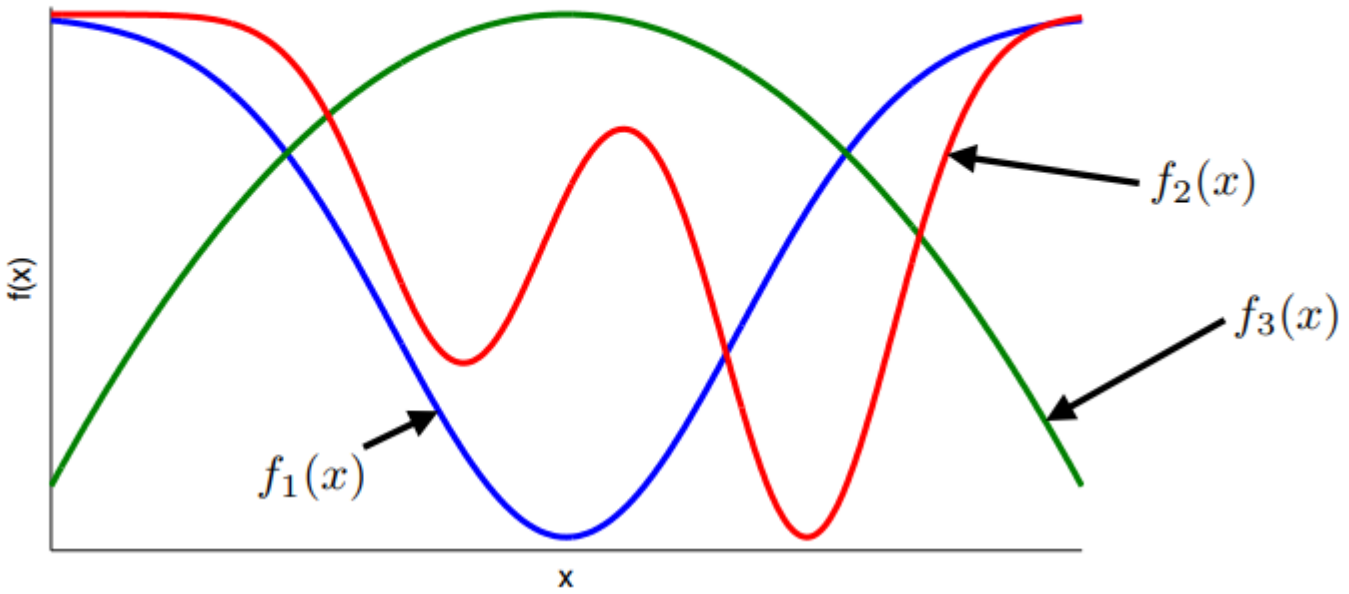
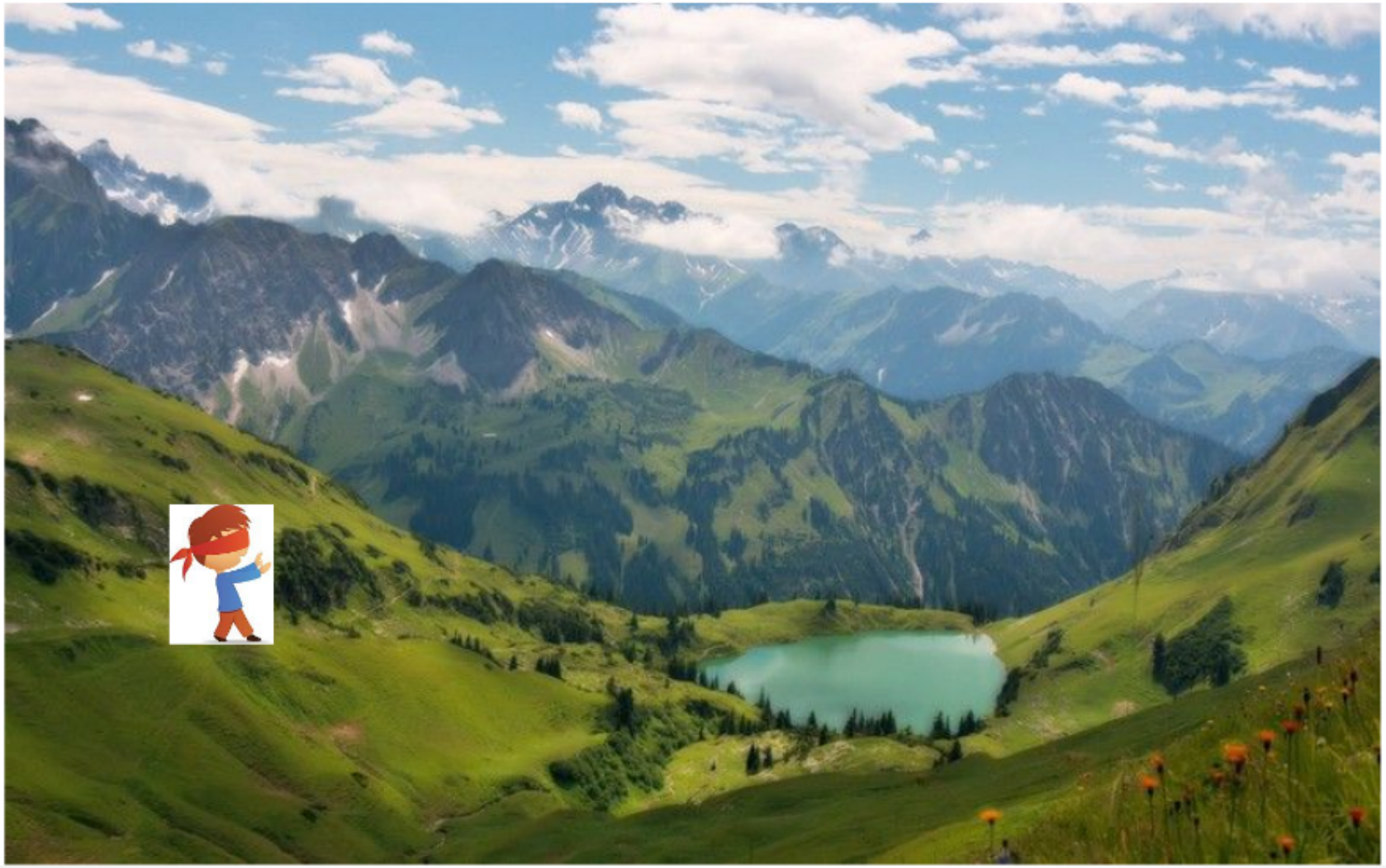
Gradient Descent

Gradient Descent Single Variable

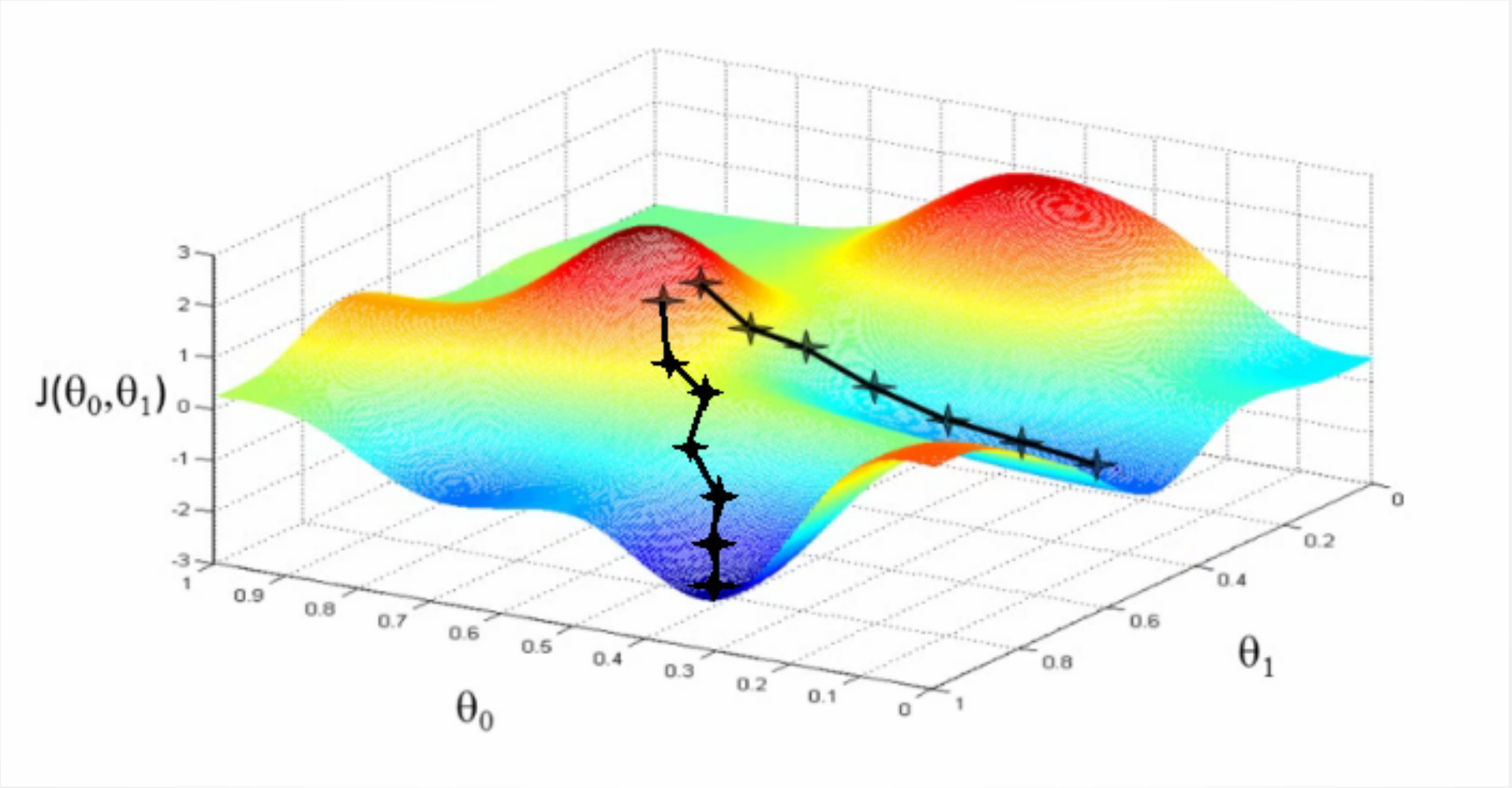
- The function can have one, multiple or no **local optima**



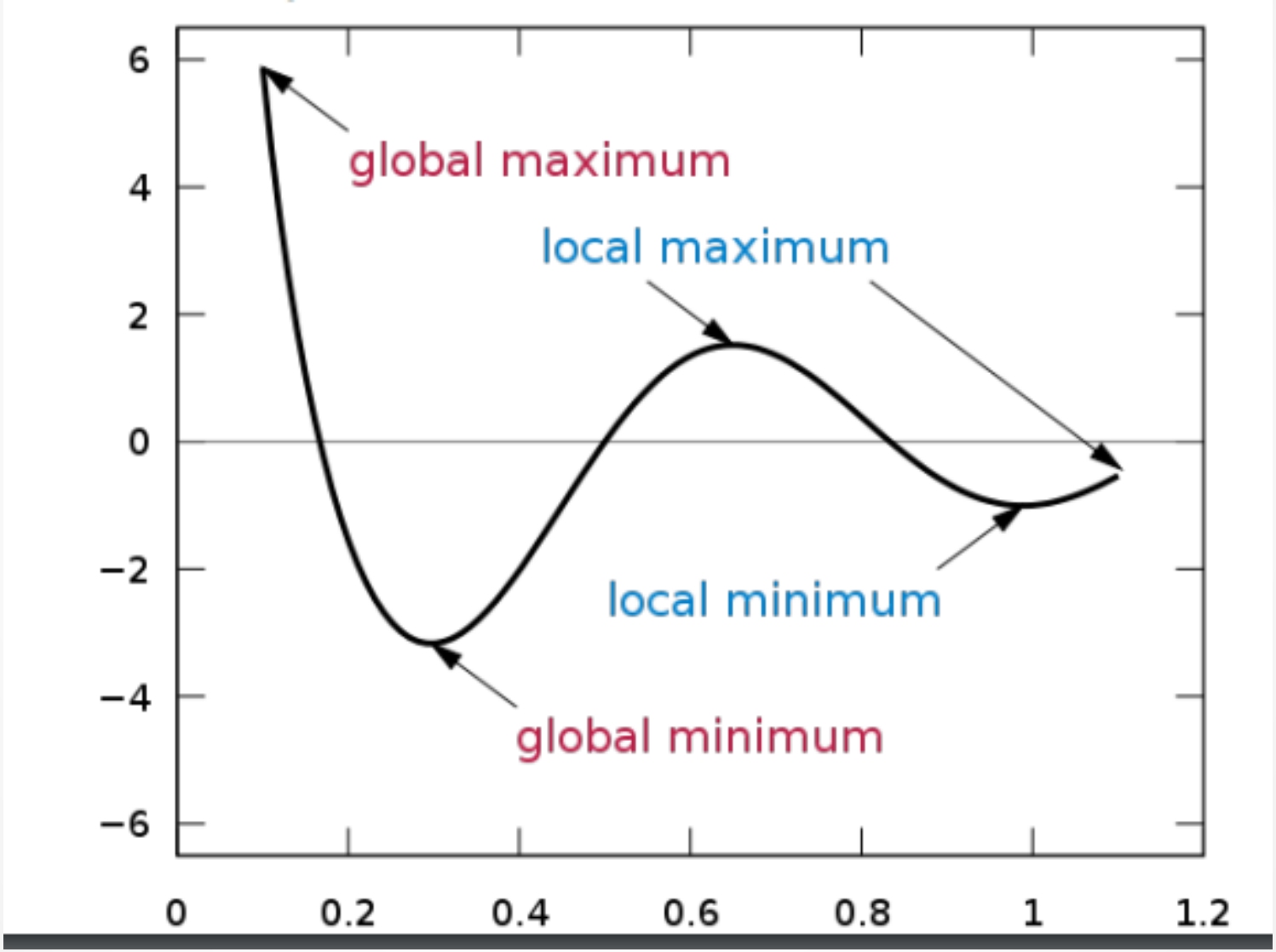
Gradient Descent 2 Variables



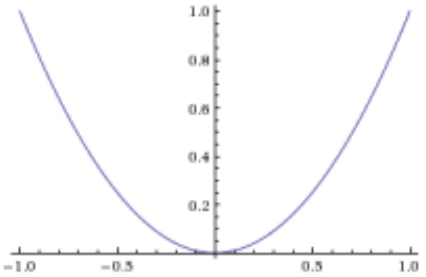
Gradient Descent 2 Variables



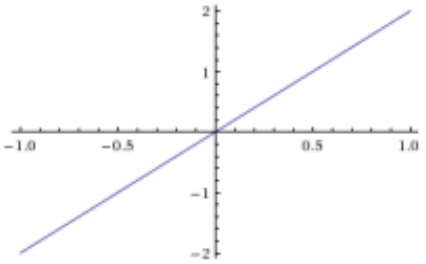
Global/Local Minimum/Maximum



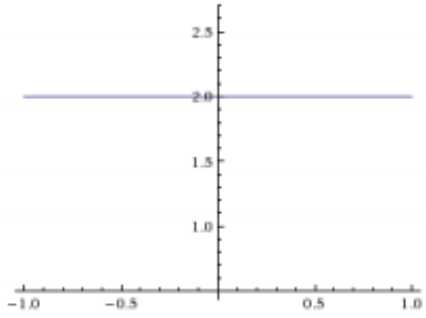
Derivative



$f(x)=x^2$



$f'(x)=df/dx=2x$



$f''(x)=d^2 f/dx=2$

- Slope of the tangent line

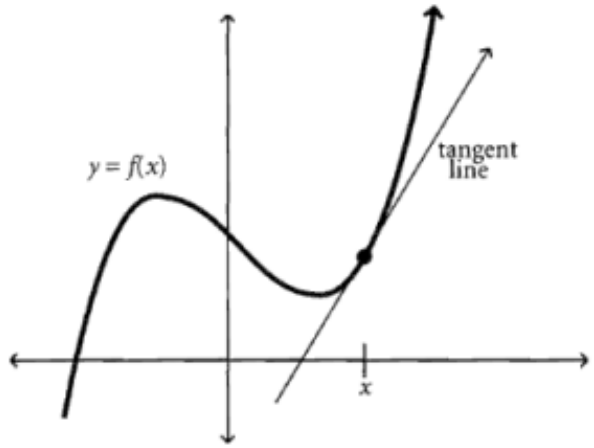
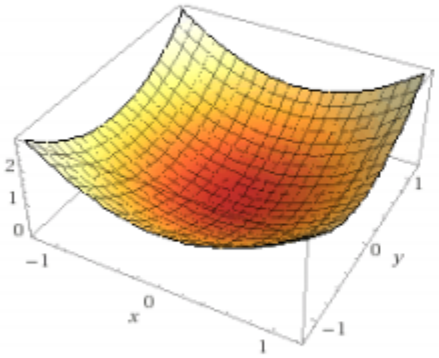


Figure 6.2

- Easy when a function is univariate

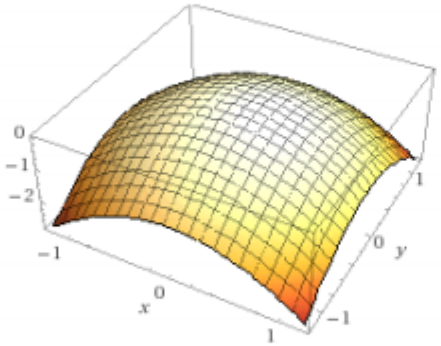
Partial Derivative–Multivariate Functions

For multivariate functions (e.g two variables) we need partial derivatives – one per dimension. Examples of multivariate functions:



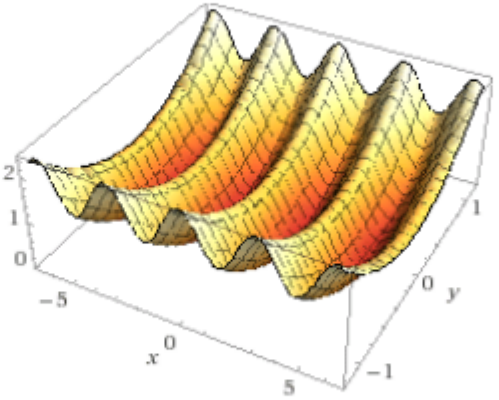
$f(x,y)=x\hat{}2+y\hat{}2$

Convex!

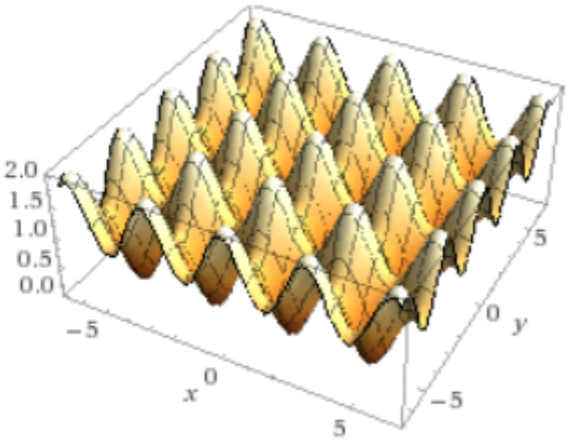


$f(x,y)=-x\hat{}2-y\hat{}2$

Concave!

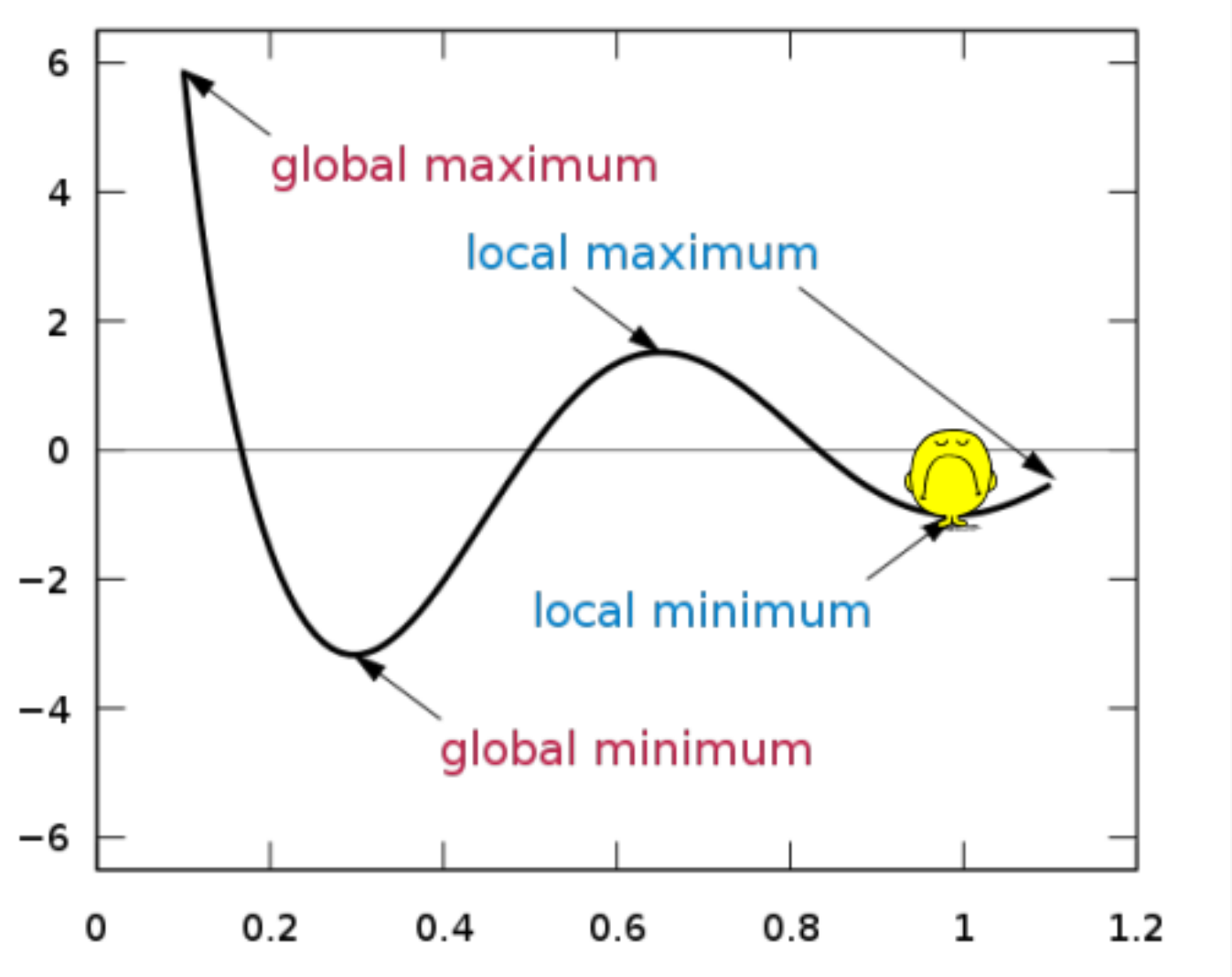


$f(x,y)=\cos\hat{}2\,(x)+y\hat{}2$



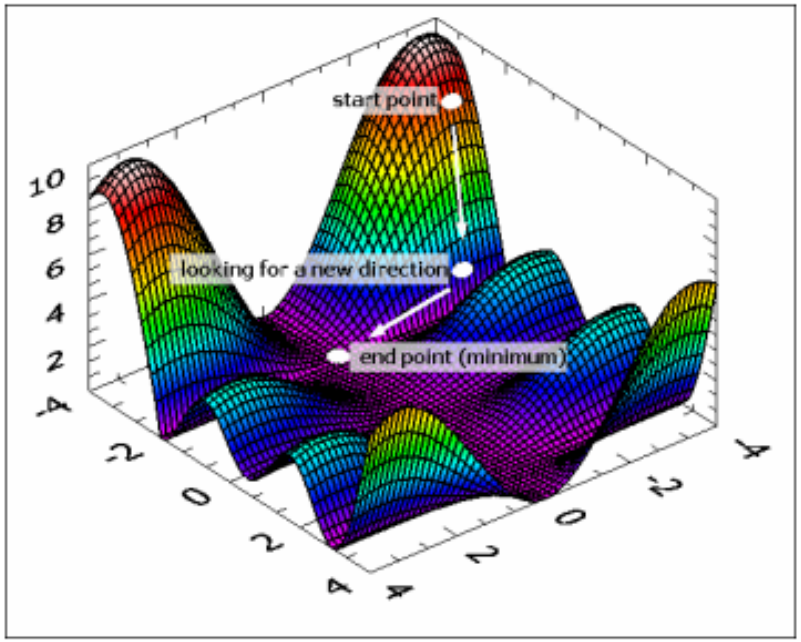
$f(x,y)=\cos\hat{}2\,(x)+\cos\hat{}2\,(y)$

Potential issues of gradient descent

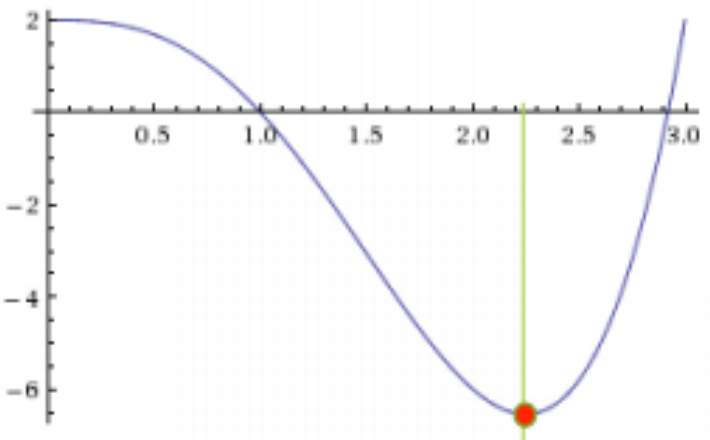


Gradient Descent Algorithm & Walkthrough

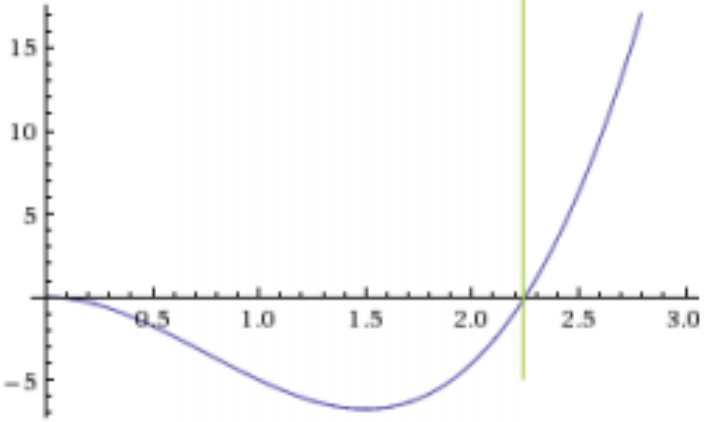
- Idea
  - Start somewhere
  - Take steps based on the gradient vector of the current position till convergence
- Convergence :
  - happens when change between two steps < ε



Gradient Descent Algorithm & Walkthrough



$f(x)=x^4-3x^3+2$



$f'(x)=4x^3-9x^2$

Gradient Descent Algorithm & Walkthrough

EVOLVE
MACHINE LEARNERS

```
def f_x(x):
    ret=x**4 -3*x**3+2
    return ret

def f_prime_x(x):
    ret=4*x**3-9*x**2
    return ret
```

```
def gradient_descent():
    x_old=0.0
    x_new=6.0
    precission=0.00001
    max_iter=1000
    learning_rate=0.01
#    learning_rate=0.0001
#    learning_rate=0.1
    iter=0

    while abs(x_new-x_old)>precission and iter<max_iter:
        x_old=x_new
        x_new=x_old-learning_rate*f_prime_x(x_old)
        iter=iter+1
        print('iter',iter,'new_x', x_new)

    print('Local Minimum at', x_new)
    print('Function Value ', f_x(x_new))

gradient_descent()
```

Bruto Force

```
def f_x(x):
    ret=x**4 -3*x**3+2
    return ret
```

```
def bruto_force():
    x_start=-10.0
    x_end=10.0
    max_iter=1000
    learning_rate=0.01
#    learning_rate=0.0001
#    learning_rate=0.1
    iter=0
    min_x=x_start
    min_f_x=f_x(min_x)
    x=x_start

    while x<x_end:
        new_f_x=f_x(x)
        if new_f_x<min_f_x:
            min_x=x
            min_f_x=new_f_x
        x=x+learning_rate

    print('Local Minimum at', min_x)
    print('Function Value ', min_f_x)

bruto_force()
```