Week 4:
Constrained and Numerical Optimisation

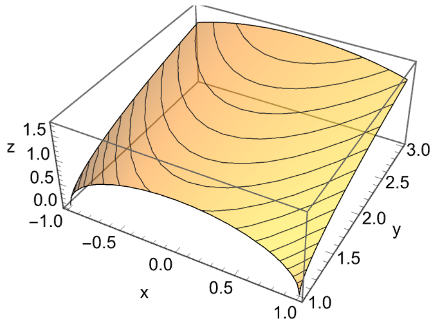
MSIN00180 Quantitative Methods for Business

Topics

- Lagrange Multipliers
- Estimating change in a specific direction
- Linearization
- Taylor’s Series
- Gradient Descent Algorithm
- Newton’s Method

Lagrange Multipliers

Consider the function $f(x, y) = \sqrt{y - x^2}$

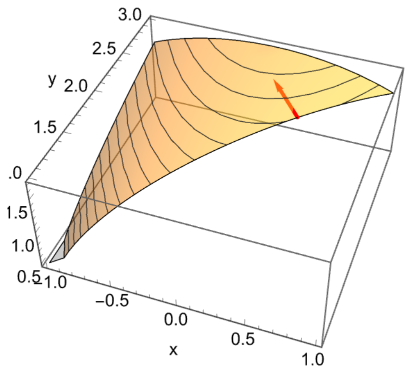


We now want to find the maximum of this function along a particular "cut", along the line $y = x + 2$, representing a constraint on the values of x and y.

It can be observed that the **gradient vector** of the function f is **normal** to the line y=x+2 at the maximum point of f along this line ...

Out[1022]=

```
x= 0.405  y= 2.405
```



Lagrange Multipliers

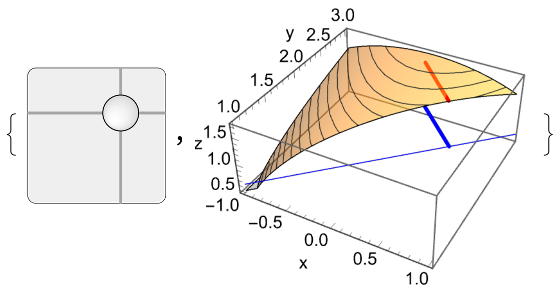
We can re-express the **constraint** line $y = x + 2$ as a new function $g(x, y) = 0$ thus

$g(x, y) = y - x - 2 = 0$

with a gradient vector of: $\nabla g = g_x\, i + g_y\, j \ = -1\, i + 1\, j$

Observe that **∇f and ∇g are parallel at the extreme value** of $f(x, y)$ when constrained by $g(x, y) = 0$

Out[ ]=



For two vectors to be parallel one must be a scalar multiple of the other so

$\nabla f = \lambda \nabla g$, where λ (“lambda”) is the scalar multiplier.

This is the basis of the Lagrange Multiplier method for finding constrained extreme values.

Lagrange Multiplier Method

Suppose that $f(x, y)$ and $g(x, y)$ are differentiable and $\nabla g \neq 0$ when $g(x, y) = 0$.

To find the local maximum and minimum values of f subject to the constraint $g(x, y) = 0$ (if these exist), find the values x, y, z, and λ that satisfy the equations

$$\nabla f = \lambda \nabla g \quad \text{and} \quad g(x, y) = 0.$$

For functions of more than two independent variables, add extra variables as needed.

Example 1

Find the extreme value of $f(x, y) = \sqrt{y - x^2}$ when constrained by $g(x, y) = y - x - 2 = 0$.

$$\nabla \boldsymbol{f} = \{f_x, f_y\} = \left\{\frac{-2x}{\sqrt{y-x^2}}, \frac{1}{\sqrt{y-x^2}}\right\}, \qquad \nabla \boldsymbol{g} = \{g_x, g_y\} = \{-1, 1\}$$

So the Lagrange Multiplier $\nabla \boldsymbol{f} = \boldsymbol{\lambda} \nabla \boldsymbol{g}$ gives us 2 equations

$f_x = \lambda g_x \quad \longrightarrow \quad \frac{-x}{\sqrt{y-x^2}} = -1\,\lambda \qquad$ ...(1)

$f_y = \lambda g_y \quad \longrightarrow \quad \frac{1}{2\sqrt{y-x^2}} = 1\,\lambda \qquad$ ...(2)

Substituting (2) into (1) gives
$-2\,x\,\lambda = -1\,\lambda \quad \longrightarrow \quad x = \frac{1}{2}$

Substitute this value for x into $g(x, y) = y - x - 2 = 0$ gives
$y = \frac{1}{2} + 2 \quad \longrightarrow \quad y = \frac{5}{2}$

The extreme value of f at this point is: $f\left(\frac{1}{2}, \frac{5}{2}\right) = \sqrt{\frac{5}{2} - \left(\frac{1}{2}\right)^2} = \frac{3}{2}$

Mathematica Maximise Function

Mathematica’s **Maximize** function can be used to solve constrained functions as per the current example:

In[1023]:=

```
Maximize[{√(y-x^2),y==x+2},{x,y}]
```

$$\text{Maximize}\left[\left\{\sqrt{y-x^2},\ y==x+2\right\},\{x,y\}\right]$$

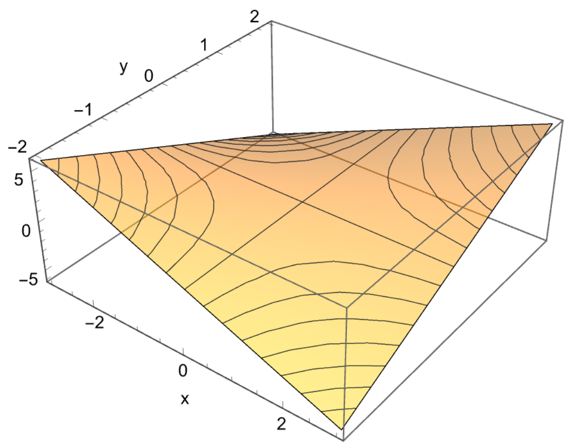
Out[1023]=

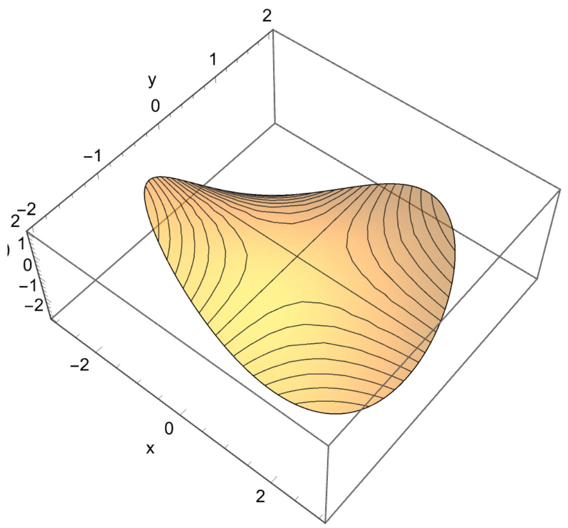
$$\left\{\frac{3}{2},\ \left\{x\to\frac{1}{2},\ y\to\frac{5}{2}\right\}\right\}$$

Example 2

Find the extreme values of the function $f(x, y) = x\, y$ when constrained to the boundary line $\frac{x^2}{8} + \frac{y^2}{2} = 1$

Out[▫]=





Example 2

Express the constraint in the form $g(x, y) = 0$

$g(x, y) = \frac{x^2}{8} + \frac{y^2}{2} - 1 = 0$ ...(1)

Find ∇f and ∇g

$\nabla f = \{f_x, f_y\} = \{y, x\}$, $\nabla g = \{g_x, g_y\} = \left\{\frac{x}{4}, y\right\}$

Apply the Lagrange multiplier $\nabla f = \lambda \nabla g$

$y = \lambda \frac{x}{4}$ ...(2)

$x = \lambda y$ ...(3)

Sub (3) into (2): $y = \lambda \frac{\lambda y}{4} \longrightarrow \lambda = \pm 2$ ...(4)

Sub (4) into (3) gives: $x = \pm 2\, y$ ...(5)

Sub (5) into (1) gives

$\frac{(\pm 2\, y)^2}{8} + \frac{y^2}{2} - 1 = 0 \longrightarrow y^2 = 1 \longrightarrow y = \pm 1$

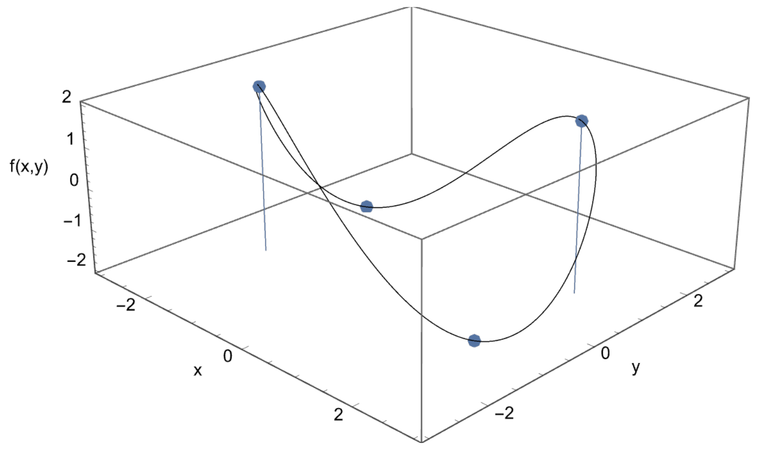
Sub this value for y into (5) gives $x = \pm 2$

Example 2

This implies 4 extreme points at: (2,1), (2,-1), (-2,1) and (-2,-1)

The extreme points (2,1) and (-2, -1) are maxima with values of $f(x, y) = x\,y\; = \;2$ and the extreme points (-2,1) and (2, -1) are minima with values of $f(x, y) = \; -2$.

Out[ ]=



Lagrange Multipliers with Two Constraints

Some problems require us to find the extreme values of $f(x, y, z)$ whose variables are subject to two constraints:

$g_1(x, y) = 0$ and $g_2(x, y) = 0$

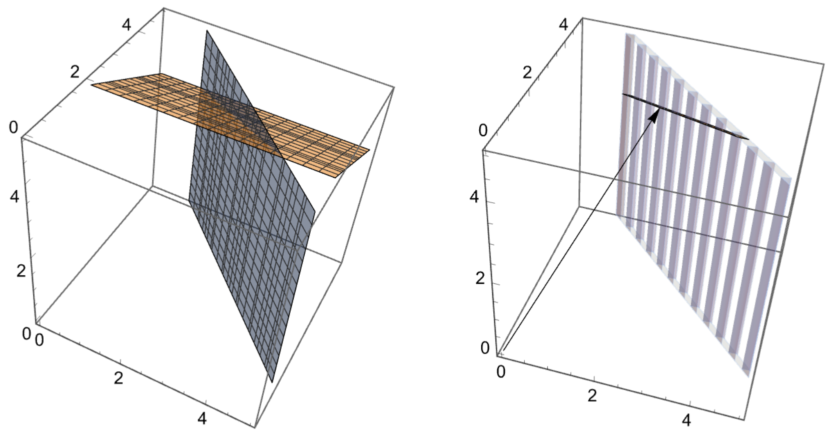
Provided g_1 and g_2 are differentiable, with ∇g_1 not parallel to ∇g_2 the extreme values are found by solving

$\boldsymbol{\nabla f = \lambda \nabla g_1 + \mu \nabla g_2}$ and $\boldsymbol{g_1(x, y) = 0}$ and $\boldsymbol{g_2(x, y) = 0}$

Two Constraints Example

Find the minimum distance from the origin (0,0,0) to the points that both lie on the plane $y + 2z - 12 = 0$ and by the plane $x + y - 6 = 0$

Out[ ]=



Two Constraints Example

The distance from the origin (0,0,0) to a point (x,y,z) is given by $\sqrt{x^2+y^2+z^2}$.

So the **objective function** (the function being minimized) $f(x, y, z) = \sqrt{x^2+y^2+z^2}$

The solution is given by the Lagrange Multiplier equation

$\nabla f = \lambda \nabla g_1 + \mu \nabla g_2$

where the constraint functions, $g_1(x, y, z) = y + 2z - 12 = 0$, and $g_2(x, y, z) = x + y - 6 = 0$.

$\nabla f = \left\{\frac{2x}{\sqrt{x^2+y^2+z^2}}, \frac{2y}{\sqrt{x^2+y^2+z^2}}, \frac{2z}{\sqrt{x^2+y^2+z^2}}\right\}$

$\nabla g_1 = \{0, 1, 2\}$

$\nabla g_2 = \{1, 1, 0\}$

Two Constraints Example

The Lagrange multiplier equation gives the following 3 equations:

$$\frac{2x}{\sqrt{x^2+y^2+z^2}} = \lambda.0 + \mu.1 \quad \longrightarrow \quad \mu = \frac{2x}{\sqrt{x^2+y^2+z^2}}$$

$$\frac{2y}{\sqrt{x^2+y^2+z^2}} = \lambda.1 + \mu.1 \quad \longrightarrow \quad \lambda = \frac{2y}{\sqrt{x^2+y^2+z^2}} - \frac{2x}{\sqrt{x^2+y^2+z^2}}$$

$$\frac{2z}{\sqrt{x^2+y^2+z^2}} = \lambda.2 + \mu.0 \quad \longrightarrow \quad \frac{z}{\sqrt{x^2+y^2+z^2}} = \frac{2y}{\sqrt{x^2+y^2+z^2}} - \frac{2x}{\sqrt{x^2+y^2+z^2}}$$

$$\Longrightarrow \quad z = 2y - 2x \text{ ...(1)}$$

The original constraint equations are:

$y + 2z - 12 = 0$ ...(2), $\qquad x = 6 - y$ ...(3)

Sub (1) into (2): $y + 2(2y - 2x) - 12 = 0 \quad \longrightarrow \quad -4x + 5y = 12$ ...(4)

Sub (3) into (4): $-4(6 - y) + 5y = 12 \quad \longrightarrow \quad \mathbf{y = 4}$

Sub y into (3): $\quad \longrightarrow \quad \mathbf{x = 2}$

Sub x, y into (1): $\quad \longrightarrow \quad z = 2y - 2x = 8 - 4 \quad \longrightarrow \quad \mathbf{z = 4}$

The closest point is (2,4,4) where the distance from the origin is $\sqrt{2^2 + 4^2 + 4^2} = 6$

Estimating Change in a Specific Direction

To estimate the change in the value of a differentiable function f when we move a small distance ds from a point P_0 in a particular direction $\boldsymbol{u}$, use the formula

$df =$ (Directional Derivative at P_0) $\times\ ds$

$$df = \left(\nabla f\ |_{P_0} \cdot \boldsymbol{u}\right)\ ds$$

where **u** is a **unit vector**.

When **u** is not a unit vector use $df = \left(\nabla f\ |_{P_0} \cdot \frac{\boldsymbol{u}}{|\boldsymbol{u}|}\right)\ ds$

Example

Monthly profits (p) can be expressed as a function of the **average cost of producing 1 item** (c), the **quantity sold in one month** (q), the **sales price mark-up** (m), and some **fixed monthly cost** (F) which is assumed constant.

profits = revenues - costs

$p(q, c, m) = q\,c\,(1 + m) - q\,c - F \quad = q\,c\,m \; - F$

Suppose the business currently is operating with the following values

$q_0 = 100, \quad c_0 = 20, \quad m_0 = 0.1$

Now suppose the business believes that there is a **relationship between mark-up and the quantity sold** such that increasing mark-up by 1% (+0.01) decreases sales by 1 unit (-1). The business wants to understand the impact on its current monthly profits by moving in direction of increasing its mark-up by 5% (and therefore reducing quantity by 5).

Example

Calculate the gradient vector for p

$p(q, c, m) = q\,c\,m - F$

$\nabla p = \{p_q, p_c, p_m\} = \{c\,m,\ q\,m,\ q\,c\}$

$\nabla p_{(100,20,0.1)} = \{2, 10, 2000\}$

The direction of change can be expressed as

$\boldsymbol{u} = \{-1, 0, 0.01\}$

i.e. q changes by -1, c does not change, and m changes by +0.01

Example

Estimated impact (change in profits) in moving in this direction is

$$
\begin{aligned}
\mathrm{dp} &= \nabla p \cdot \frac{u}{|u|}\,\mathrm{ds} \\
&= \{2, 10, 2000\} \cdot \frac{1}{|u|}\{-1, 0, 0.01\}\,\mathrm{ds} \\
&= \frac{1}{|u|}(-2 + 20)\,\mathrm{ds} \\
&= \frac{1}{|u|}\,18\,\mathrm{ds}
\end{aligned}
$$

(It is unnecessary to calculate $|u|$ as it will be cancelled out later.)

For a +5% change in mark-up

$\mathrm{ds} = 5\ |u|$

So the estimated increase in monthly profits is given by

$\mathrm{dp} = \frac{1}{|u|}\,18 \times 5 \times |u| = 90$

Solution in *Mathematica*

In[ ]:=
```
p[q_,c_,m_]:=q c m - F
∇p=Grad[p[q,c,m],{q,c,m}];
∇p0=∇p /. {q→ 100,c→ 20, m→ 0.1};
dp = ∇p0.{-1,0,0.01}ds /. ds→ 5 (* estimated change *)
```

Out[ ]=
```
90.
```

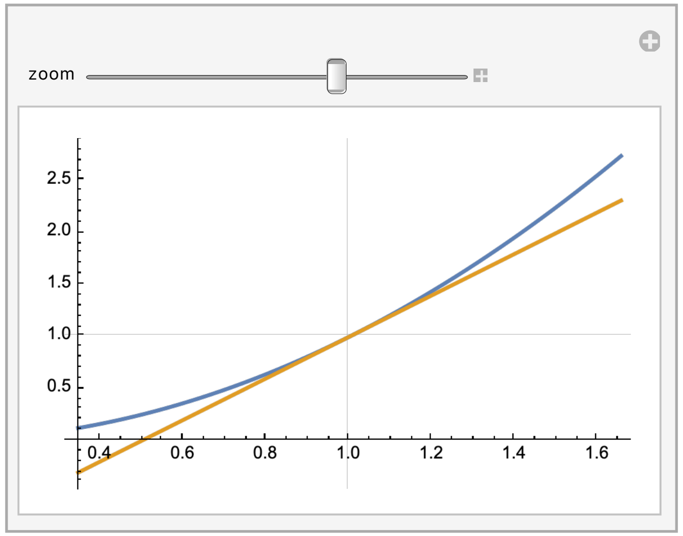
Linearization

The more you magnify the graph of a function at a given point, the more the graph resembles its tangent.

Accordingly, the equation of the tangent line at a point can be used to approximate the function close to that point. The equation of a tangent of the function $f(x)$ at $x = a$ is given by

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

Out[ ]=



Linearization

If f is differentiable at $x = a$, then the approximating function

$$L(x) = f(a) + f'(a)(x - a)$$

is the **linearization** of f at a.

Example

Find the linearization of $f(x) = \sqrt{1+x}$ at $x = 0$

Using the Chain Rule ... $f'(x) = \frac{1}{2}(1+x)^{\frac{-1}{2}}$

$\boldsymbol{L(x) = f(a) + f'(a)(x-a)}$ at $x = a$

$L(x) = \sqrt{1+a} + \frac{1}{2}(1+a)^{\frac{-1}{2}}(x-a)$ at $x = a$

$L(x) = \sqrt{1} + \frac{1}{2}(1)^{\frac{-1}{2}}(x)$ at $x = 0$

$L(x) = 1 + \frac{1}{2}x$

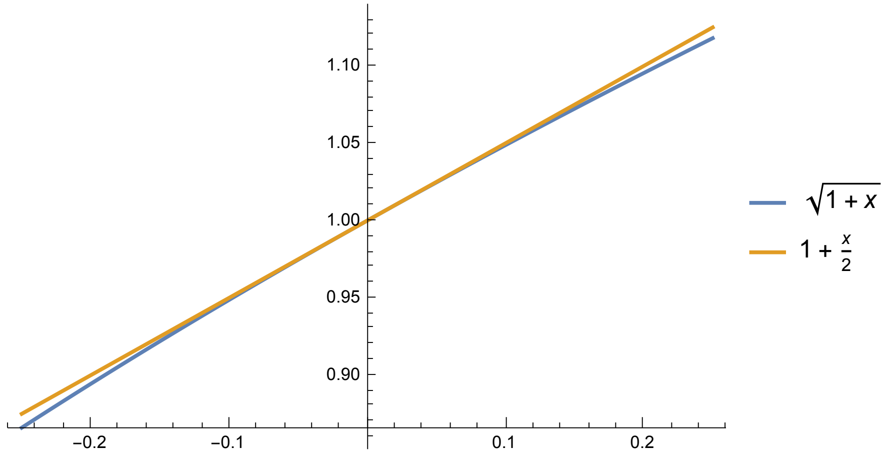
$\sqrt{1+x}$ is approximated by $\left(1 + \frac{1}{2}x\right)$ close to $x = 0$

Example

$\sqrt{1+x}$ is approximated by $\left(1+\frac{1}{2}x\right)$ close to $x=0$

In[ ]:=
```
Plot[{ √(1+x) ,1+(1/2)x},{x,-0.25,0.25}, PlotLegends→"Expressions"]
```

Out[ ]=



Linearization of a function of 2 variables

The linearization of a function $f(x, y)$ at a point (x_0, y_0) where f is differentiable is the function

$$L(x, y) = f(x_0, y_0) + f_x(x_0, y_0)\,(x - x_0) + f_y(x_0, y_0)\,(y - y_0)$$

The approximation $f(x, y) \approx L(x, y)$ is the standard linear approximation of f at (x_0, y_0).

Linearization and Gradient Vectors

The linearization can also be expressed using the gradient vector ∇f

$L(x, y) = f(x_0, y_0) + \nabla f_{(x_0,y_0)} \cdot \boldsymbol{u}$ where $\boldsymbol{u} = (x - x_0)\,\boldsymbol{i} + (y - y_0)\,\boldsymbol{j}$

This is equivalent to adding the **estimating change in a specific direction** (previous topic) to the initial value for f

$L(x, y) = f(x_0, y_0) + \mathrm{df}$ where $\mathrm{df} = \nabla f_{(x_0,y_0)} \cdot \frac{u}{|u|}\,\mathrm{ds}$ and $\mathrm{ds} = |u|$

Mathematica Example

```
p[q_,c_,m_]:=q c m - 50;
q0=100.;c0=20.;m0=0.1;
∇p=Grad[p[q,c,m],{q,c,m}];
∇p0=∇p /. {q→ 100,c→ 20, m→ 0.1};

(*Linearization formula*)
pL[q_,c_,m_]:=p[q0,c0,m0]+∇p0.{(q-q0),(c-c0),(m-m0)}

(*Shows the Linearization gives the same answer as the last slide example*)
pL[95,20,0.15]-p[100,20,0.1]
```

```
90.
```

Error in Linearization Estimates

It can be shown that the error $E(x, y)$ incurred in replacing $f(x, y)$ by its linearization

$$L(x, y) = f(x_0, y_0) + f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0)$$

satisfies the inequality

$$\left| E(x, y) \right| \leq \frac{M}{2}(|x - x_0| + |y - y_0|)^2$$

where

$$M = \max(|f_{xx}|, |f_{yy}|, |f_{xy}|)$$

Example

For the function $f(x, y) = x^2 + y^2$ find the upper bound percentage error of the linearization estimate $L(1.1, 1.1)$ relative to the known value of $f(1, 1)$.

$f_x = 2x, \quad f_{xx} = 2$

$f_y = 2y, \quad f_{yy} = 2$

$f_{xy} = 0$

So $M = \max(|f_{xx}|, |f_{yy}|, |f_{xy}|) = 2$

The error of the linearization estimate at (1.1,1.1) is given by
$|E(x, y)| \leqslant \frac{1}{2} M(|x - x_0| + |y - y_0|)^2$

So
$$\begin{aligned}|E(1.1, 1.1)| &\leqslant \tfrac{1}{2}(2)(|1.1 - 1| + |1.1 - 1|)^2 \\ &\leqslant 0.2^2 \\ &\leqslant 0.04\end{aligned}$$

Compare to real error found in *Mathematica*

In[ ]:=
```
f[x_,y_]:=x^2+y^2;
L[x_,y_,x0_,y0_]:=f[x0,y0]+2x(x-x0)+2y(y-y0);
Print["L(1.1,1.1) = ",L[1.1,1.1,1,1]];
Print["f(1.1,1.1) = ",f[1.1,1.1]];
Print["real error = ",Abs[f[1.1,1.1]-L[1.1,1.1,1,1]]];
```

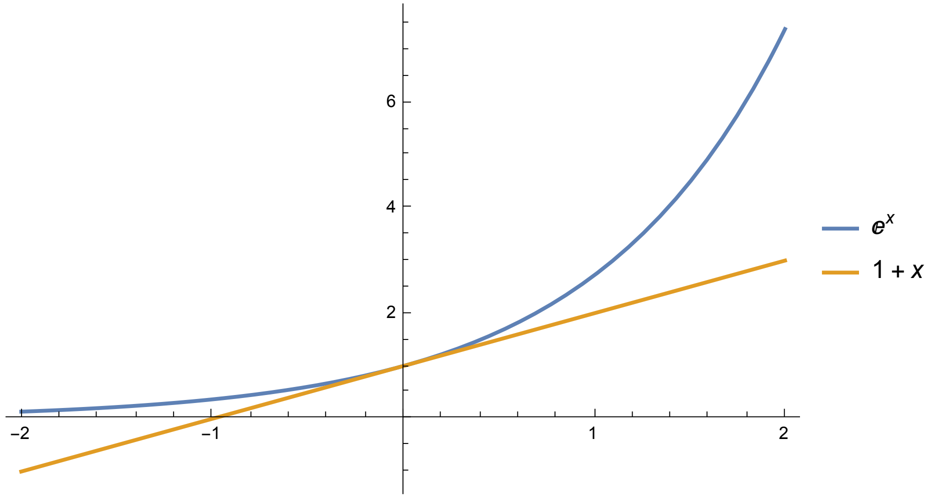
```
L(1.1,1.1) = 2.44

f(1.1,1.1) = 2.42

real error = 0.02
```

Polynomial Approximations

First consider the linearization of an example function like $f(x) = e^x$ about the point $x_0 = 0$.

$$L(x) = f(x_0) + f'(x_0)(x - x_0) = e^0 + e^0(x - 0) = 1 + x$$

Out[ ]=

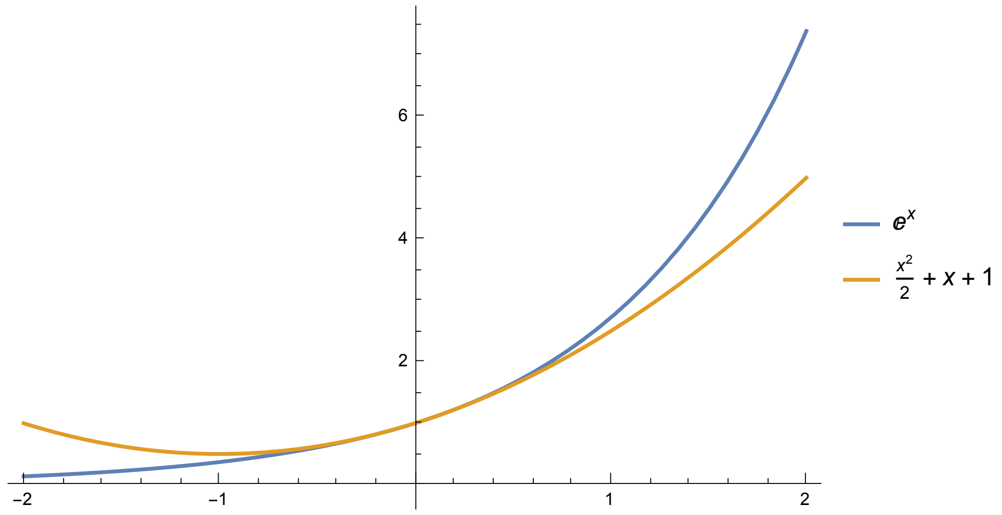


Polynomial Approximations

A second degree polynomial (i.e. quadratic) $g(x) = a\,x^2 + b\,x + c$ is a local approximation to $f(x)$ at the point $x_0 = 0$ when $g(x_0) = f(x_0)$, $g'(x_0) = f'(x_0)$, and $g''(x_0) = f''(x_0)$.

$g(x_0) = f(x_0) \quad \longrightarrow a\,x_0{}^2 + b\,x_0 + c = e^{x_0} \longrightarrow \; a\,0^2 + b\,0 + c = e^0(=1)$
$g'(x_0) = f'(x_0) \quad \longrightarrow 2\,a\,x_0 + b \;= e^{x_0} \longrightarrow 2\,a\,0 + b \;= 1$
$g''(x_0) = f''(x_0) \quad \longrightarrow 2\,a \;= e^{x_0} \longrightarrow 2\,a \;= 1$
$\Rightarrow\; a = \frac{1}{2},\; b = 1,\; c = 1$

Out[ ]=

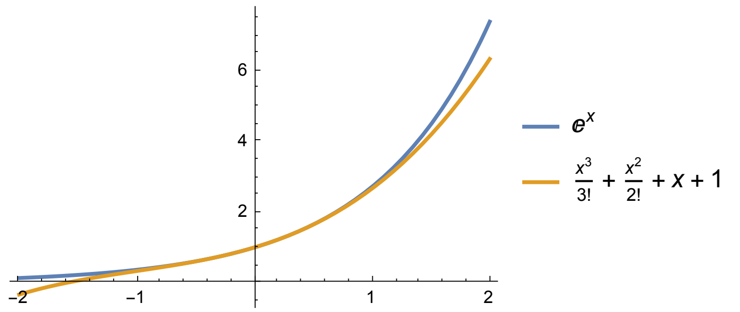


Higher Order Polynomial Approximations

Using a similar approach we can show that

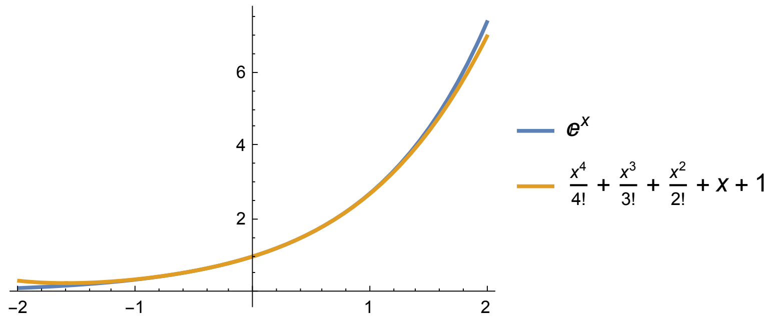
$e^x \approx h(x) = \frac{1}{3!}x^3 + \frac{1}{2!}x^2 + x + 1$

Out[ ]=



and so on

Out[ ]=



Taylor Series

It is possible to generalise this approach for any differentiable function $f(x)$ about a point $x = x_0$

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!} f''(x_0)(x - x_0)^2 + \frac{1}{3!} f'''(x_0)(x - x_0)^3 + \ldots$$

This process can be continued indefinitely as long as f can be differentiated often enough.

This series is known as the Taylor Series

$$f(x) \approx f(x_0) + \sum_{n=1}^{\infty} \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n$$

Maclaurin Series

Setting $x_0 = 0$ gives a special case of the Taylor Series called the Maclaurin Series

$$f(x) \approx f(0) + \sum_{n=1}^{\infty} \frac{1}{n!} f^{(n)}(0)\, x^n$$

Example

Find a second-order Maclaurin series polynomial approximation to the function $f(x) = \frac{1}{\sqrt{1-x^2}}$ and plot these for $x \in (-1, 1)$.

The second-order approximation is:

$$f(x) \approx f(0) + \sum_{n=1}^{2} \frac{1}{n!} f^{(n)}(0)\, x^n \quad = f(0) + f'(0)\, x + \frac{1}{2} f''(0)\, x^2$$

$$f(0) = 1$$

$$f'(x) = -\frac{1}{2}\left(\frac{1}{\sqrt{1-x^2}}\right)^{-\frac{3}{2}}(-2\,x) \qquad \Longrightarrow f'(0) = 0$$

$$f''(x) = \left(\frac{1}{\sqrt{1-x^2}}\right)^{-\frac{3}{2}}(1) + x\,\frac{d}{dx}\left(\left(\frac{1}{\sqrt{1-x^2}}\right)^{-\frac{3}{2}}\right) \Longrightarrow f''(0) = \left(\frac{1}{\sqrt{1-0^2}}\right)^{-\frac{3}{2}}(1) + 0 \times \ldots = 1$$

so

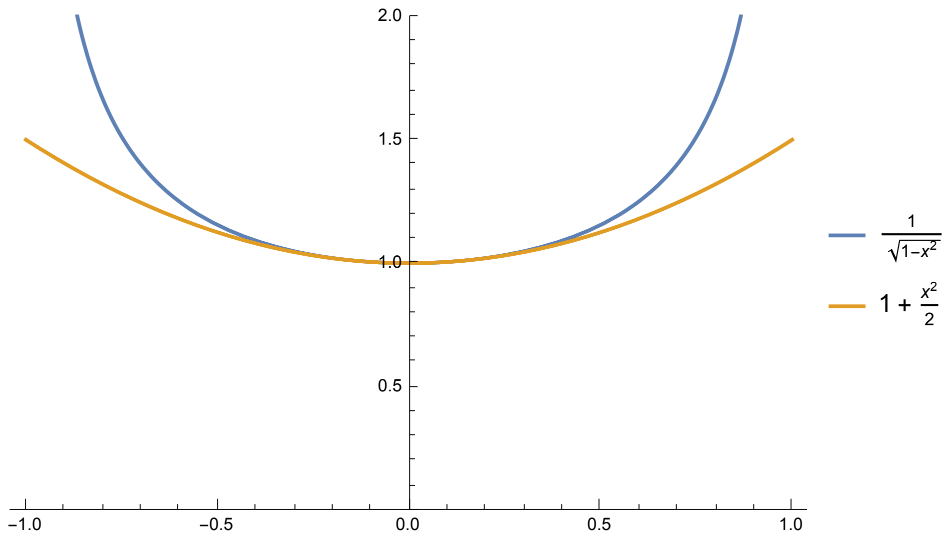
$$f(x) \approx f(0) + f'(0)\, x + \frac{1}{2} f''(0)\, x^2 \quad = 1 + \frac{1}{2} x^2$$

Example plotted in *Mathematica*

In[ ]:=
```
Plot[{1/Sqrt[1-x^2],1+1/2 x^2},{x,-1,1}, PlotRange→{0,2},PlotLegends→"Expressions"]
```

$$\texttt{Plot}\left[\left\{\frac{1}{\sqrt{1-x^2}},1+\frac{1}{2}x^2\right\},\{x,-1,1\},\ \texttt{PlotRange}\to\{0,2\},\texttt{PlotLegends}\to\texttt{"Expressions"}\right]$$

Out[ ]=



Maclaurin Series approximations using *Mathematica*

Series returns Taylor (and therefore Maclaurin) Series polynomials.

In[1026]:=

```
MacApprox2 = Series[1/Sqrt[1-x^2],{x,0,2}] //Normal
```

$$\text{MacApprox2} = \text{Series}\left[\frac{1}{\sqrt{1-x^2}},\{x,0,2\}\right] \text{//Normal}$$

Out[1026]=

$$1+\frac{x^2}{2}$$

Here we find the order-4 polynomial centred around *x* = 0.

In[ ]:=

$$\text{MacApprox4} = \text{Series}\left[\frac{1}{\sqrt{1-x^2}},\{x,0,4\}\right] \text{//Normal}$$

Out[ ]=

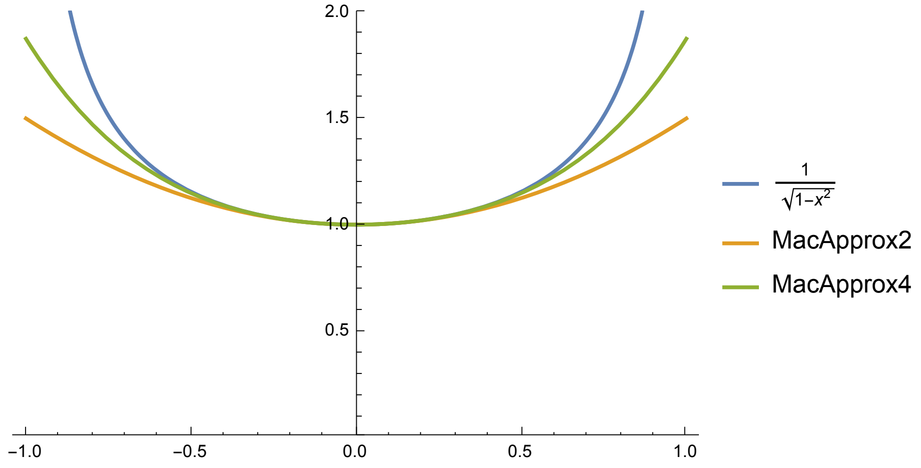
$$1+\frac{x^2}{2}+\frac{3\,x^4}{8}$$

Maclaurin Series approximations using *Mathematica*

Plot showing both the order-2 and order-4 approximation around $x = 0$.

In[ ]:=
```
Plot[{1/Sqrt[1-x^2],MacApprox2,MacApprox4},{x,-1,1}, PlotRange→{0,2},PlotLegends→"Expressions"]
```

Out[ ]=



Maclaurin Series approximations using *Mathematica*

In[46]:=
```
TaylorApprox2 = Series[1/√(1-x^2),{x,a,2}] //Normal//Simplify

taylor2[x1_,a1_]:=TaylorApprox2 /.{x→x1,a→a1}
```

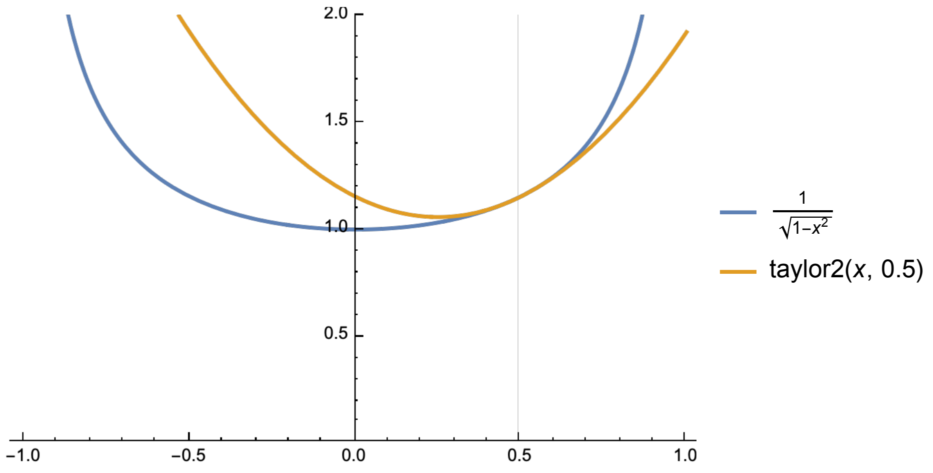
Out[46]=

$$\frac{2+6a^4-6a^3x+x^2+a^2\left(-5+2x^2\right)}{2\left(1-a^2\right)^{5/2}}$$

Here we plot the 2nd-order Taylor approximation around the point $x = a = 0.5$:

In[48]:=
```
Plot[{1/√(1-x^2),taylor2[x,0.5]},{x,-1,1},GridLines→{{0.5},None}, PlotRange→{0,2},PlotLegends→"Expressions"]
```

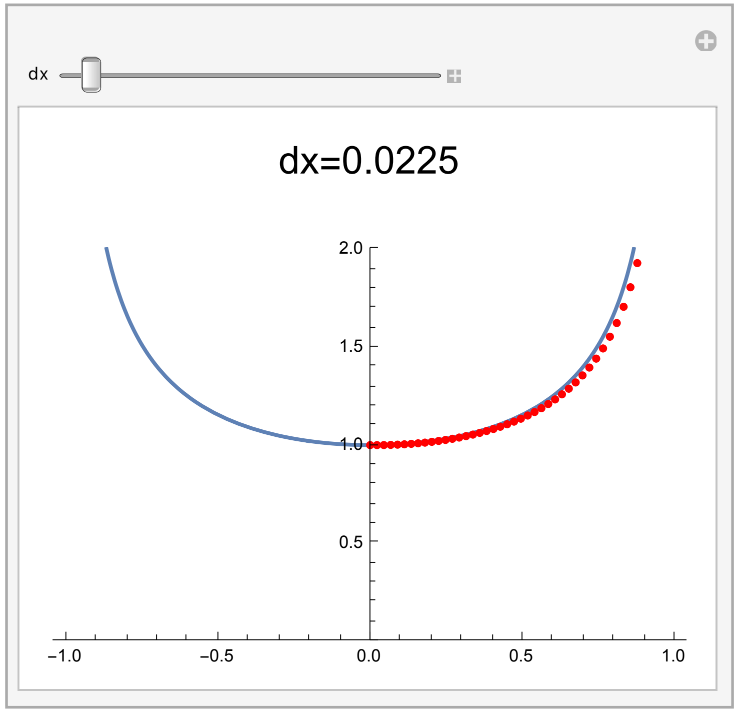
Out[48]=



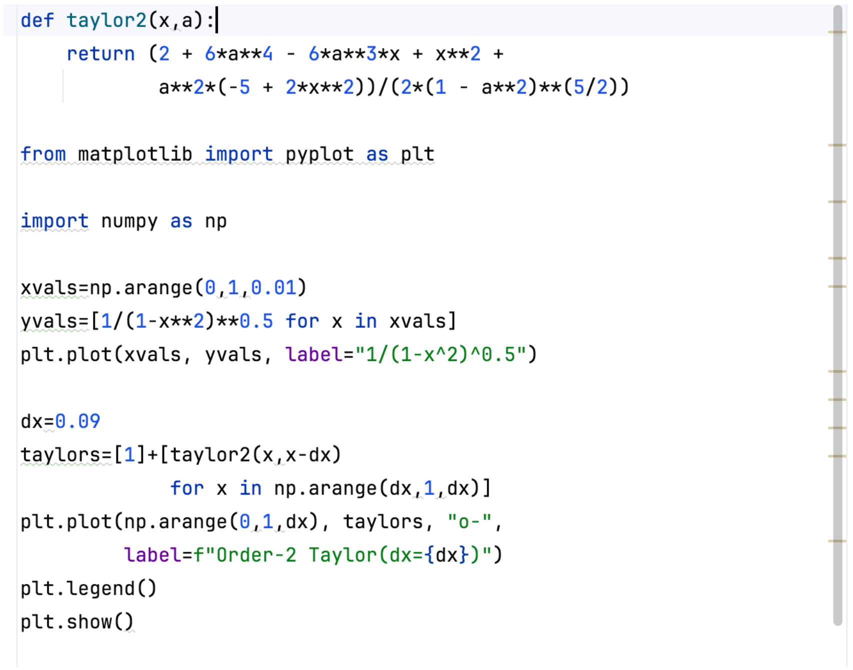
Repeated use of Taylor Series approximations

In practice, higher order Taylor polynomials may not be possible (or practical) to find.
In this case, low order polynomials (eg order 2) may be repeatedly applied over small increments of x, re-centreing the Taylor polynomial at each new increment.
This Mathematica dynamic plot shows this using the previously found order 2 polynomial for different increment values (dx).

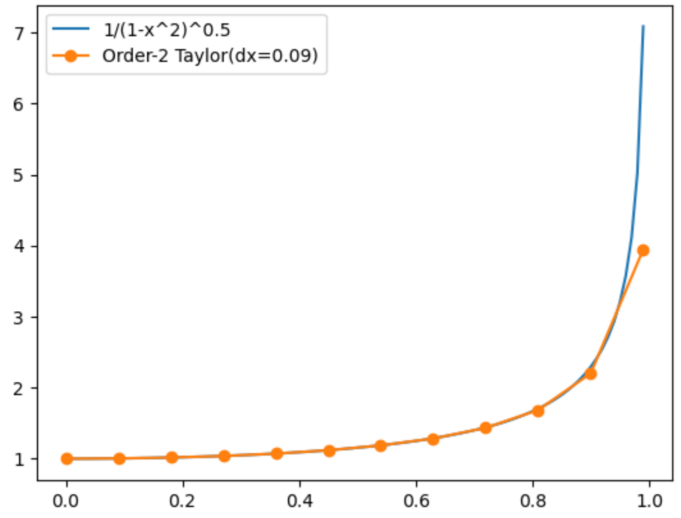
Out[49]=



or in Python ...



```
def taylor2(x,a):
    return (2 + 6*a**4 - 6*a**3*x + x**2 +
            a**2*(-5 + 2*x**2))/(2*(1 - a**2)**(5/2))

from matplotlib import pyplot as plt

import numpy as np

xvals=np.arange(0,1,0.01)
yvals=[1/(1-x**2)**0.5 for x in xvals]
plt.plot(xvals, yvals, label="1/(1-x^2)^0.5")

dx=0.09
taylors=[1]+[taylor2(x,x-dx)
             for x in np.arange(dx,1,dx)]
plt.plot(np.arange(0,1,dx), taylors, "o-",
        label=f"Order-2 Taylor(dx={dx})")
plt.legend()
plt.show()
```



This code is in the TaylorExample.py file on Moodle.

Gradient Descent Algorithm

Many practical problems have hundreds of variables.

Calculations become intractable very quickly so a better method is needed.

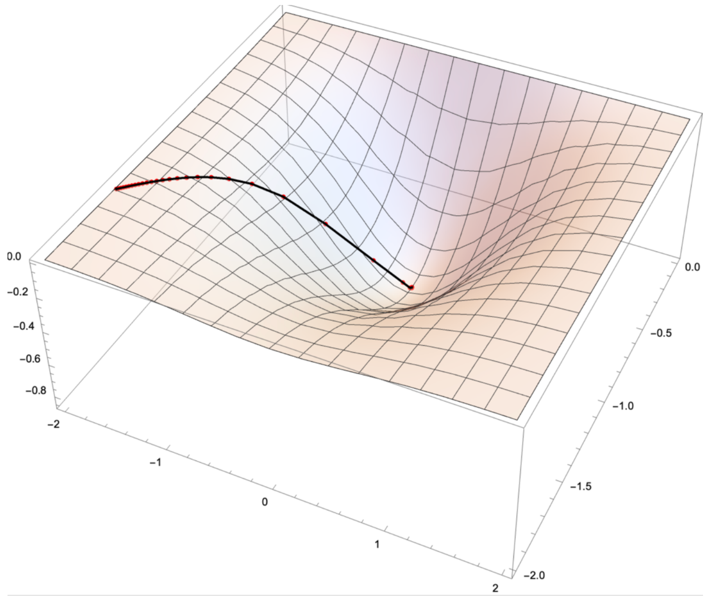
The Gradient Descent Algorithm is a method for finding (approximately) a local minimum.

Gradient descent is an optimization algorithm that is commonly-used to train machine learning models and neural networks.

How the Gradient Descent Algorithm Works

Follow the path of steepest descent down moving in small increments to allow for the steepest direction of descent to be repeatedly re-calculated.

Stop when we get close enough to the “bottom”.



Direction of Steepest Descent

The downward direction where the slope is steepest is the opposite direction to the gradient vector i.e.

$$-\nabla f$$

(This was proved at the end of last week’s lecture)

Gradient Descent Algorithm

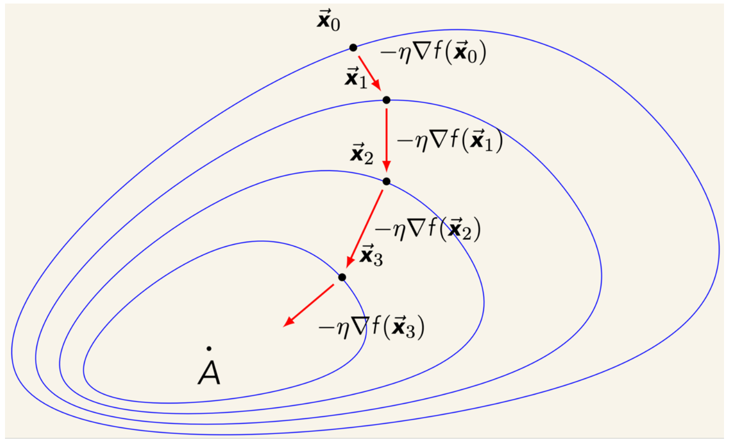
INPUT: $f : \mathbb{R}^n \to \mathbb{R}$ differentiable

$\eta > 0$	the learning rate
$\delta > 0$	the precision
$\boldsymbol{x}_0$	the initial point

ALGORITHM:

```
1. $t \leftarrow 0$
2. Define $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t)$
3. If $| \boldsymbol{x}_t - \boldsymbol{x}_{t+1} | > \delta$ let $t \leftarrow t + 1$ and go to step 2.
```

OUTPUT: $\boldsymbol{x}_t$



$\vec{\boldsymbol{x}}_0$
$-\eta\nabla f(\vec{\boldsymbol{x}}_0)$
$\vec{\boldsymbol{x}}_1$
$-\eta\nabla f(\vec{\boldsymbol{x}}_1)$
$\vec{\boldsymbol{x}}_2$
$-\eta\nabla f(\vec{\boldsymbol{x}}_2)$
$\vec{\boldsymbol{x}}_3$
$-\eta\nabla f(\vec{\boldsymbol{x}}_3)$
A

Algorithm Code

ALGORITHM:

```
1. $t \leftarrow 0$
2. Define $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t)$
3. If $| \boldsymbol{x}_t - \boldsymbol{x}_{t+1} | > \delta$ let $t \leftarrow t + 1$ and go to step 2.
```

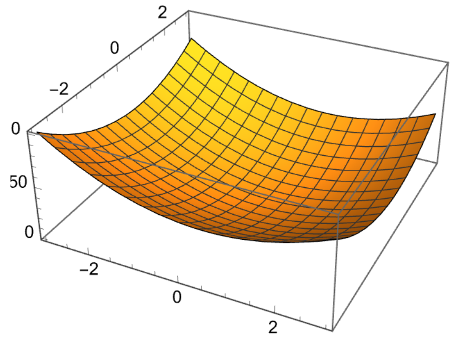
Instead of running step 2 until we achieve the desired accuracy δ, we will just run step 2 a fixed number of times (n), maybe 10 or 20. This makes coding simpler and will help you to investigate the effect of the learning rate parameter η.

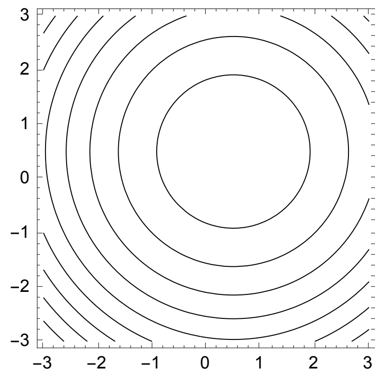
```
gda[f_,x0_,η_,n_]:=
    Module[{pt=x0, gradf=Grad[f,{x,y}], path={x0}},
        Do[
                {pt=pt - η gradf /.{x→pt⟦1⟧,y→pt⟦2⟧},
                path = Append[path, pt]}
            ,n];
        path]
```

Example Problem

Find a local minimum for $f(x, y) = 4\left(x^2 - x + y^2 - y + 1\right)$ using the gradient descent algorithm.

Out[ ]=





In[ ]:=

```
Grad[f,{x,y}]
```

Out[ ]=

```
{4 (-1 + 2 x), 4 (-1 + 2 y)}
```

Impact of η on Example Problem

Fast convergence: $x_0 = (-2, 2)$, $\eta=0.1$, number of iterations $n = 10$

In[108]:=

```
gda[4(x^2-x+y^2-y+1),{-2,-2},0.1,10]
```

Out[108]=

```
{{-2, -2}, {0., 0.}, {0.4, 0.4}, {0.48, 0.48}, {0.496, 0.496}, {0.4992, 0.4992},
 {0.49984, 0.49984}, {0.499968, 0.499968}, {0.499994, 0.499994}, {0.499999, 0.499999}, {0.5, 0.5}}
```

Oscillating but convergent: $x_0 = (-2, 2)$, $\eta=0.2$, number of iterations $n = 10$

In[109]:=

```
gda[4(x^2-x+y^2-y+1),{-2,-2},0.2,10]
```

Out[109]=

```
{{-2, -2}, {2., 2.}, {-0.4, -0.4}, {1.04, 1.04}, {0.176, 0.176}, {0.6944, 0.6944},
 {0.38336, 0.38336}, {0.569984, 0.569984}, {0.45801, 0.45801}, {0.525194, 0.525194}, {0.484883, 0.484883}}
```

Divergent: $x_0 = (-2, 2)$, $\eta=0.3$, number of iterations $n = 10$

In[110]:=

```
gda[4(x^2-x+y^2-y+1),{-2,-2},0.3,10]
```

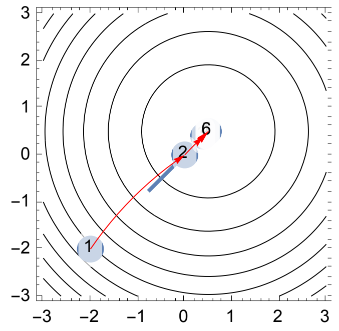
Out[110]=

```
{{-2, -2}, {4., 4.}, {-4.4, -4.4}, {7.36, 7.36}, {-9.104, -9.104}, {13.9456, 13.9456},
 {-18.3238, -18.3238}, {26.8534, 26.8534}, {-36.3947, -36.3947}, {52.1526, 52.1526}, {-71.8137, -71.8137}}
```

Algorithm Visualisation

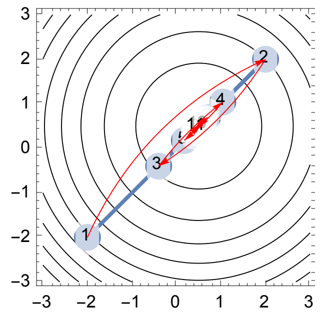
$x_0 = (-2, 2)$, η=0.1, number of iterations $n = 5$

Out[280]=



$x_0 = (-2, 2)$, η=0.2, number of iterations $n = 10$

Out[283]=



Machine Learning via Gradient Descent

Linear regression finds a “best fit” line to n data points (x_i, y_i) by minimising the following cost function:

$$f(m, c) = \sum_{i=1}^{n} (y_i - (m\, x_i + c))^2$$

The gradient descent algorithm requires us to find the gradient vector:

$$\nabla f(m, c) = \{f_m,\ f_c\}$$

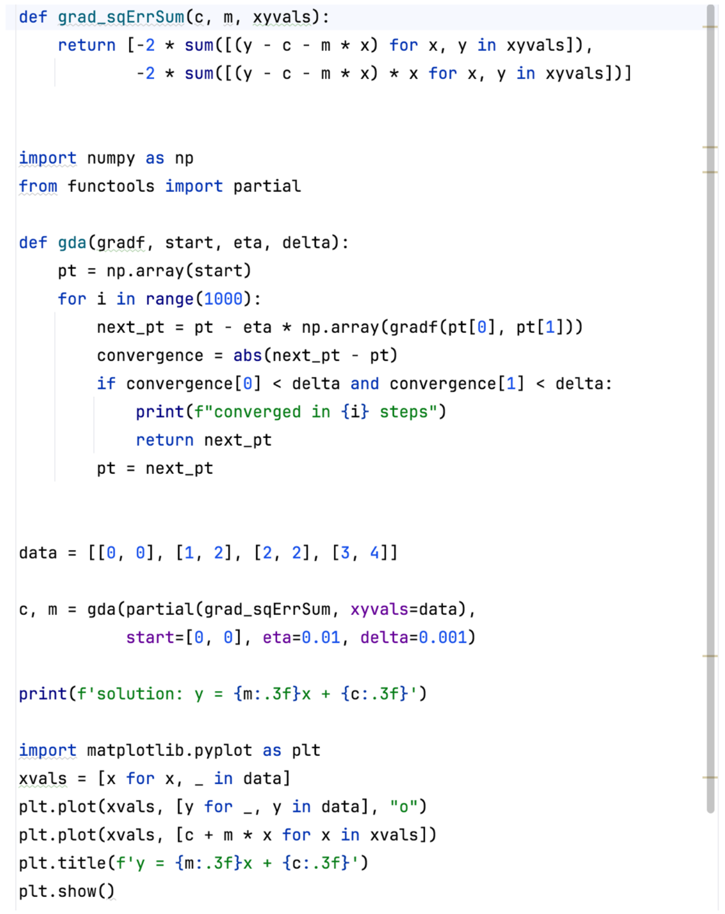
As all x_i, y_i values are constant values these partial derivatives are:

$$f_m = \sum_{i=1}^{n} 2\,(y_i - (m\, x_i + c))\,(-x_i), \qquad f_c = \sum_{i=1}^{n} 2\,(y_i - (m\, x_i + c))\,(-1)$$

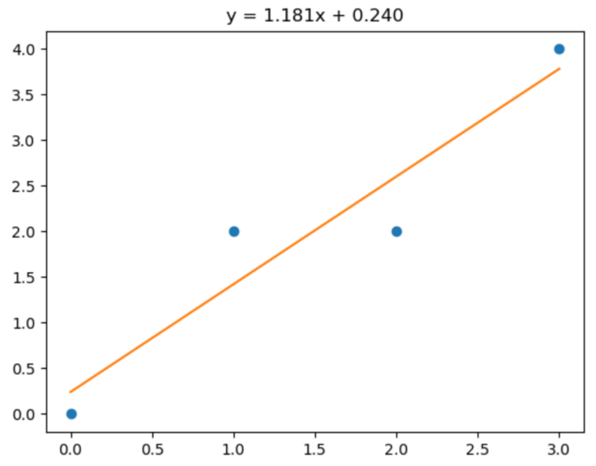
With these partial derivatives the gradient descent algorithm can be followed in the normal way.

Machine learning methods - including neural networks - are based on minimising an associated cost function so can be run using a similar approach to above.

in Python ...



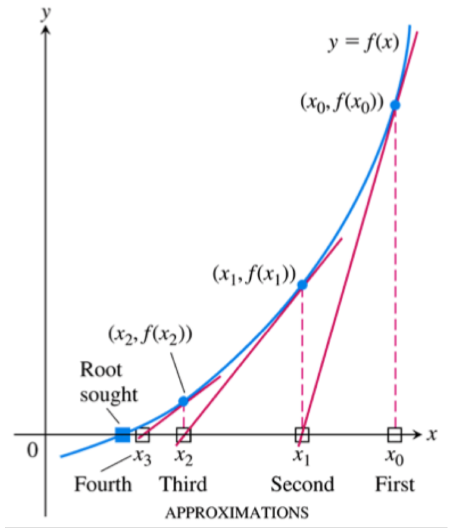
```python
def grad_sqErrSum(c, m, xyvals):
    return [-2 * sum([(y - c - m * x) for x, y in xyvals]),
            -2 * sum([(y - c - m * x) * x for x, y in xyvals])]


import numpy as np
from functools import partial

def gda(gradf, start, eta, delta):
    pt = np.array(start)
    for i in range(1000):
        next_pt = pt - eta * np.array(gradf(pt[0], pt[1]))
        convergence = abs(next_pt - pt)
        if convergence[0] < delta and convergence[1] < delta:
            print(f"converged in {i} steps")
            return next_pt
        pt = next_pt


data = [[0, 0], [1, 2], [2, 2], [3, 4]]

c, m = gda(partial(grad_sqErrSum, xyvals=data),
           start=[0, 0], eta=0.01, delta=0.001)

print(f'solution: y = {m:.3f}x + {c:.3f}')

import matplotlib.pyplot as plt
xvals = [x for x, _ in data]
plt.plot(xvals, [y for _, y in data], "o")
plt.plot(xvals, [c + m * x for x in xvals])
plt.title(f'y = {m:.3f}x + {c:.3f}')
plt.show()
```



This code is in GDAexample.py on Moodle.

Newton’s Method (aka Newton-Raphson Method)

Newton’s Method is a numerical method for finding the roots of differentiable functions.

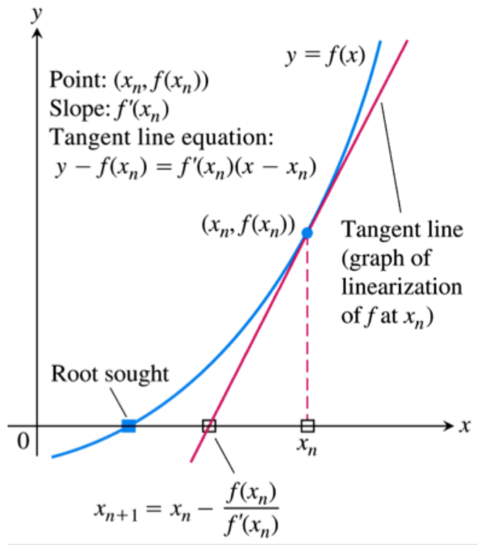


Newton’s Method

1. Guess a first approximation to a solution of the equation $f(x) = 0$.

2. Use the first approximation to get a second, the second to get a third, and so on , using the formula

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



y
$y = f(x)$
Point: $(x_n, f(x_n))$
Slope: $f'(x_n)$
Tangent line equation:
$y - f(x_n) = f'(x_n)(x - x_n)$
$(x_n, f(x_n))$
Tangent line (graph of linearization of f at x_n)
Root sought
0
x_n
x
$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

Example

Find approximation for a root of $x^3 - x - 1 = 0$ using Newton’s Method.

Let $f(x) = x^3 - x - 1$ so $f'(x) = 3x^2 - 1$

Use Newton's method formula: $x_{n+1} = x_n - \frac{(x_n^3 - x_n - 1)}{(3x_n^2 - 1)}$

Try $x_1 = 1.5$ as a first guess.

Repeatedly performing this calculation in *Mathematica* we get ...

In[337]:=

$$x_n - \frac{(x_n^3 - x_n - 1)}{(3x_n^2 - 1)} \;/.\; x_n \to 1.5$$

Out[337]=

```
1.34783
```

In[338]:=

$$x_n - \frac{(x_n^3 - x_n - 1)}{(3x_n^2 - 1)} \;/.\; x_n \to \%$$

Out[338]=

```
1.3252
```

The actual root is x = 1.32472 ...

Limitations of Newton’s Method

In practice Newton’s method usually converges quickly on a root. However, Newton’s method does not always converge

When Newton’s method does converge, it converges to a root. But the root may not be the one that is closest to your initial guess.

It is always a good idea to sketch graphs first to identify a good initial guess for the root you want.

Mathematica Code for Newton’s Method

In[419]:=

```
NewtonRoot[f_,x0_,n_]:=
    Module[{xn=x0, df=D[f,x]},
        Do[{xn=xn -(f/df)/.x→xn //N,
            Print[xn]},
            n]]
```

In[420]:=

```
NewtonRoot[x^3-x-1, 1.5, 5]
```

```
1.34783

1.3252

1.32472

1.32472

1.32472
```

FindRoot

The *Mathematica* function **FindRoot** uses alternative numerical methods to find roots, including Newton’s Method.

In[425]:=

```
FindRoot[x^3-x-1,{x,1.5}, Method→"Newton"]
```

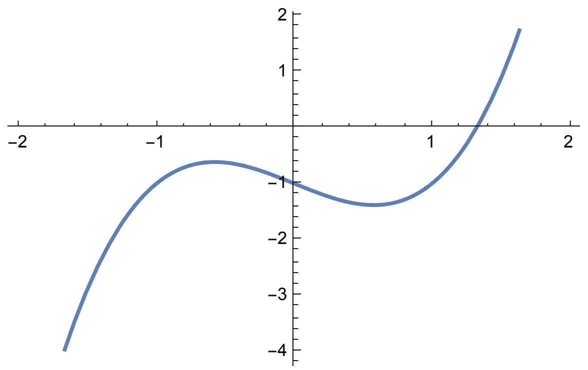
Out[425]=

```
{x → 1.32472}
```

In[424]:=

```
Plot[x^3-x-1,{x,-2,2}]
```

Out[424]=


2
1
−2
−1
1
2
−1
−2
−3
−4