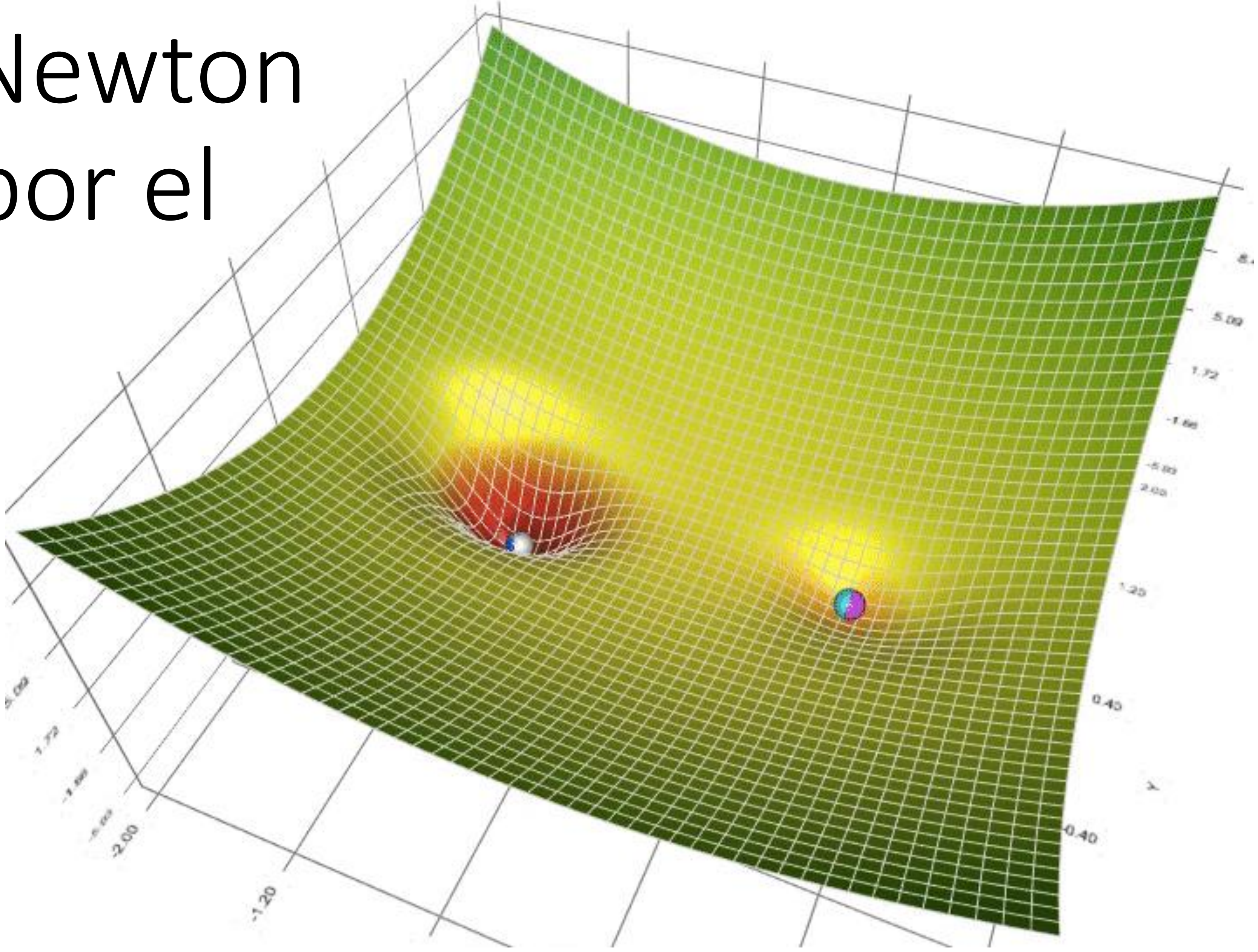
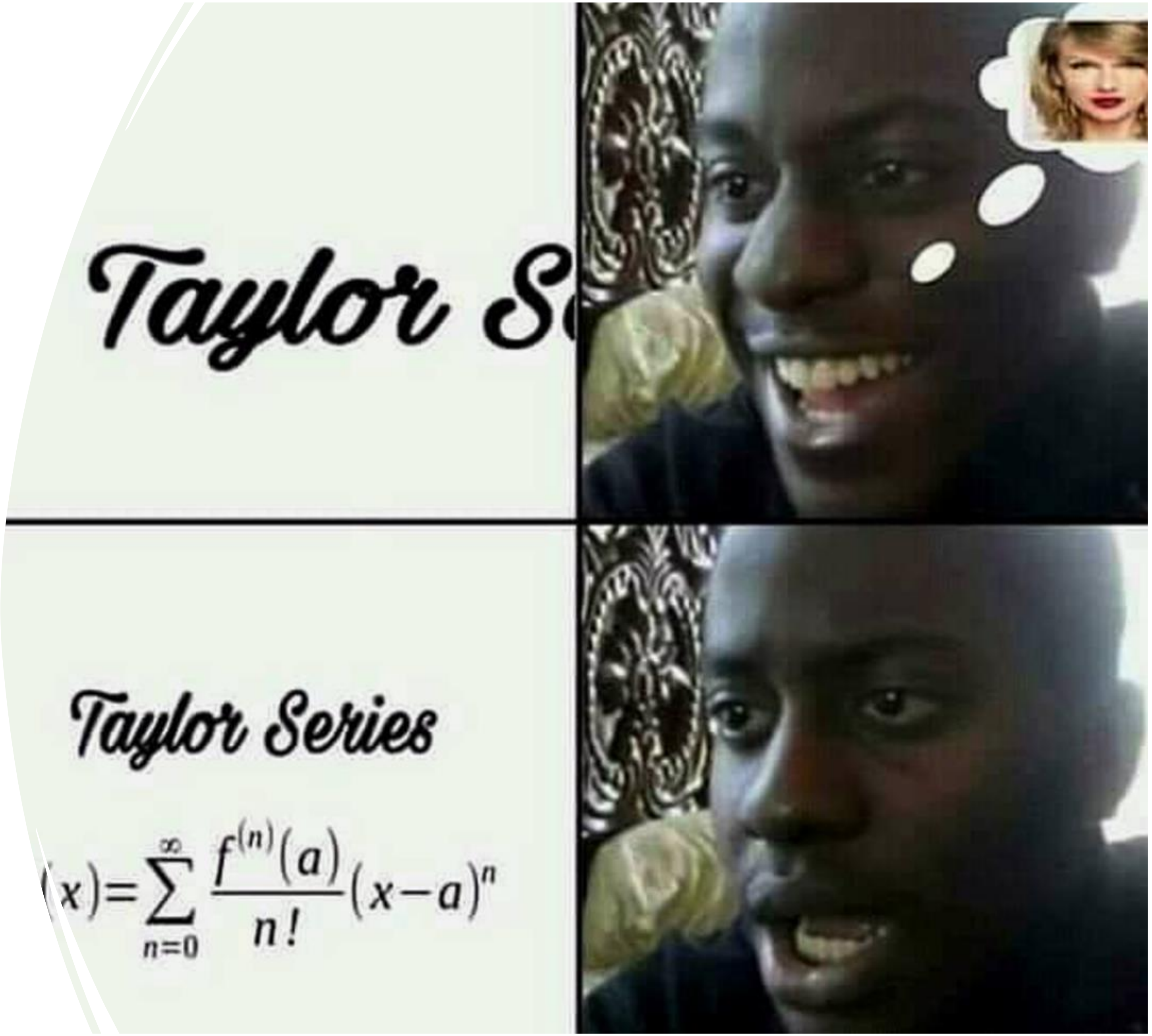
Método de Newton y Descenso por el gradiente



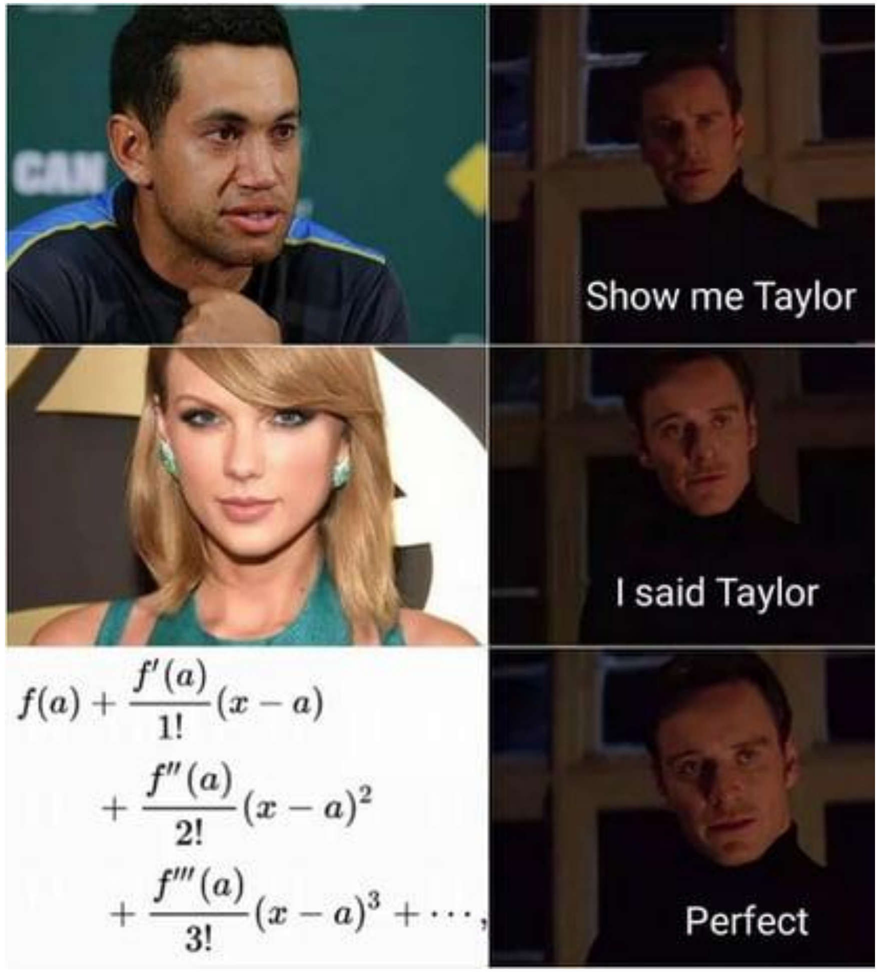
Introducción a Métodos de Machine Learning

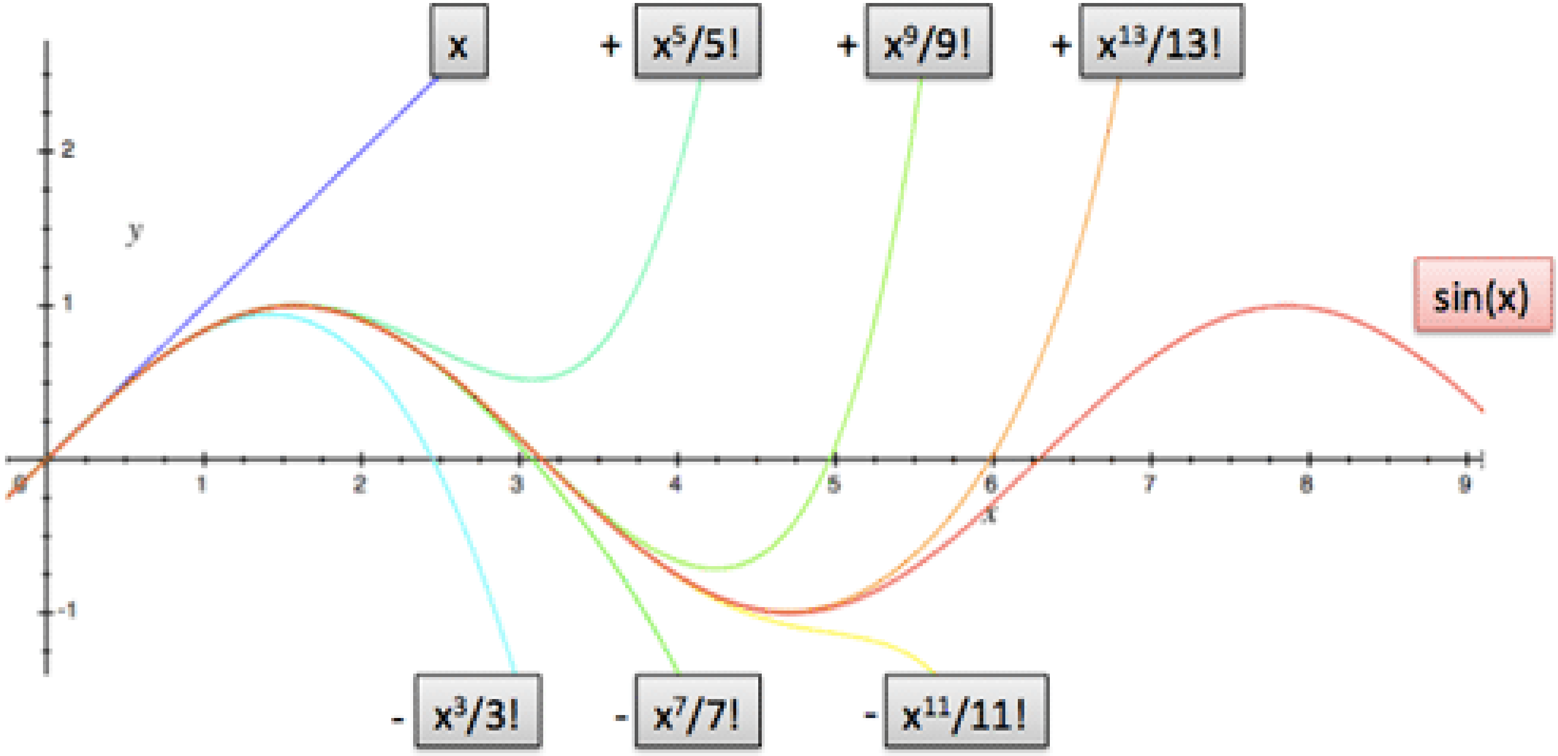
2023

Taylor Series



Better Models of Sine





Programming Taylor's Series

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$f(x) = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \mathcal{O}(x^4)$$

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```python
def graph(formula, x_range):
    x = np.array(x_range)
    y = formula(x)
    plt.plot(x, y)
```

Programming Taylor's Series

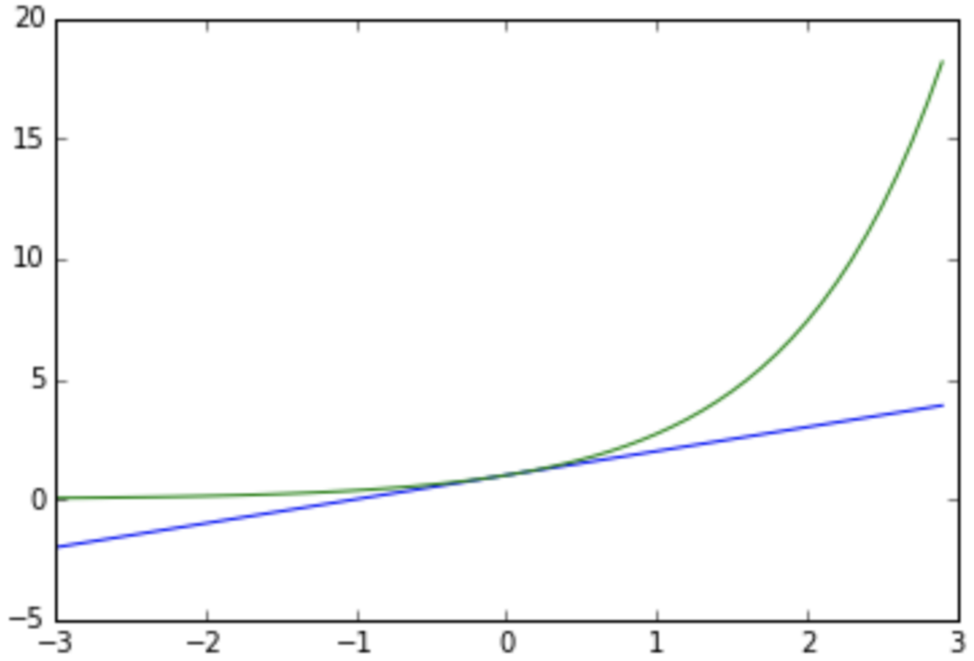
$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$f(x) = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \mathcal{O}(x^4)$$

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
def graph(formula, x_range):
    x = np.array(x_range)
    y = formula(x)
    plt.plot(x, y)
```

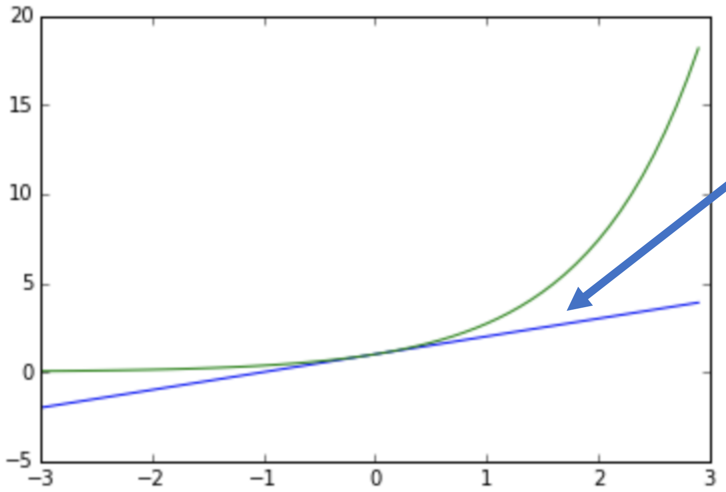
```
graph(lambda x: 1 + x, np.arange(-3,3,0.1))
graph(lambda x: np.exp(x), np.arange(-3,3,0.1))
```



Programming Taylor’s Series

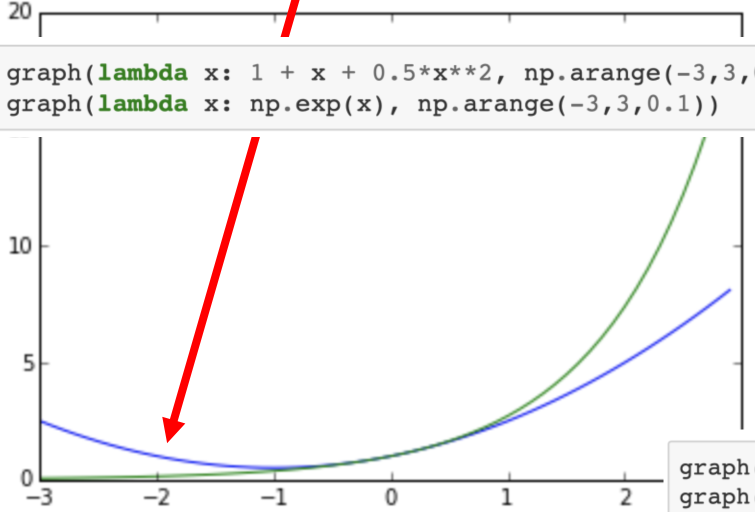
$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

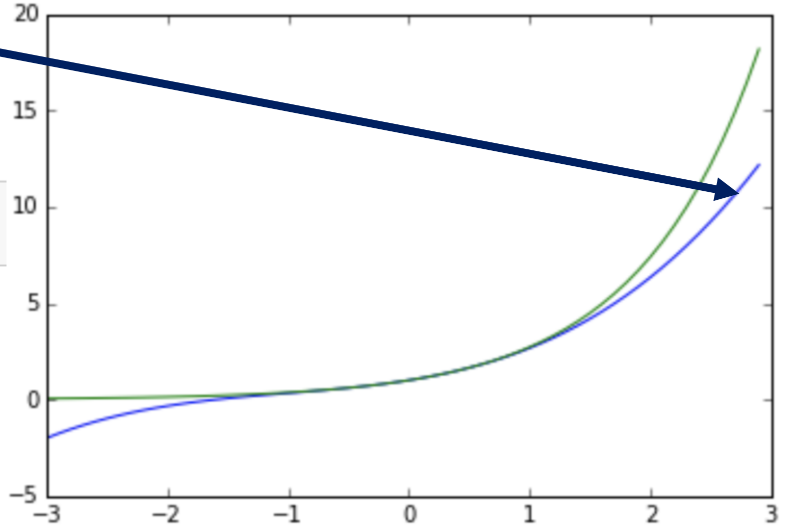
$$f(x) = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \mathcal{O}(x^4)$$



```
graph(lambda x: 1 + x, np.arange(-3,3,0.1))
graph(lambda x: np.exp(x), np.arange(-3,3,0.1))
```

```
graph(lambda x: 1 + x + 0.5*x**2, np.arange(-3,3,0.1))
graph(lambda x: np.exp(x), np.arange(-3,3,0.1))
```



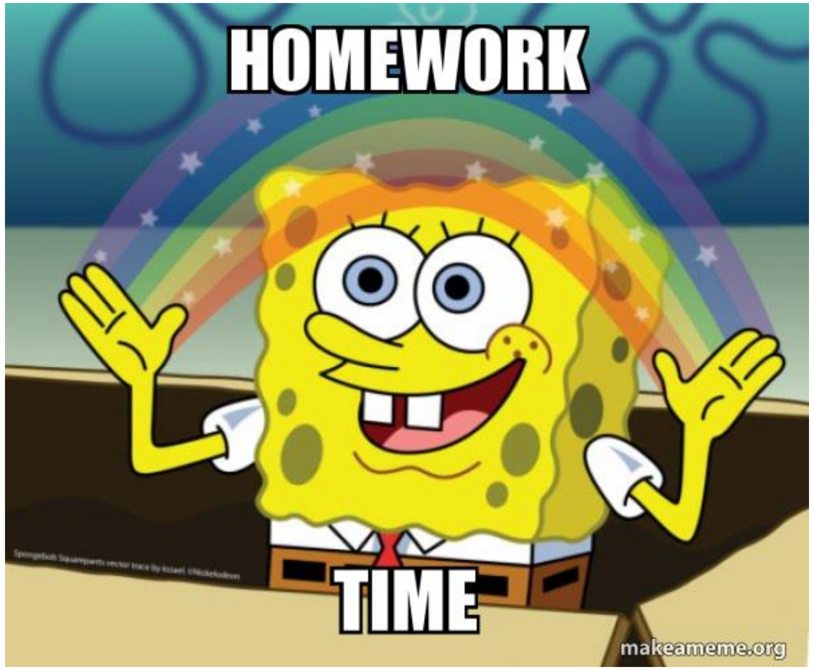


```
graph(lambda x: 1 + x + 0.5*x**2 + (1./6)*x**3, np.arange(-3,3,0.1))
graph(lambda x: np.exp(x), np.arange(-3,3,0.1))
```

Tarea:

Parte 1:

- Intente agregar usted mismo términos de orden superior.
- Cambie la función y calcule su serie de Taylor mediante la fórmula anterior, trace el resultado para diferentes aproximaciones de orden.



Newton's method

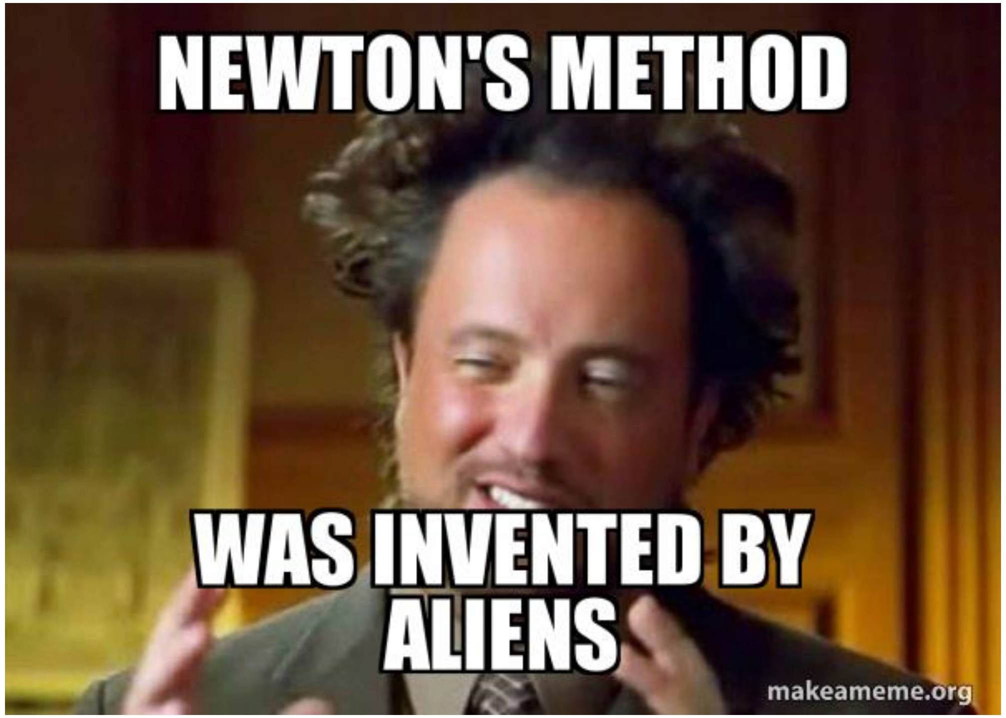
$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Isaac Newton developed a method to exploit the above in order to find roots of a function

That is, for a function f(x), find the value of x such that f(x)=0. Consider a first order expansion of a function:

$$y = f(x_n) + f'(x_n)(x - x_n)$$

Set this equal to zero and solve for the next point in terms of previous points:



$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

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

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

Newton's method

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

$$y = f(x_n) + f'(x_n)(x - x_n)$$

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

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

This gives an iterative method for finding roots. Write a function that solves for the roots of an equation

```
def dx(f, x):
    return abs(0-f(x))

def newtons_method(f, df, x0, epsilon):
    delta = dx(f, x0)
    while delta > epsilon:
        x0 = x0 - f(x0)/df(x0)
        delta = dx(f, x0)
    print 'Root at: ', x0
    print 'f(x) at root: ', f(x0)
    return delta
```

Newton's method

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

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

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

To use Newton's method (for finding roots) we need to know the function **f(x)** and its derivative **f'(x).** Let's test it out on the following polynomial

Example: $$f(x) = 6x^5 - 5x^4 - 4x^3 + 3x^2$$

$$f'(x) = 30x^4 - 20x^3 - 12x^2 + 6x$$

```
def f(x):
    return 6*x**5-5*x**4-4*x**3+3*x**2

def df(x):
    return 30*x**4-20*x**3-12*x**2+6*x
```

```
def dx(f, x):
    return abs(0-f(x))

def newtons_method(f, df, x0, epsilon):
    delta = dx(f, x0)
    while delta > epsilon:
        x0 = x0 - f(x0)/df(x0)
        delta = dx(f, x0)
    print 'Root at: ', x0
    print 'f(x) at root: ', f(x0)
    return delta
```

```
graph(lambda x: 6*x**5-5*x**4-4*x**3+3*x**2, np.arange(-1.5,2.,0.1))
```

Newton's method

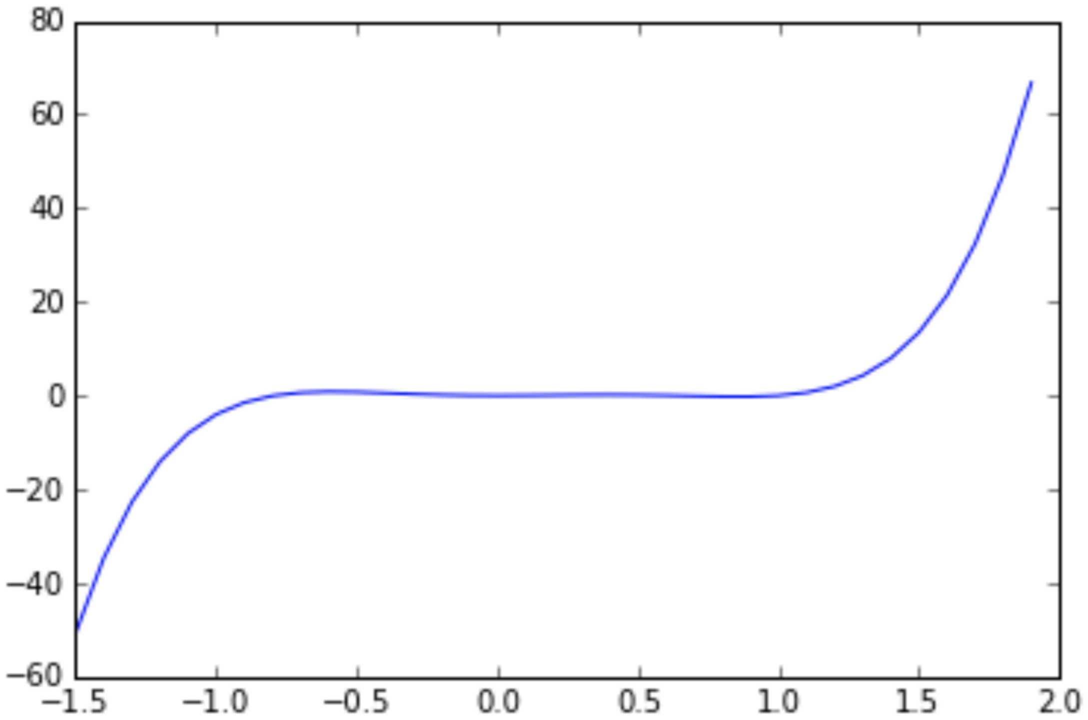
$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$
$$f'(x_n)x_{n+1} = f'(x_n)x_n - f(x_n)$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

To use Newton's method (for finding roots) we need to know the function **f(x)** and its derivative **f'(x).** Let's test it out on the following polynomial

Example: $$f(x) = 6x^5 - 5x^4 - 4x^3 + 3x^2$$
$$f'(x) = 30x^4 - 20x^3 - 12x^2 + 6x$$

```
def f(x):
    return 6*x**5-5*x**4-4*x**3+3*x**2

def df(x):
    return 30*x**4-20*x**3-12*x**2+6*x
```




```
graph(lambda x: 6*x**5-5*x**4-4*x**3+3*x**2, np.arange(-1.5,2.,0.1))
```

Newton's method

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$
$$f'(x_n)x_{n+1} = f'(x_n)x_n - f(x_n)$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

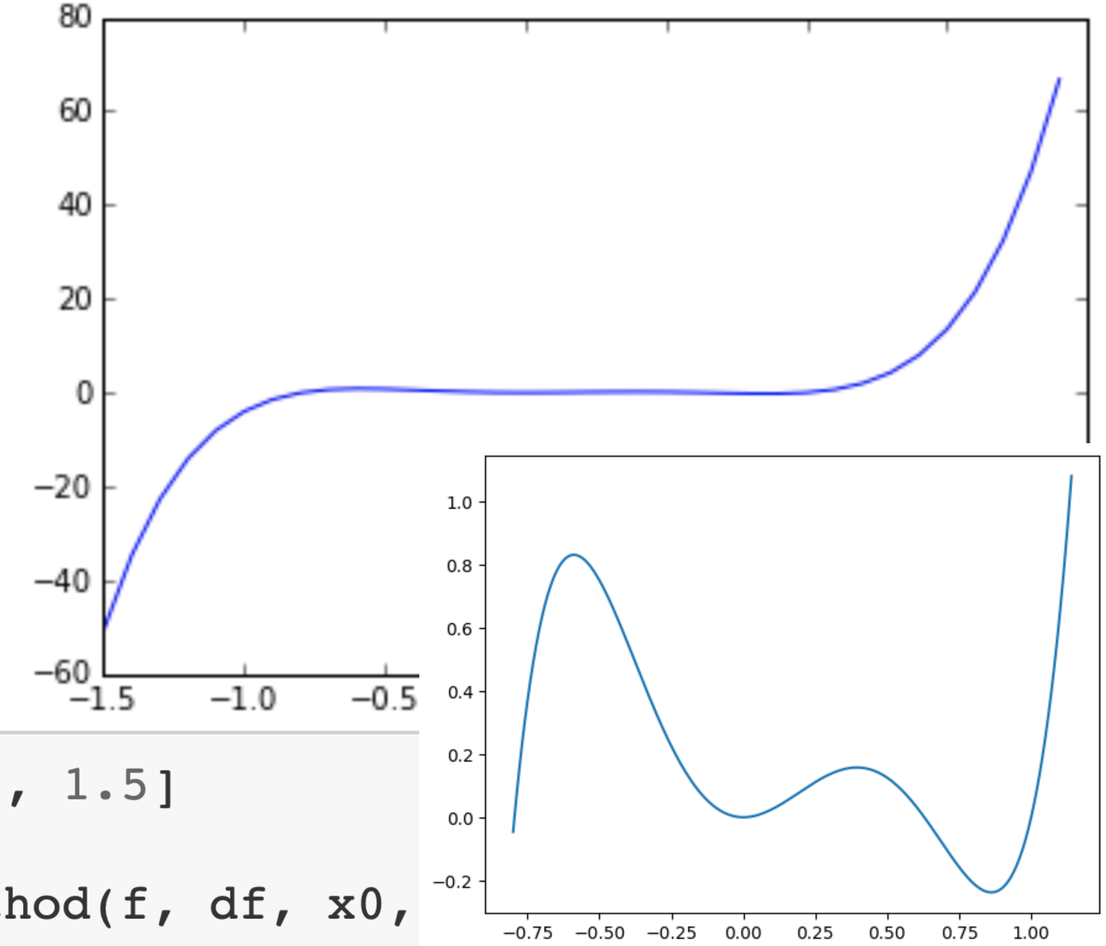
To use Newton's method (for finding roots) we need to know the function **f(x)** and its derivative **f'(x).** Let's test it out on the following polynomial

Example: $$f(x) = 6x^5 - 5x^4 - 4x^3 + 3x^2$$

$$f'(x) = 30x^4 - 20x^3 - 12x^2 + 6x$$

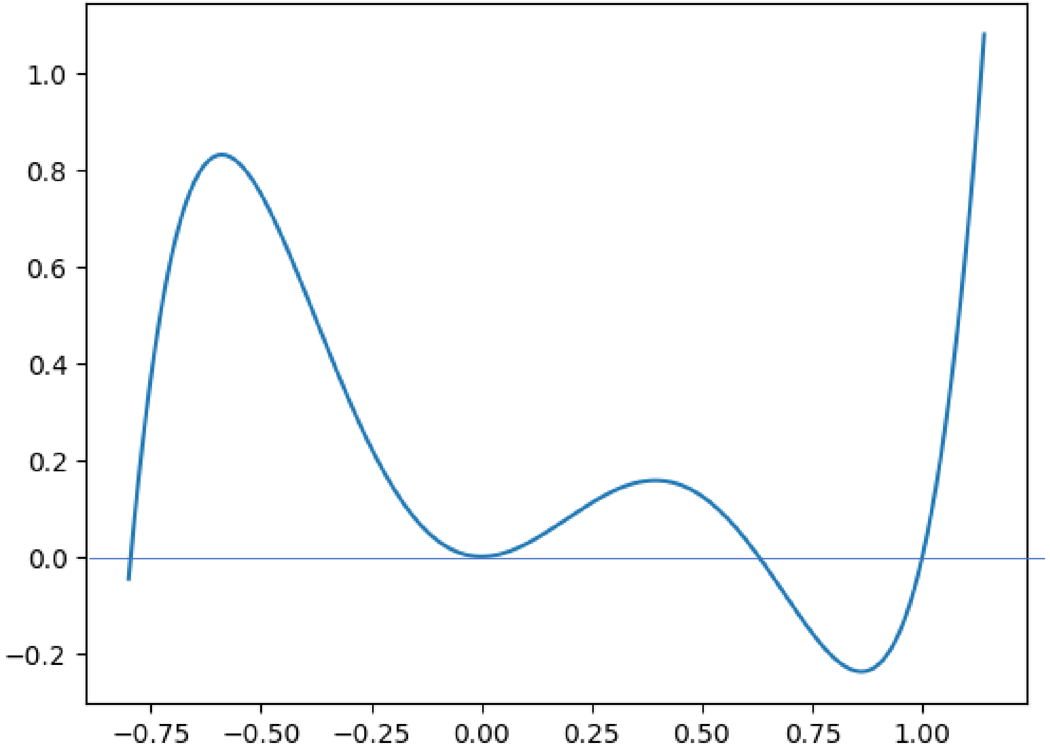
```
def f(x):
    return 6*x**5-5*x**4-4*x**3+3*x**2

def df(x):
    return 30*x**4-20*x**3-12*x**2+6*x
```



We can now find the root for any initial point we are interested in:

```
x0s = [0, .5, 1, 1.5]
for x0 in x0s:
    newtons_method(f, df, x0,
```




```
x0s = [0., .5, 1, 1.5]
for x0 in x0s:
    newtons_method(f, df, x0, 1e-5)
```

```
x0: 0.0
Root at:  0.0
f(x) at root:  0.0
x0: 0.5
Root at:  0.6286680781673306
f(x) at root:  -1.3785387997788945e-06
x0: 1
Root at:  1
f(x) at root:  0
x0: 1.5
Root at:  1.0000000000540352
f(x) at root:  2.1614132705849443e-10
```

```
x0s = [0.25, -.65, 0.9, 1.5]
for x0 in x0s:
    newtons_method(f, df, x0, 1e-5)
```

```
x0: 0.25
Root at:  0.001431336200813353
f(x) at root:  6.1344193615743226e-06
x0: -0.65
Root at:  -0.7953336554934082
f(x) at root:  -9.75177139039829e-08
x0: 0.9
Root at:  1.0000000023517583
f(x) at root:  9.407033374486673e-09
x0: 1.5
Root at:  1.0000000000540352
f(x) at root:  2.1614132705849443e-10
```

Newton's method

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n)$$

$$y = f(x_n) + f'(x_n)(x - x_n)$$

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

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

This gives an iterative method for finding roots. Write a function that solves for the roots of an equation

We can also use Newton's method for finding square roots or evaluating functions. Suppose we want to numerically evaluate √ 612. Define a function:

$$f(x) = x^2 - 612$$

$$f'(x) = 2x$$

Tarea:

Parte 1:

- Intente agregar usted mismo términos de orden superior.
- Cambie la función y calcule su serie de Taylor mediante la fórmula anterior, trace el resultado para diferentes aproximaciones de orden.

Parte 2:

- Llame al procedimiento para generar una aproximación a √ 612 resolviendo las raíces de f(x) .
- Resuelva para las raíces de las funciones:

$$f(x) = cos(x) - x^3$$
$$f'(x) = -sin(s) - 3x^2$$

¿Cuál es un buen valor inicial? El coseno está limitado por [0,1], así que pruebe con un valor de x0 de 0,5.

Newton Rhapson Method for Optimization

We can write Taylor's theorem for functions of **more than one variable**.

For example, let's considier a vector $\mathbf{x} \in \mathbb{R}^d$.

We can state Taylor's theorem in n dimensions as follows:

$$f(\mathbf{x}) = f(\mathbf{a}) + \sum_{k=1}^{d} \frac{\partial f}{\partial x_k}(\mathbf{a})(x_k - a_k) + \frac{1}{2} \sum_{j,k=1}^{d} \frac{\partial^2 f}{\partial x_j \partial x_k}(\mathbf{b})(x_j - a_j)(x_k - a_k) + \mathcal{O}(n^3)$$

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

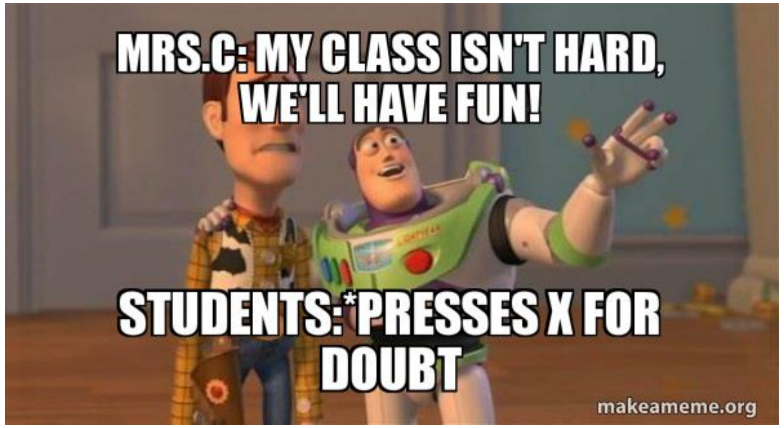
Newton Rhapson Method for Optimization

$$f(\mathbf{x}) = f(\mathbf{a}) + \sum_{k=1}^{d} \frac{\partial f}{\partial x_k}(\mathbf{a})(x_k - a_k) + \frac{1}{2} \sum_{j,k=1}^{d} \frac{\partial^2 f}{\partial x_j \partial x_k}(\mathbf{b})(x_j - a_j)(x_k - a_k) + \mathcal{O}(n^3)$$

Now consider a second order approximation of a function.
We'll consider multivariate functions and consider the gradient $\nabla f(x_n)$ and hessian $H(x_n)$

$$f(x_n) \approx f(x_n) + \nabla f(x_n)(x_{n+1} - x_n) + \frac{1}{2}(x_{n+1} - x_n)^T H(x_n)(x_{n+1} - x_n)$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$



Newton Rhapson Method for Optimization

$$f(x_n) \approx f(x_n) + \nabla f(x_n)(x_{n+1} - x_n) + \frac{1}{2}(x_{n+1} - x_n)^T H(x_n)(x_{n+1} - x_n)$$

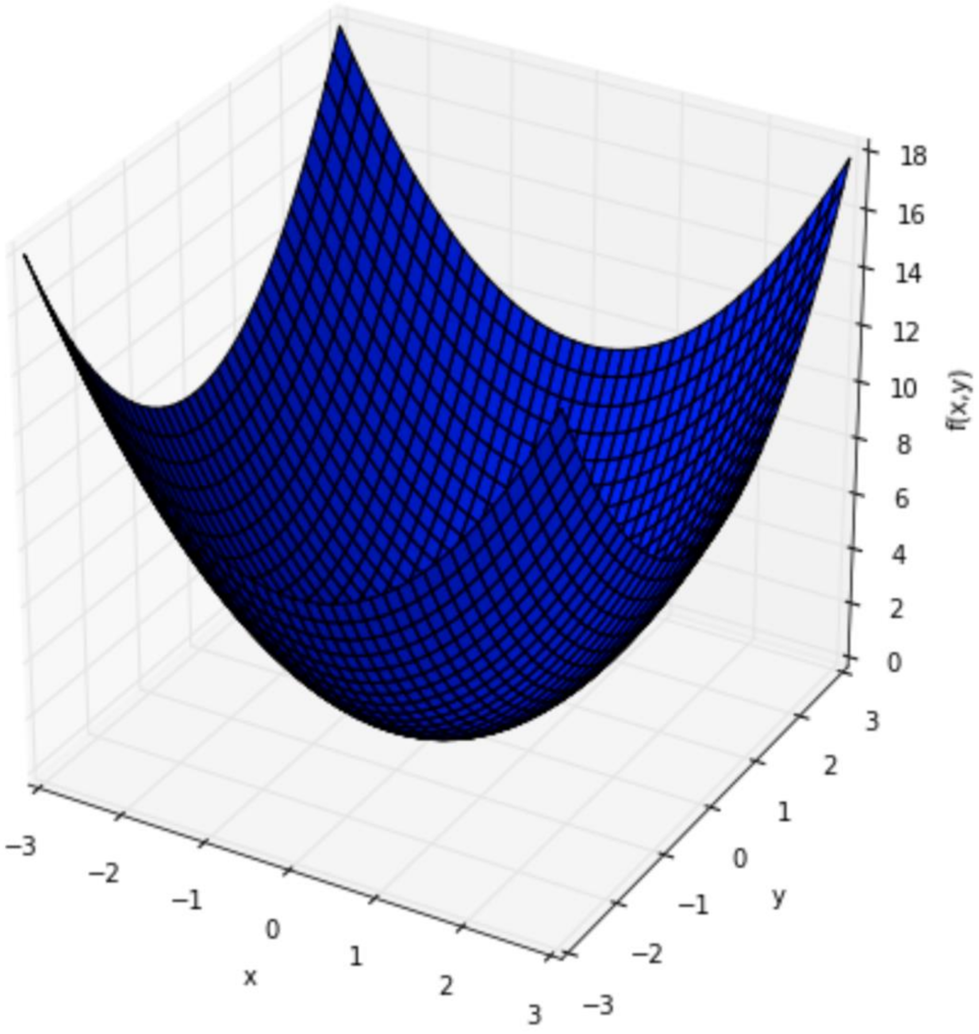
$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

We want to minimize this quadratic function. What value of x_n+1 minimizes the second order approximation on the RHS? Set this equal to zero, expand terms to simplify and solve for x_n+1.
When $H(x_n)$ is positive definite, then our Newton iteration should be:

$$x_{n+1} = x_n - H(x_n)^{-1} \nabla f(x_n)$$

adding a step size η:

$$x_{n+1} = x_n - \eta \times H(x_n)^{-1} \nabla f(x_n)$$


18
16
14
12
10
8
6
4
2
0
f(x,y)
−3
−2
−1
0
1
2
3
x
3
2
1
0
−1
−2
−3
y