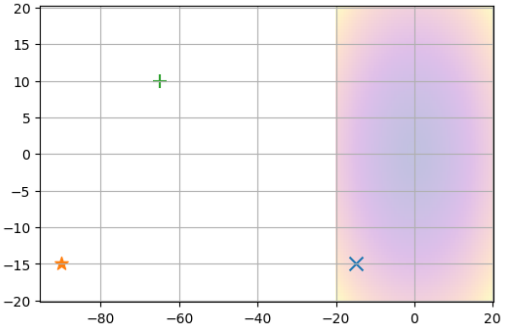
```
24
25 [[ 1.30663524e-09 -6.01589392e-10]
26  [ 1.25731639e-09 -6.99211954e-10]
27  [ 1.28111797e-09 -6.25620062e-10]]]
```

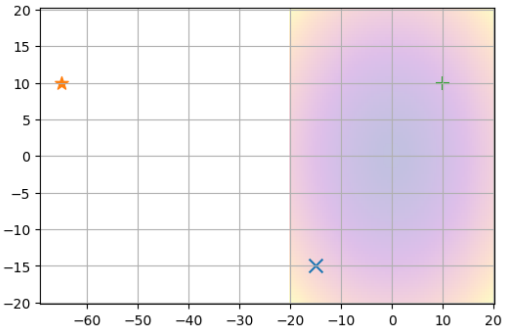
```python
cb, history = callback_generator()
sol = nelder_mead(f, np.array([-15,-15]), told=10**-1, callback=cb)

print(sol, f(sol))
print(len(history))
plt.pcolormesh(x, y, z, cmap='plasma', shading='auto');
s = lambda n: 100#/(n+1)
for n,points in enumerate(read_history(history)[:10]):
    plt.figure()
    plt.pcolormesh(x, y, z, cmap='plasma', shading='auto', alpha=0.25);
    a,b,c, = points
    plt.scatter(*a, s=s(n), marker='x');
    plt.scatter(*b, s=s(n), marker='*');
    plt.scatter(*c, s=s(n), marker='+');
```

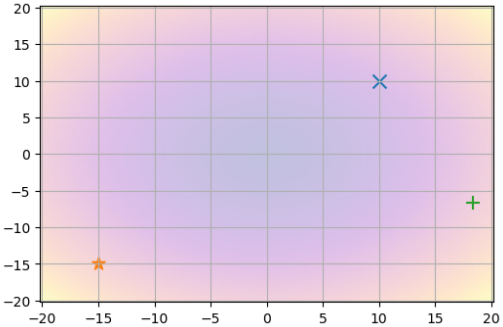
```
[ 0.00812179 -0.02246722] 0.0005707392225003524
23
```

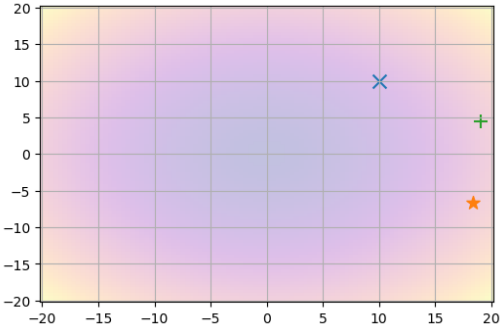

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

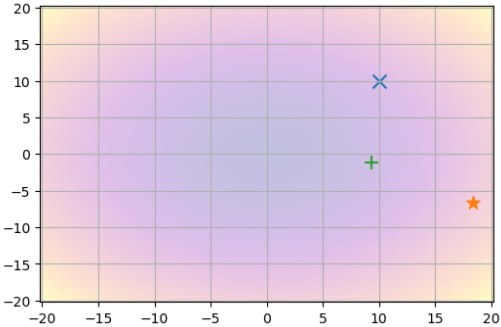

20
0
−20
−40
−60
−80
−80
−60
−40
−20
0
20

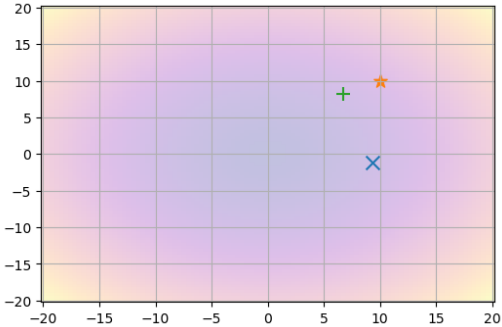

20
15
10
5
0
−5
−10
−15
−20
−80
−60
−40
−20
0
20

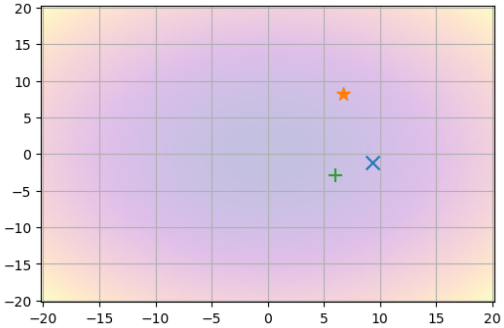

20
15
10
5
0
−5
−10
−15
−20
−60
−50
−40
−30
−20
−10
0
10
20

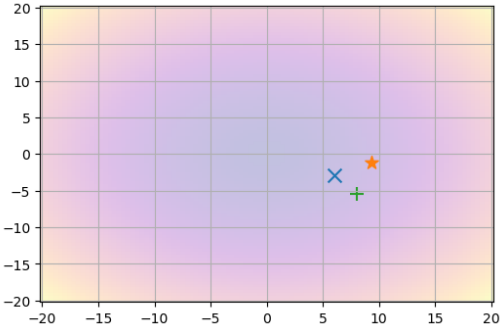

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

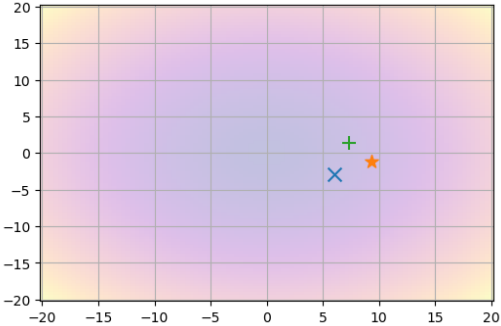

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

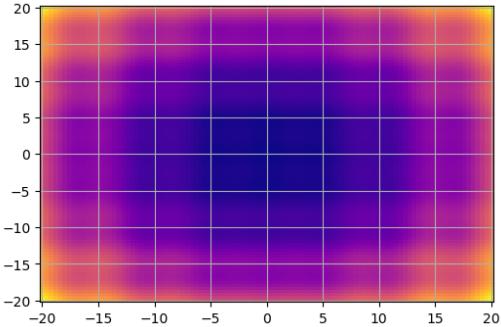

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

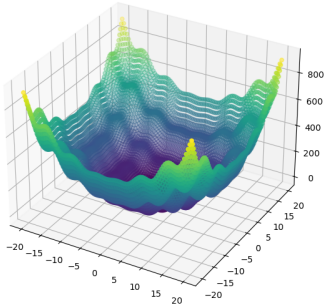

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

```python
from scipy.optimize import minimize
cb, history = callback_generator()
result = minimize(f, np.array([-15,-15]), method='Nelder-Mead', callback=cb)

print(result)
print(history)
```

```
message: Optimization terminated successfully.
       success: True
        status: 0
           fun: 1.5368435378969786e-09
             x: [-2.245e-05 -3.214e-05]
           nit: 48
          nfev: 87
 final_simplex: (array([[-2.245e-05, -3.214e-05],
                        [ 5.734e-05,  9.610e-06],
                        [ 5.199e-05, -4.571e-05]]), array([ 1.537e-09,  3.381e-09,  4.793e-09]))
[[(array([-15., -15.]),), {}], [(array([-14.625, -13.875]),), {}], [(array([-14.625, -13.875]),), {}],
↪  [(array([-12.75, -12.75]),), {}], [(array([-12.75, -12.75]),), {}], [(array([-10.125,  -9.375]),), {}],
↪  [(array([-10.125,  -9.375]),), {}], [(array([-3.75, -3.75]),), {}], [(array([-3.75, -3.75]),), {}],
↪  [(array([1.875, 2.625]),), {}], [(array([1.875, 2.625]),), {}], [(array([-0.75, -0.75]),), {}], [(array([-0.75,
↪  -0.75]),), {}], [(array([-0.75, -0.75]),), {}], [(array([0.46875, 0.28125]),), {}], [(array([0.1171875,
↪  0.4453125]),), {}], [(array([-0.22851562, -0.19335938]),), {}], [(array([0.20654297, 0.20361328]),), {}],
↪  [(array([-0.07507324, -0.21496582]),), {}], [(array([-0.08139038, -0.09951782]),), {}], [(array([0.06415558,
↪  0.02318573]),), {}], [(array([0.02461052, 0.05023384]),), {}], [(array([-0.01850367, -0.03140402]),), {}],
↪  [(array([-0.02749765,  0.0025295 ]),), {}], [(array([0.00080493, 0.01789829]),), {}], [(array([0.00080493,
↪  0.01789829]),), {}], [(array([0.01237757, 0.00477373]),), {}], [(array([-0.00466688,  0.00037047]),), {}],
↪  [(array([-0.00466688,  0.00037047]),), {}], [(array([-0.00466688,  0.00037047]),), {}], [(array([ 0.00311535,
↪  -0.00215119]),), {}], [(array([0.00279572, 0.00015819]),), {}], [(array([-0.00085567, -0.00031301]),), {}],
↪  [(array([-0.00085567, -0.00031301]),), {}], [(array([-0.00085567, -0.00031301]),), {}], [(array([-0.00085567,
↪  -0.00031301]),), {}], [(array([ 0.00031535, -0.00038227]),), {}], [(array([3.09977062e-04, 7.61451720e-05]),),
↪  {}], [(array([3.09977062e-04, 7.61451720e-05]),), {}], [(array([3.09977062e-04, 7.61451720e-05]),), {}],
↪  [(array([-1.27477568e-04, -4.11388547e-05]),), {}], [(array([-1.27477568e-04, -4.11388547e-05]),), {}],
↪  [(array([9.99153955e-05, 5.81477130e-05]),), {}], [(array([9.99153955e-05, 5.81477130e-05]),), {}],
↪  [(array([-2.24467344e-05, -3.21401253e-05]),), {}], [(array([-2.24467344e-05, -3.21401253e-05]),), {}],
```

```
def f(x):
    return np.sum(x**2 + 3*x*np.sin(x) , axis=0)

x = np.linspace(-20,20,100)
y = np.linspace(-20,20,100)
X,Y = np.meshgrid(x,y)
z = f(np.vstack([X.ravel(),Y.ravel()])).reshape((100,100))

plt.pcolormesh(x, y, z, cmap='plasma', shading='auto');
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
Z = f(np.vstack([X.ravel(),Y.ravel()]))
surf = ax.scatter3D(X, Y, Z, c=Z, cmap='viridis')
```


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


800
600
400
200
0
−20
−15
−10
−5
0
5
10
15
20

```
cb, history = callback_generator()
sol = nelder_mead(f, np.array([-15,-15]), callback=cb)

print(sol, f(sol))
print(len(history))
print(read_history(history))
```

```
[6.78086133e+00 2.77454271e-07] 55.691320151649876
1000
[[[-9.00000000e+01 -1.50000000e+01]
  [-1.50000000e+01 -9.00000000e+01]
  [-1.50000000e+01 -1.50000000e+01]]

 [[-1.50000000e+01 -1.50000000e+01]
  [-9.00000000e+01 -1.50000000e+01]
  [-6.50000000e+01  1.00000000e+01]]

 [[-1.50000000e+01 -1.50000000e+01]
  [-6.50000000e+01  1.00000000e+01]
  [ 1.00000000e+01  1.00000000e+01]]

 ...

 [[ 6.78086134e+00  2.76096811e-07]
  [ 6.78086134e+00  2.75794364e-07]
  [ 6.78086134e+00  2.76473094e-07]]

 [[ 6.78086134e+00  2.76473094e-07]
  [ 6.78086134e+00  2.76096811e-07]
  [ 6.78086133e+00  2.76775541e-07]]
```

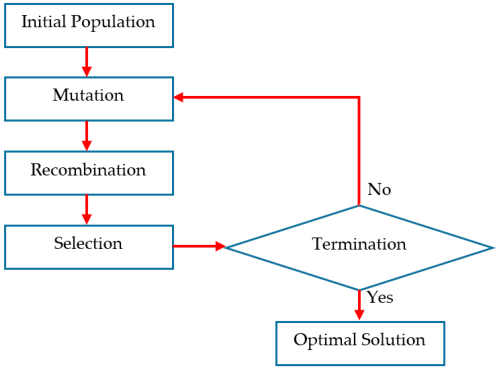
```
24
25 [[ 6.78086133e+00  2.76775541e-07]
26  [ 6.78086134e+00  2.76473094e-07]
27  [ 6.78086133e+00  2.77151824e-07]]]
```

```python
from scipy.optimize import minimize
cb, history = callback_generator()
result = minimize(f, np.array([-15,-15]), method='Nelder-Mead', callback=cb)

print(result)
```

```
message: Optimization terminated successfully.
       success: True
        status: 0
           fun: 473.5918048265534
             x: [-1.661e+01 -1.661e+01]
           nit: 39
          nfev: 72
 final_simplex: (array([[-1.661e+01, -1.661e+01],
                        [-1.661e+01, -1.661e+01],
                        [-1.661e+01, -1.661e+01]]), array([ 4.736e+02,  4.736e+02,  4.736e+02]))
```

Genetic Algorithms

- when function is complex, non-differentiable, multi-modal, vast search-space
- when dimensionality of search space is really high
- when there is high sensitivity to starting point
- when function is noisy or with random components



Differential Evolution

N_p population of vectors
Mutation:

$$\mathbf{u} = \mathbf{x_{r1}} + F(\mathbf{x_{r2}} - \mathbf{x_{r3}})$$

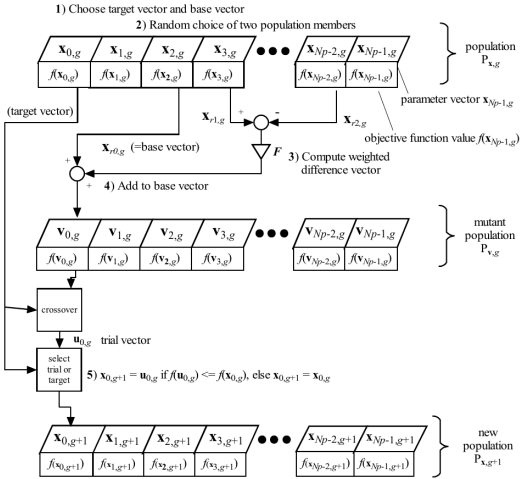
where F is the mutation *scale factor*
Crossover:

$$(\mathbf{v})_j = \begin{cases} (\mathbf{u_i})_j, & \text{if } rand(0,1) \leq C_r \\ (\mathbf{x_i})_j, & \text{otherwise} \end{cases}$$

Differential Evolution (2)

```
while (convergence criterion not met):
    for i in range(N_p):
        r1 = random_int(0, N_p)
        r2 = random_int(0, N_p)
        r3 = random_int(0, N_p)

        u = x_r1 + F * (x_r2 - x_r3)

        for j in range(len(u_i)):
            if rand(0,1) > C_r:
              u[j] = x_r1[j]

        if f(u) <= f(x_i):
            x_i = u
```

Differential Evolution (3)



Differential Evolution: Parameters

- N_p: pupulation size
- F: combination scaling factor
- C_r: crossover threshold
- mutating vector selection: random-to-random, current-to-best, best,
- number of difference vectors
- distribution for crossover test (random, exp, binomial...)

`DE/<mutating vector selection>/<mutators number>/<crossover distribution>`:
`DE/rand/1/bin` `DE/rand-to-best/2/exp`
How to initialize the population? (grid...)

```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import differential_evolution
import random
random.seed(69)
plt.rcParams['axes.grid'] = True
```

```python
def sort_x(X, f):
    f_vals = np.apply_along_axis(f, axis=1, arr=X)
    return X[np.argsort(f_vals)]

def simplex_too_small(X, told):
    maxdist = 0
    for x in X:
        for y in X:
            d = np.sqrt(np.sum((x-y)**2))
            maxdist = max(maxdist, d)
    return maxdist <= told

def grid(bounds, Np): #Bonuds: one row: min/max per dimension
    d = len(bounds) #dimensions
    n = int(Np**(1/d))# grid must have Npˆ(1/d) points per dimension to have Np ~ nˆd
    G = np.meshgrid(*[np.linspace(m,M,n) for m,M in bounds])
    return np.vstack([x.ravel() for x in G]).T
```

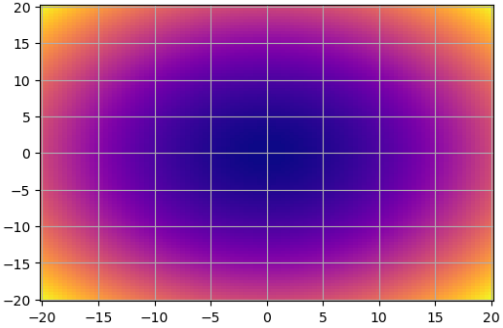
```
bounds = np.array([[-20,20],[-20,20]])
Np = 10
G = grid(bounds, Np)
print(G)
print(len(G))
```

```
[[-20. -20.]
 [  0. -20.]
 [ 20. -20.]
 [-20.   0.]
 [  0.   0.]
 [ 20.   0.]
 [-20.  20.]
 [  0.  20.]
 [ 20.  20.]]
9
```

```python
# DE/rand/1/rand
def DE_rand_1_rand(f, bounds, Np=10, F=0.9, Cr=0.05, told=1e-6, callback=False):
    d = len(bounds)
    P = grid(bounds, Np)
    Np = len(P)
    not_assigned_count = 0
    callback(P.copy())

    while not simplex_too_small(P, told) and not_assigned_count < 10:
        not_assigned_count += 1

        for i in range(Np):
            r1, r2, r3 = random.randint(0,Np-1), random.randint(0,Np-1), random.randint(0,Np-1)

            u = P[r1] + F * (P[r2]-P[r3])

            for j in range(d):
                if random.random() < Cr:
                    u[j] = P[r1][j]

            if f(u) < f(P[i]):
                P[i] = u
                not_assigned_count = 0

        callback(P.copy())

    return sort_x(P, f)[0]

```

```python
def callback_generator():
    history = list()
    def cb(*args, **kwargs):
        history.append([args, kwargs])
    return cb, history

def read_history(h):
    return np.array([h[0][0] for h in history])
```

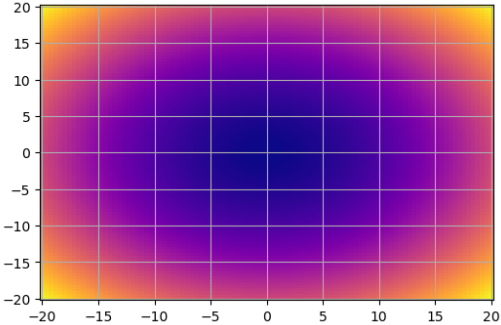
```python
def f(x):
    return np.sum(x**2, axis=0)

x = np.linspace(-20,20,100)
y = np.linspace(-20,20,100)
Xf,Yf = np.meshgrid(x,y)
zf = f(np.vstack([Xf.ravel(),Yf.ravel()])).reshape((100,100))

plt.pcolormesh(x, y, zf, cmap='plasma', shading='auto');
```

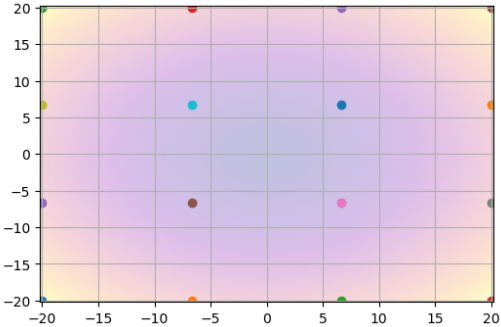


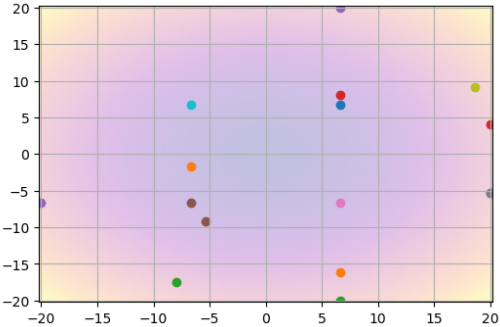
```
cb, history = callback_generator()
bounds = np.array([[-20,20],[-20,20]])
sol = DE_rand_1_rand(f, bounds, Np=20, callback=cb)

print(sol, f(sol))
print(len(history))
```

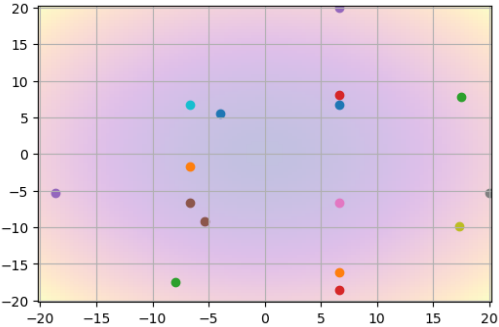
```
[ 1.18250598e-07 -5.67594669e-08] 1.7204841083990467e-14
89
```

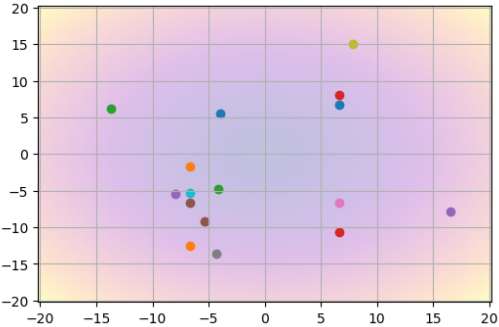
```
plt.pcolormesh(x, y, zf, cmap='plasma', shading='auto');
for n,points in enumerate(read_history(history)[:10]):
    plt.figure()
    plt.pcolormesh(x, y, zf, cmap='plasma', shading='auto', alpha=0.25);
    for p in points:
        plt.scatter(*p);

```

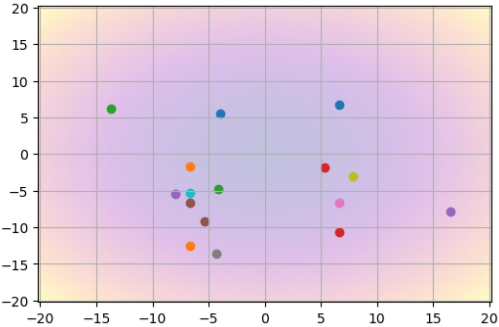


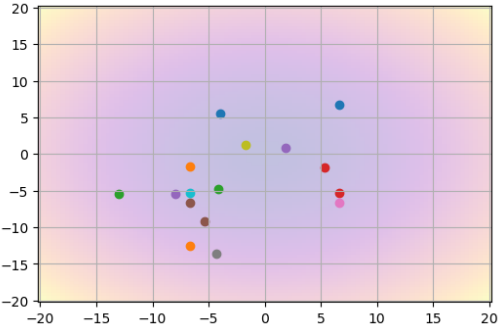

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

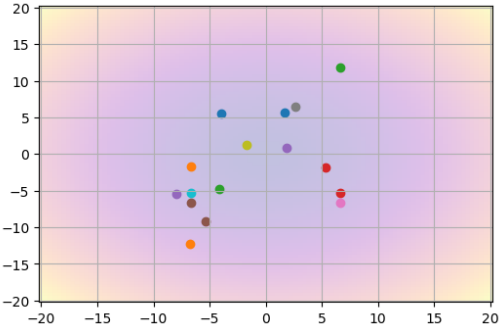

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

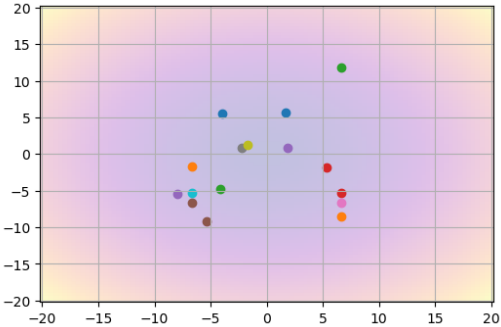

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

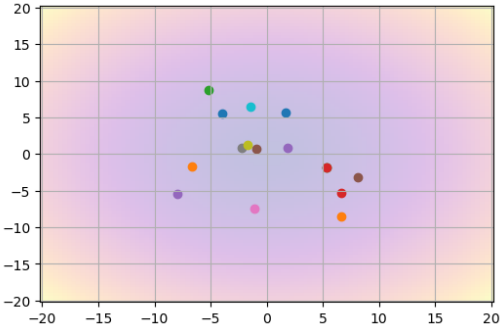

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

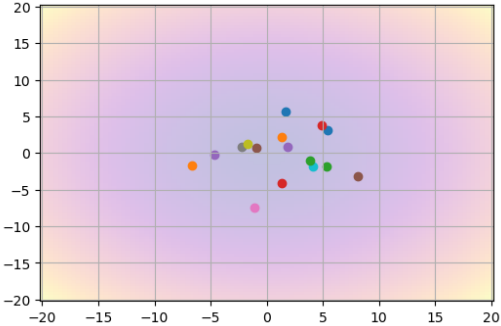

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

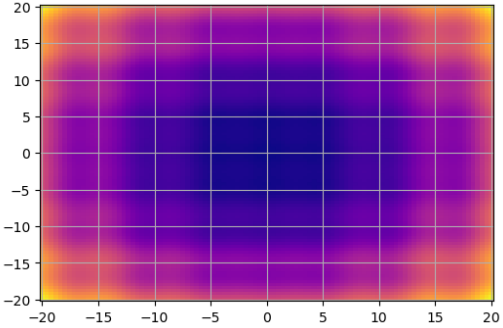

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

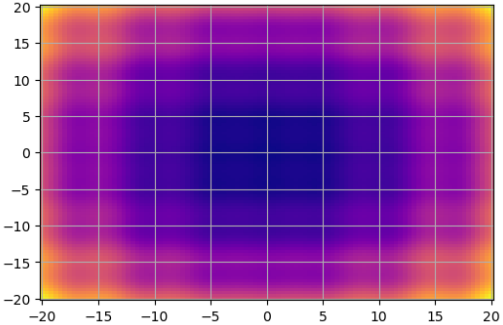
```python
def g(x):
    return np.sum(x**2 + 3*x*np.sin(x) , axis=0)

x = np.linspace(-20,20,100)
y = np.linspace(-20,20,100)
Xg,Yg = np.meshgrid(x,y)
zg = g(np.vstack([Xg.ravel(),Yg.ravel()])).reshape((100,100))

plt.pcolormesh(x, y, zg, cmap='plasma', shading='auto');
```

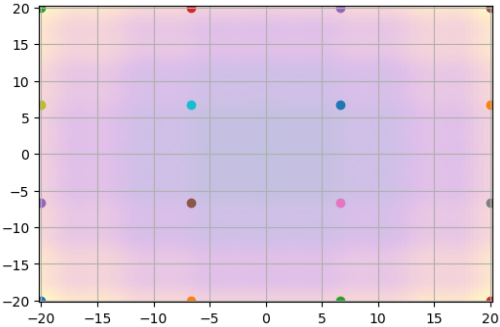


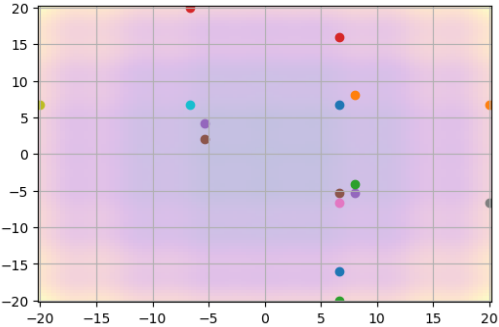
```python
cb, history = callback_generator()
sol = DE_rand_1_rand(g, bounds, Np=20, callback=cb)

print(sol, f(sol))
print(len(history))
```

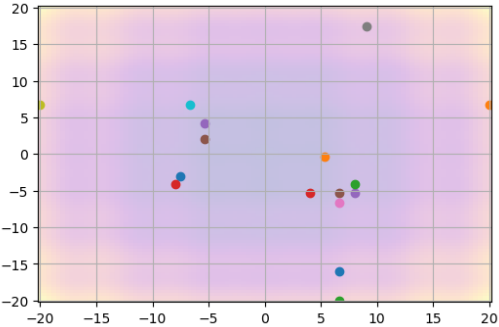
```
[-2.12358516e-08  9.96017632e-08] 1.0371472616947027e-14
94
```

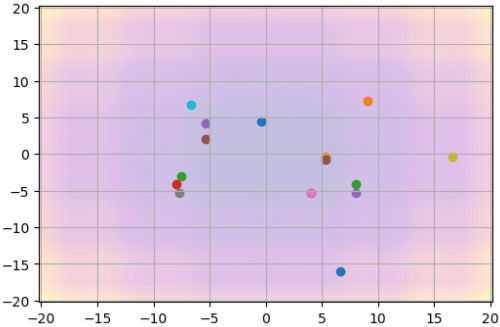
```
plt.pcolormesh(x, y, zg, cmap='plasma', shading='auto');
for n,points in enumerate(read_history(history)[:10]):
    plt.figure()
    plt.pcolormesh(x, y, zg, cmap='plasma', shading='auto', alpha=0.25);
    for p in points:
        plt.scatter(*p);

```

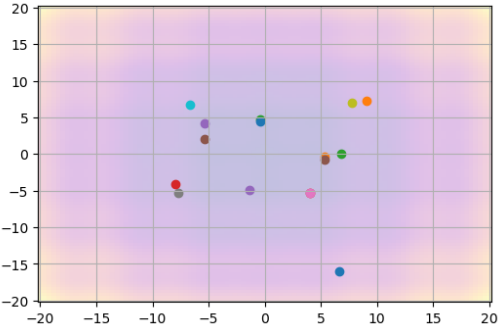


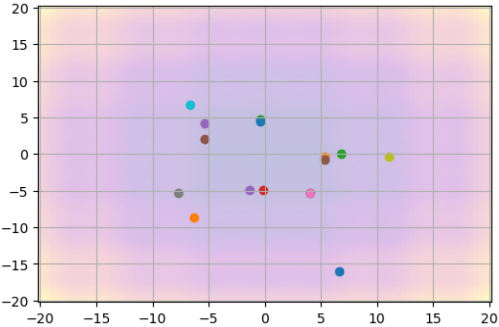

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

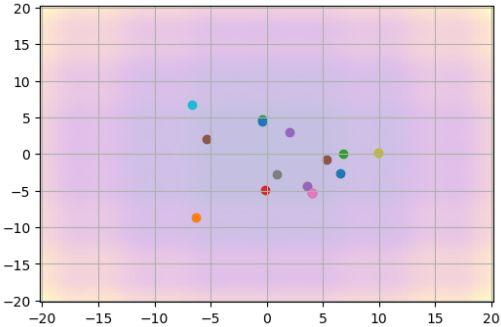

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

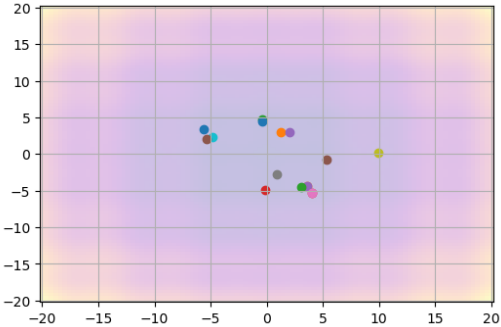

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

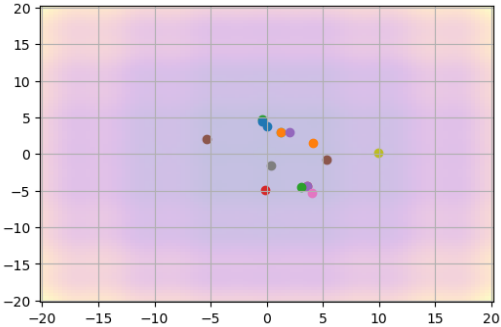

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

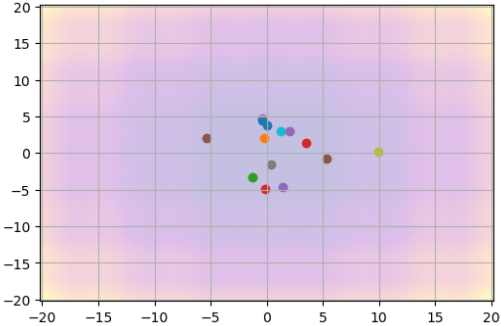

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

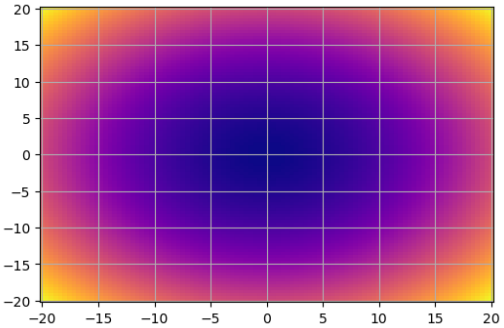

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

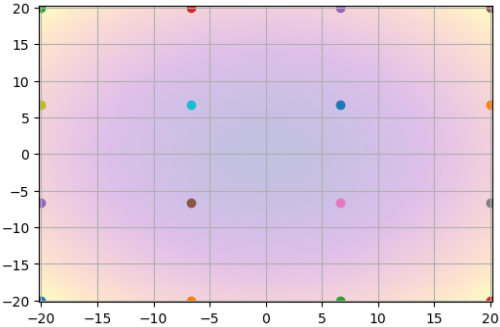
```python
# DE/best/1/rand
def DE_best_1_rand(f, bounds, Np=10, F=0.99, Cr=0.1, told=1e-6, callback=False):
    d = len(bounds)
    P = grid(bounds, Np)
    Np = len(P)
    not_assigned_count = 0
    callback(P.copy())

    while not simplex_too_small(P, told) and not_assigned_count < 10:
        not_assigned_count += 1

        for i in range(Np):
            P = sort_x(P, f)

            r1, r2, r3 = 0, random.randint(0,Np-1), random.randint(0,Np-1)
            u = P[r1] + F * (P[r2]-P[r3])

            for j in range(d):
                if random.random() < Cr:
                    u[j] = P[r1][j]

            if f(u) < f(P[i]):
                P[i] = u
                not_assigned_count = 0
        callback(P.copy())

    return sort_x(P, f)[0]

```

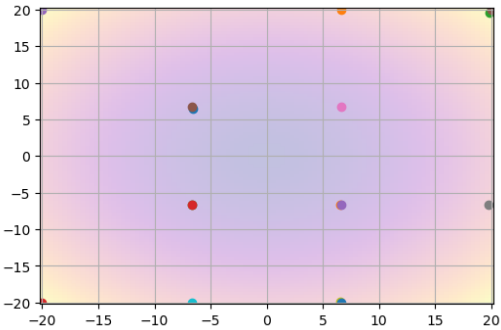
```
cb, history = callback_generator()
sol = DE_best_1_rand(f, bounds, Np=20, callback=cb)

print(sol, f(sol))
print(len(history))
```

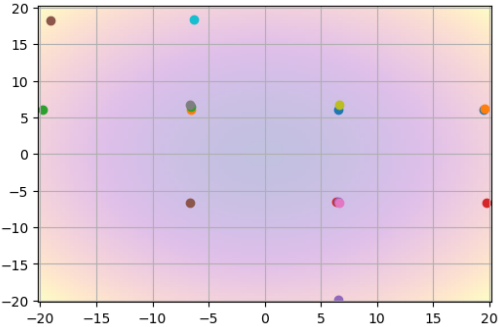
```
[ 6.53201333e+00 -3.76826660e-04] 42.66719832884287
16
```

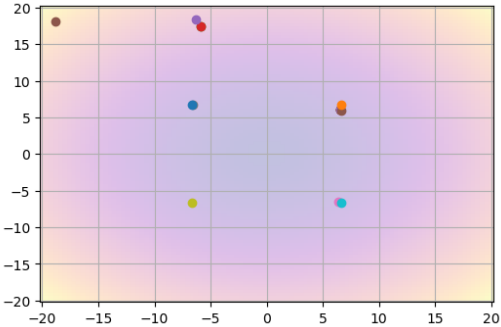
```
plt.pcolormesh(x, y, zf, cmap='plasma', shading='auto');
for n,points in enumerate(read_history(history)[:10]):
    plt.figure()
    plt.pcolormesh(x, y, zf, cmap='plasma', shading='auto', alpha=0.25);
    for p in points:
        plt.scatter(*p);
```

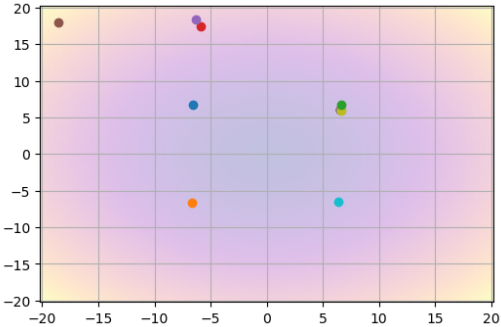


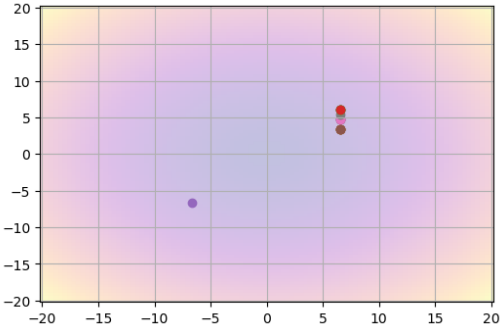

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

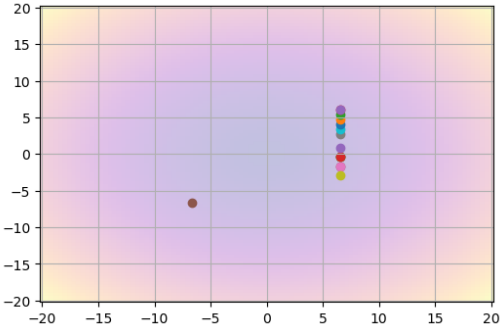

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

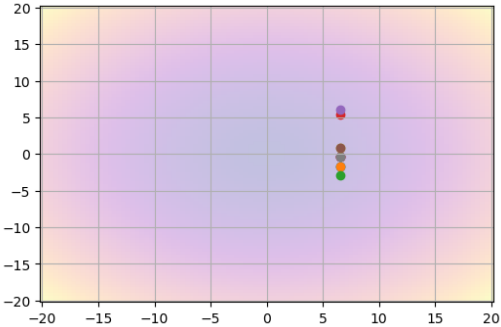

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

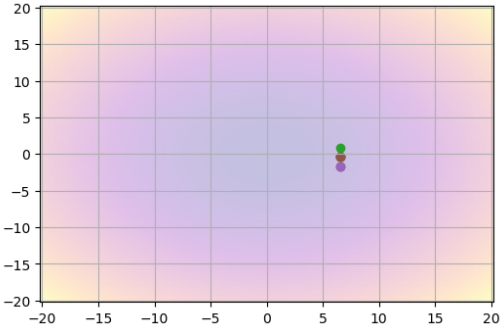

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

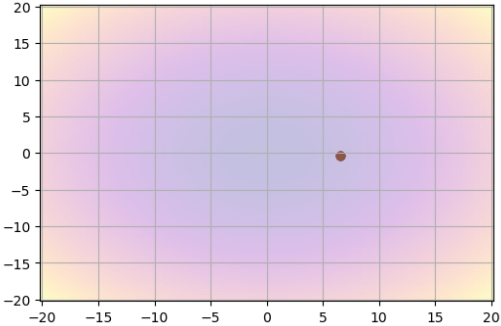

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

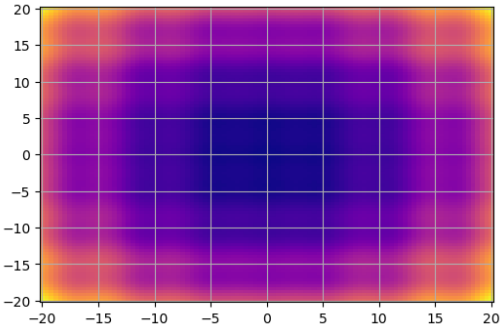

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

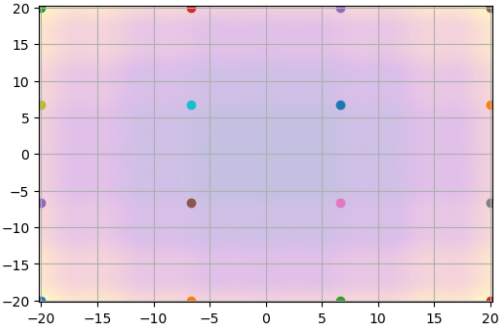

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

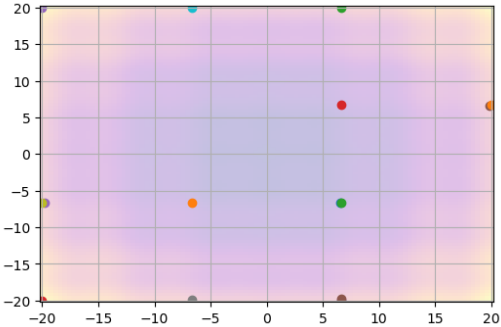
```python
cb, history = callback_generator()
sol = DE_best_1_rand(g, bounds, Np=20, callback=cb)

print(sol, f(sol))
print(len(history))
```

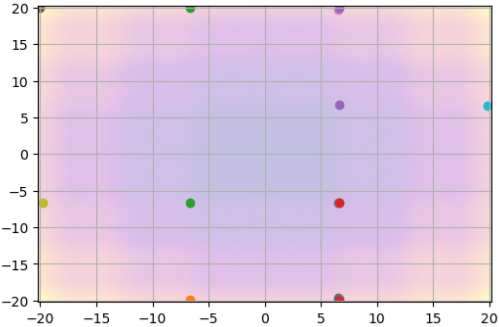
```
[-4.23869592e+00 -1.33360706e-06] 17.9665431138232
35
```

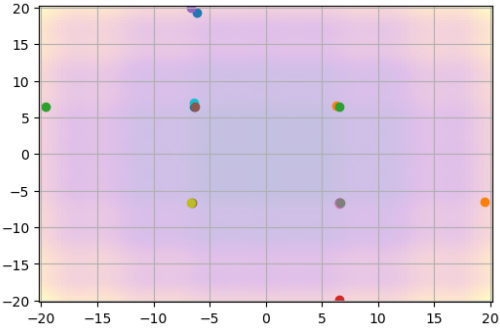
```
plt.pcolormesh(x, y, zg, cmap='plasma', shading='auto');
for n,points in enumerate(read_history(history)[:10]):
    plt.figure()
    plt.pcolormesh(x, y, zg, cmap='plasma', shading='auto', alpha=0.25);
    for p in points:
        plt.scatter(*p);
```

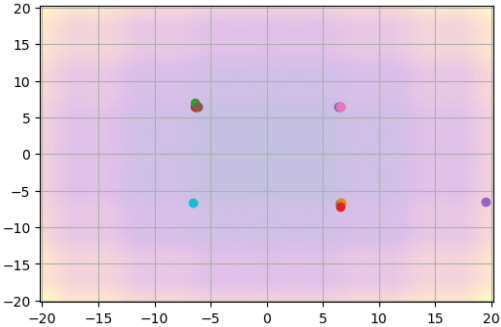


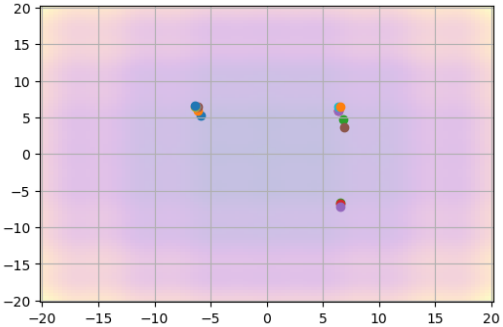

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

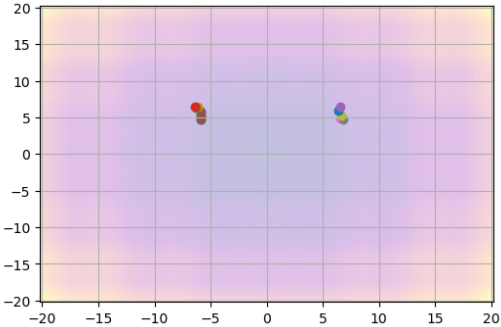

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

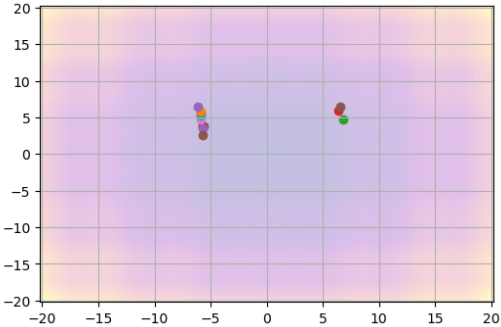

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

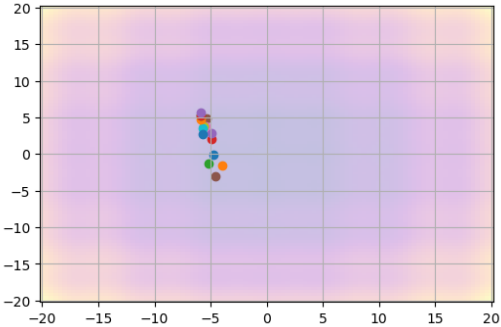

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

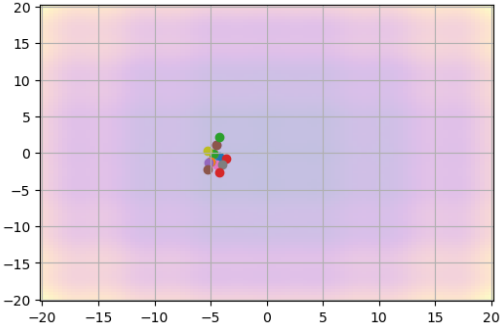

20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

```python
cb, history = callback_generator()
sol = differential_evolution(f, np.array([[-20,20],[-20,20]]), callback=cb)

print(sol)
print(len(history))
```

```
message: Optimization terminated successfully.
             success: True
                 fun: 0.0
                   x: [ 0.000e+00  0.000e+00]
                 nit: 92
                nfev: 2793
          population: [[ 0.000e+00  0.000e+00]
                       [ 0.000e+00  0.000e+00]
                       ...
                       [ 0.000e+00  0.000e+00]
                       [ 0.000e+00  0.000e+00]]
 population_energies: [ 0.000e+00  0.000e+00 ...  0.000e+00  0.000e+00]
92
```

```
print(history[:3])
```

```
[[(array([-2.08614115,  2.1683338 ]), 0.012852546783936242), {}], [(array([0.86537587, 0.04008659]),
↪  0.008914919251420144), {}], [(array([ 0.11947614, -0.82902206]), 0.011033056011863563), {}]]
```

Thanks!

Samuele Carli

carlisamuele@csspace.net
www.entersys.it
as-ai.org