Optimization

Introduction to Numerical Problem Solving, Spring 2017

Helsinki Metropolia University of Applied Sciences

Optimum point (minimum or maximum)

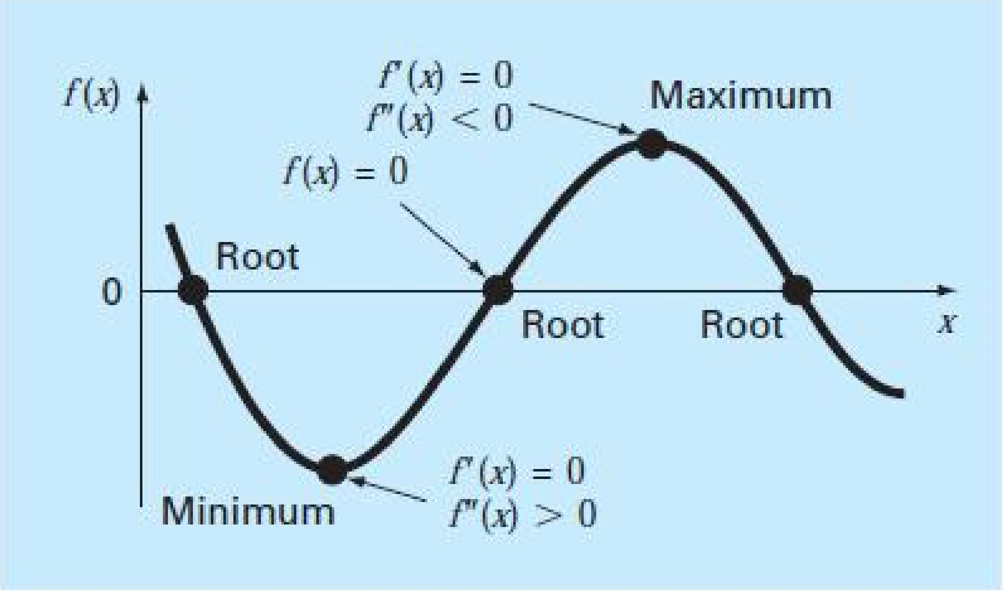
At the optimium the curve is flat, e.g $f'(x) = 0$.

If the point is maximum

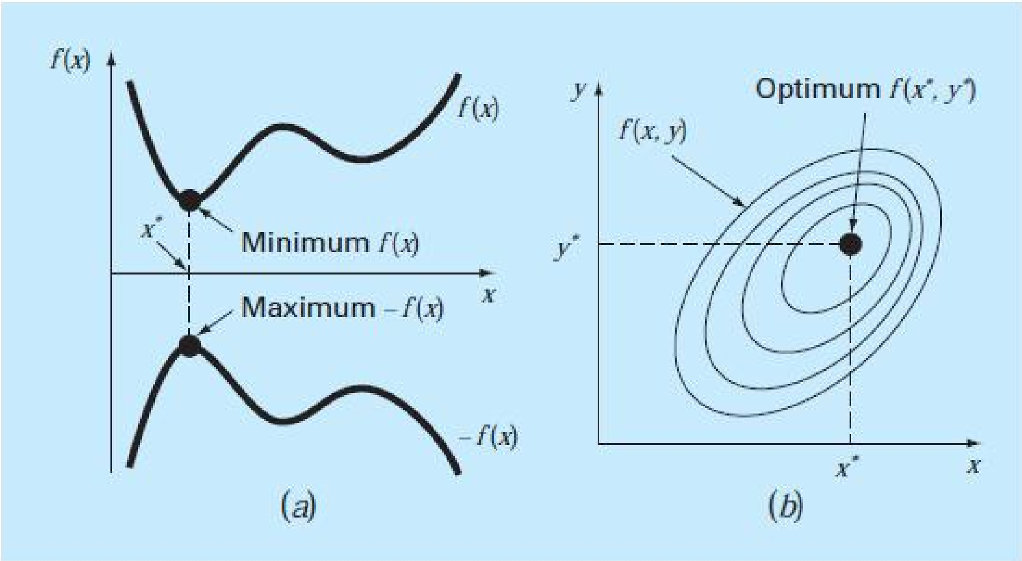
then $f''(x) < 0$.

If the point is minimum

then $f''(x) > 0$



Minimum is Maximum, also in 2D



Findining the minimum of f(x) is same as finding the maximum of -f(x) and vice versa.

2D (or higher dimensional) optimization is similarly either finding the lowest or highest point in surface but now in 2D.

Optimization methods

1D unconstrained

- Golden-section search
- Parabolic interpolation
- Newton’s method
- Brent’s method

2D, 3D, ND unconstrained

- Direct methods
- Gradient methods

Constrained optimization

- Linear programming
- Nonlinear constrained
- Software packages

Optimization

Local Optimization

minimize(fun, x0[, args, method, jac, hess, ...])	Minimization of scalar function of one or more variables.
minimize_scalar(fun[, bracket, bounds, ...])	Minimization of scalar function of one variable.
OptimizeResult	Represents the optimization result.
OptimizeWarning	

The minimize function supports the following methods:

- minimize(method='Nelder-Mead')
- minimize(method='Powell')
- minimize(method='CG')
- minimize(method='BFGS')
- minimize(method='Newton-CG')
- minimize(method='L-BFGS-B')
- minimize(method='TNC')
- minimize(method='COBYLA')
- minimize(method='SLSQP')
- minimize(method='dogleg')
- minimize(method='trust-ncg')

The minimize_scalar function supports the following methods:

- minimize_scalar(method='brent')
- minimize_scalar(method='bounded')
- minimize_scalar(method='golden')

```
from scipy.optimize
import minimize
```

Old style = Don’t use anymore

The specific optimization method interfaces below in this subsection are not recommended for use in new scripts; all of these methods are accessible via a newer, more consistent interface provided by the functions above.

General-purpose multivariate methods:

fmin(func, x0[, args, xtol, ftol, maxiter, ...])	Minimize a function using the downhill simplex algorithm.
fmin_powell(func, x0[, args, xtol, ftol, ...])	Minimize a function using modified Powell's method.
fmin_cg(f, x0[, fprime, args, gtol, norm, ...])	Minimize a function using a nonlinear conjugate gradient algorithm.
fmin_bfgs(f, x0[, fprime, args, gtol, norm, ...])	Minimize a function using the BFGS algorithm.
fmin_ncg(f, x0, fprime[, fhess_p, fhess, ...])	Unconstrained minimization of a function using the Newton-CG method.

Constrained multivariate methods:

fmin_l_bfgs_b(func, x0[, fprime, args, ...])	Minimize a function func using the L-BFGS-B algorithm.
fmin_tnc(func, x0[, fprime, args, ...])	Minimize a function with variables subject to bounds, using gradient information in a truncated Newton algorithm.
fmin_cobyla(func, x0, cons[, args, ...])	Minimize a function using the Constrained Optimization BY Linear Approximation (COBYLA) method.
fmin_slsqp(func, x0[, eqcons, f_eqcons, ...])	Minimize a function using Sequential Least SQuares Programming
differential_evolution(func, bounds[, args, ...])	Finds the global minimum of a multivariate function.

Univariate (scalar) minimization methods:

fminbound(func, x1, x2[, args, xtol, ...])	Bounded minimization for scalar functions.
brent(func[, args, brack, tol, full_output, ...])	Given a function of one-variable and a possible bracketing interval, return the minimum of the function isolated to a fractional precision of tol.
golden(func[, args, brack, tol, full_output])	Return the minimum of a function of one variable.

More optimization functions

Equation (Local) Minimizers

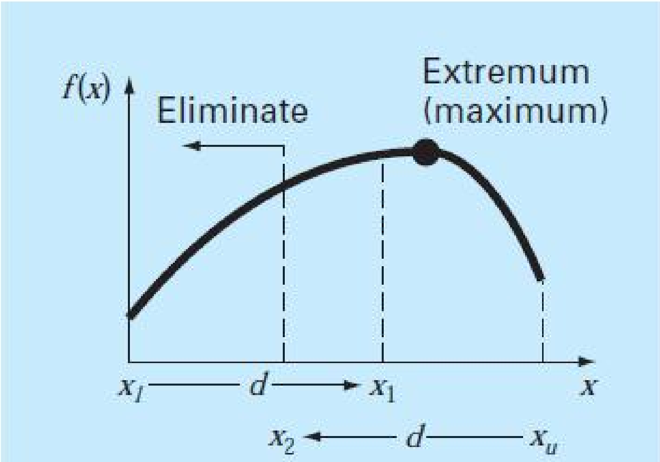
leastsq(func, x0[, args, Dfun, full_output, ...])	Minimize the sum of squares of a set of equations.
least_squares(fun, x0[, jac, bounds, ...])	Solve a nonlinear least-squares problem with bounds on the variables.
nnls(A, b)	Solve `argmin_x || Ax - b ||_2` for `x>=0`.
lsq_linear(A, b[, bounds, method, tol, ...])	Solve a linear least-squares problem with bounds on the variables.

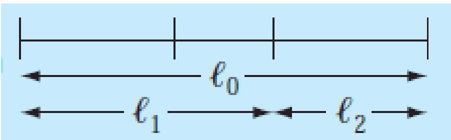
Global Optimization

basinhopping(func, x0[, niter, T, stepsize, ...])	Find the global minimum of a function using the basin-hopping algorithm
brute(func, ranges[, args, Ns, full_output, ...])	Minimize a function over a given range by brute force.
differential_evolution(func, bounds[, args, ...])	Finds the global minimum of a multivariate function.

1D unconstrained search

Golden-section search





$$l_0 = l_1 + l_2$$
$$\frac{l_1}{l_0} = \frac{l_2}{l_1} = R$$
$$\Rightarrow R = \frac{\sqrt{5}-1}{2} \approx 0.61803 \ldots$$

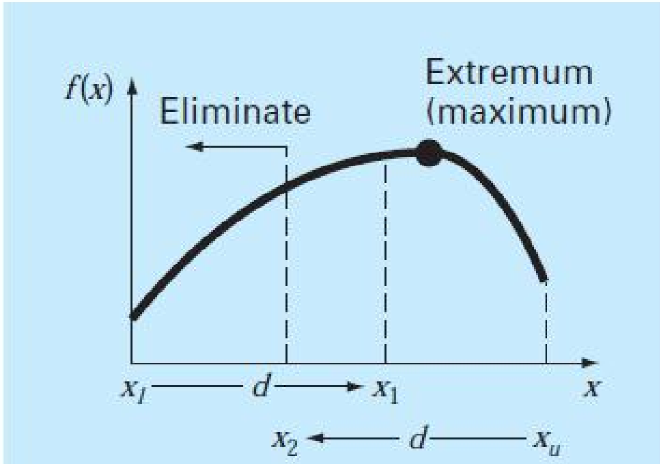
Start with two initial guesses x_l, x_u, that bracket a local extremum of f(x). Select two interior points according to Golden-ratio

$$d = \frac{\sqrt{5}-1}{2}(x_u - x_l)$$
$$x_1 = x_l + d$$
$$x_2 = x_u - d$$

If $f(x_1) > f(x_2)$ then
$$x_l = x_2$$
...

If $f(x_2) > f(x_1)$ then
$$x_u = x_1$$
...

Code for Golden-ratio Search




```
def goldenRatioSearch(f, a, b, tol = 1e-8, maxiter = 100):
    r = (sqrt(5) - 1)/2
    xopt = (b + a)/2
    fx = f(xopt)
    n = 0
    while n < maxiter:
        n = n + 1
        d = r*(b - a)
        x1 = a + d
        x2 = b - d
        f1 = f(x1)
        f2 = f(x2)
        if f1 > f2:
            a = x2
            x2 = x1
            x1 = a + d
            f2 = f1
            f1 = f(x1)
            xopt = x1
            fx = f(xopt)
        else:
            b = x1
            x1 = x2
            x2 = b - d
            f1 = f2
            f2 = f(x2)
            xopt = x1
            fx = f(xopt)
        if xopt != 0:
            ea = (1 - r)*abs((b-a)/xopt)
        if ea < tol:
            break
    return xopt, fx, ea, n
```

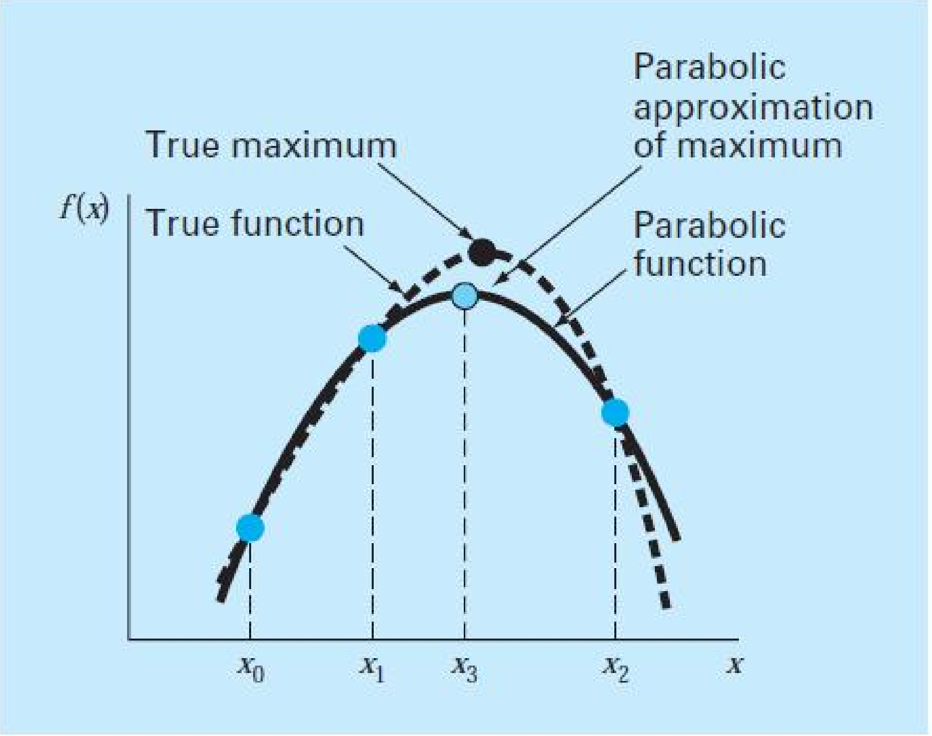
Parabolic interpolation search

$$x_3 = \frac{f(x_0)\left(x_1^2 - x_2^2\right) + f(x_1)\left(x_2^2 - x_0^2\right) + f(x_2)\left(x_0^2 - x_1^2\right)}{2\, f(x_0)(x_1 - x_2) + 2\, f(x_1)(x_2 - x_0) + 2\, f(x_2)(x_0 - x_1)}$$

Second-order polynomial provides a good approximation to the shape of f(x) near an optimum. If we have three points, we can fit a parabola and solve for an estimate of the optimal maximum x_3.

Similar to bisection or golden-section search one of the end points (x_0, x_2) is replaced with one of the inner points (x_1, x_3).

Iteration is continued until the required tolerance is achieved.



Parabolic interpolation search algorithm

Note: a = x0 and b = x3

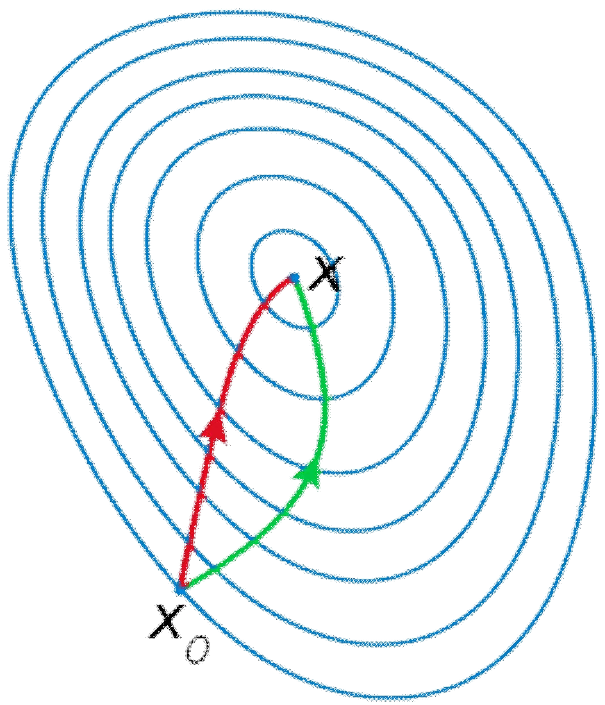
```
    fa = f(a)
    fb = f(b)
    n = 0
    while n < maxiter:
        n = n + 1
        A = fa*(x1**2 - b**2) + f1*(b**2 - a**2) + fb*(a**2 - x1**2)
        B = 2*fa*(x1 - b) + 2*f1*(b - a) + 2*fb*(a - x1)
        x2 = A/B
        if x1 < x2:
            a = x1
            fa = f1
        else:
            b = x1
            fb = f1
        d = r*(b-a)
        x1 = a + d
        f1 = f(x1)
```

Newton-Raphson’s method for optimization

Finding minimum/maximum

f’(x) = 0

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

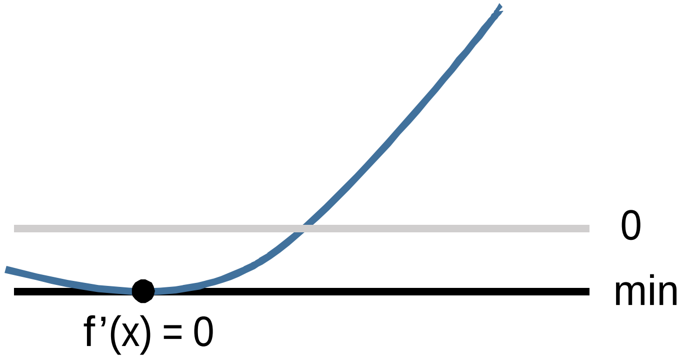


Finding extreme or root?

Finding minimum/maximum

f'(x) = 0

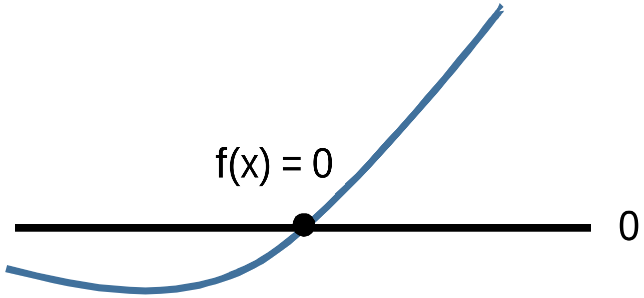
$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$



Finding a root

f(x) = 0

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



Finite difference derivatives

Finite diffence derivative

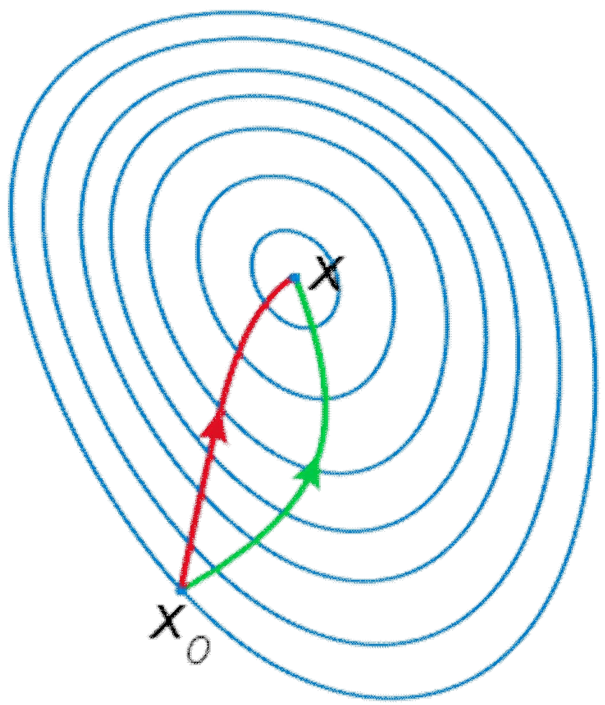
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Higher degree derivative

2nd order forward

$$f''(x) \approx \frac{\Delta_h^2[f](x)}{h^2} = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}$$

Geometric interpretation in higher dimensions



At each iteration a step towards the maximum/minimum is taken at the surface of f(x1, x2).

Newton-Raphson's method uses curvature information to take a more direct route (red line).

Gradient-descent method follows the steepest descent path (green line).

Algorithm for Newton-Raphson method

```
n = 0
h = some_small_value
hh = h*h
while n < maxiter:
    n = n + 1
    fp = (f(x0 + h) - f(x0))/h
    fpp = (f(x0 + h) - 2*f(x0) + f(x0 - h))/hh
    if fpp != 0:
        x1 = x0 - fp/fpp
    else:
        # Does this work?
        x1 = x0 + tol
    if x1 != 0:
        ea = abs((x1 - x0)/x1)
    else:
        # Can we do this way?
        ea = abs(x1 - x0)
    if ea < tol:
        break
    x0 = x1
```

Convex functions

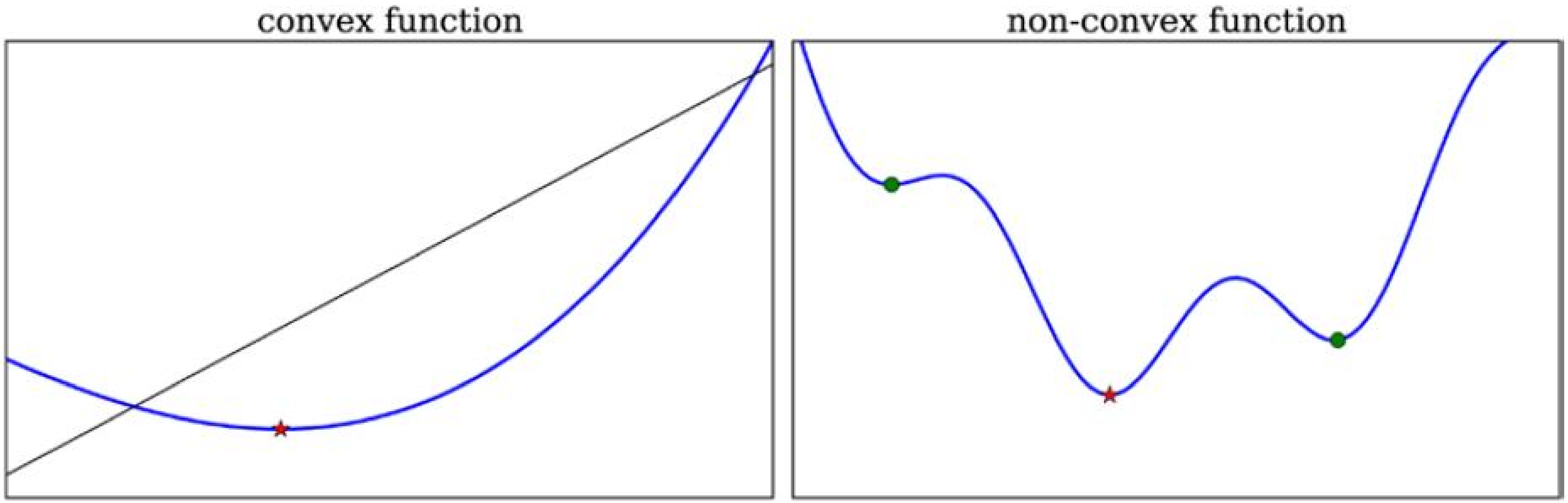


Figure 6-1. *Illustration of a convex function (left), and a non-convex function (right) with a global minima and two local minima*

Stationary points $f'(x) = 0$

$f''(x) > 0$ $f''(x) < 0$ $f''(x) = 0$

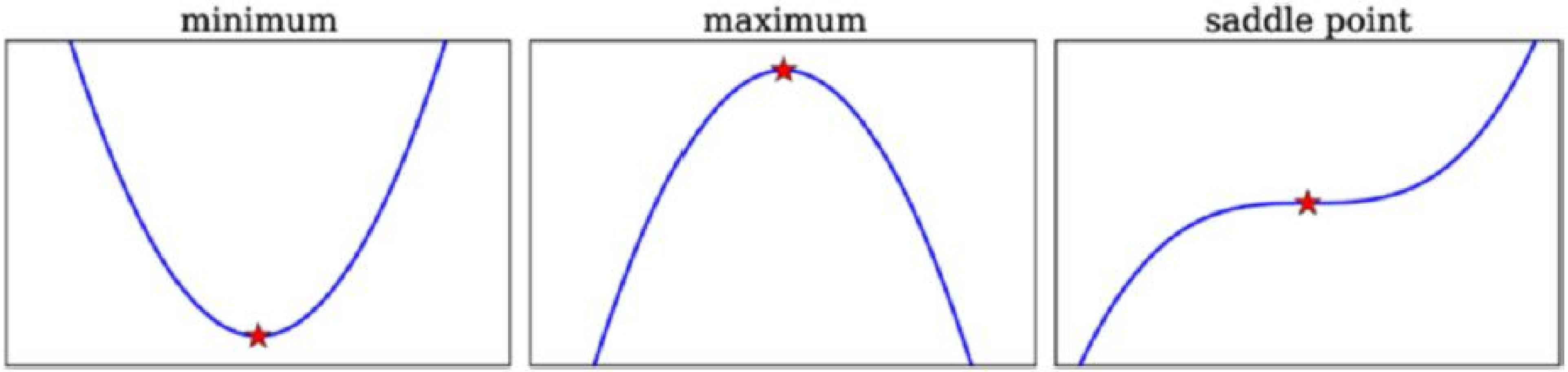
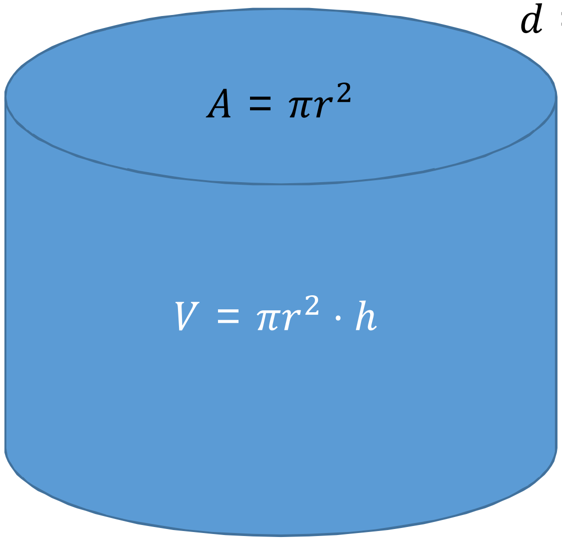


Figure 6-2. *Illustration of different stationary points of a one-dimensional function*

Minimize the surface of unit cylinder V = 1

$d = 2\pi r$

$A = 2\pi r h$



$h = V / \pi r^2$

$$A = 2 \cdot \pi r^2 + \frac{2V}{r}$$

Minimize $A(r) = 2\pi r^2 + \frac{2V}{r}$