Geometric Computing

The last topic we will look at is aspects of **Geometric Computing**.

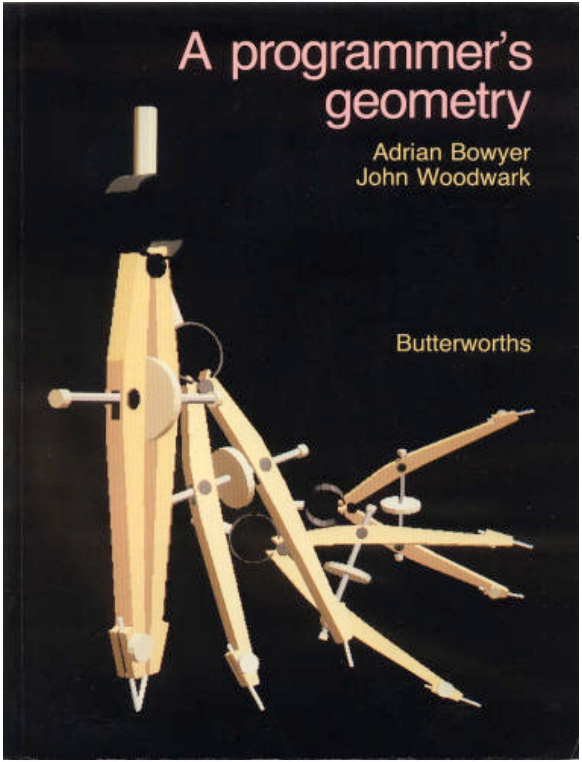
The fundamental basics of:

- Computer Graphics
- Image Processing and Computer Vision
- Spatial Reasoning, Geographic Information Systems.

Builds on Linear Algebra:

- Vectors, Matrices
- Linear Equations

References



A Programmer's Geometry,
Adrian Bowyer,
John Woodwark,
Butterworths, 1983,
ISBN: 0408012420.

MATLAB Geometry Toolbox

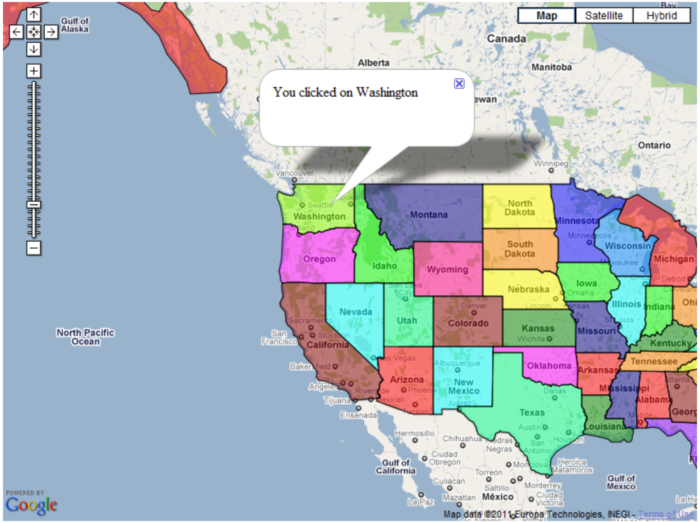
Example Applications

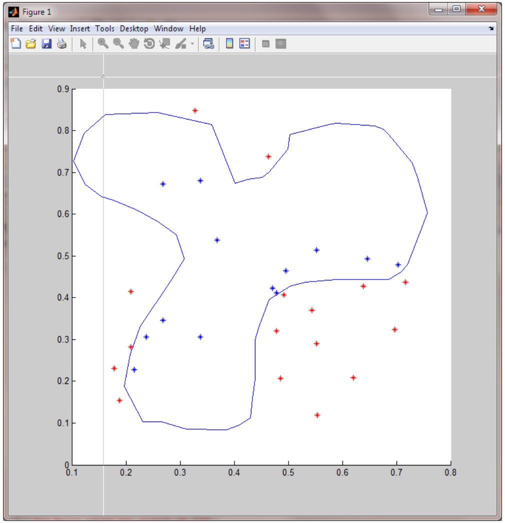
We show some practical application scenarios of geometric computing and some demos. These are only some examples and there are many more possibilities:

- Geographic Information Systems: Point Location
- Geometric Modelling: Spline Fitting
- Computer Graphics: Ray Tracing
- Image Processing: Hough Transform
- Mobile Systems: Spatial Location Sensing

Example Application: Point Location in GIS etc.

GIS involve lots of geometric primitives and their interactions. A simple example is testing if a point locates within a certain region (often modelled as a polygon). This can be used to find where you are from GPS data or identify the region that the user clicks.





Interactive Map Demo

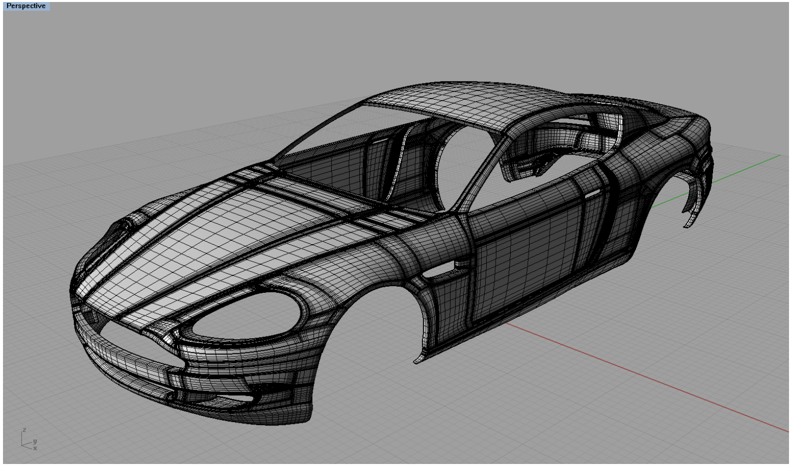
CM0268
MATLAB
DSP
GRAPHICS

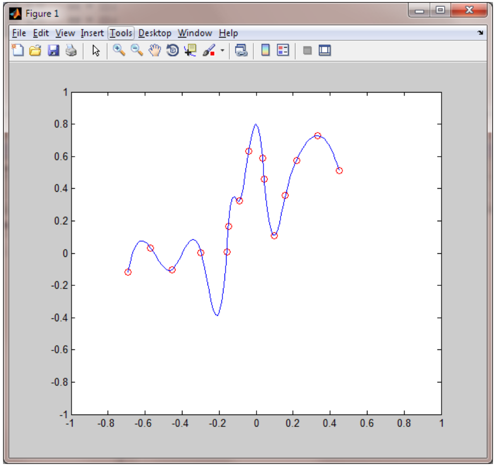
Back

Close

Example Application: Spline Fitting in Geometric Modelling

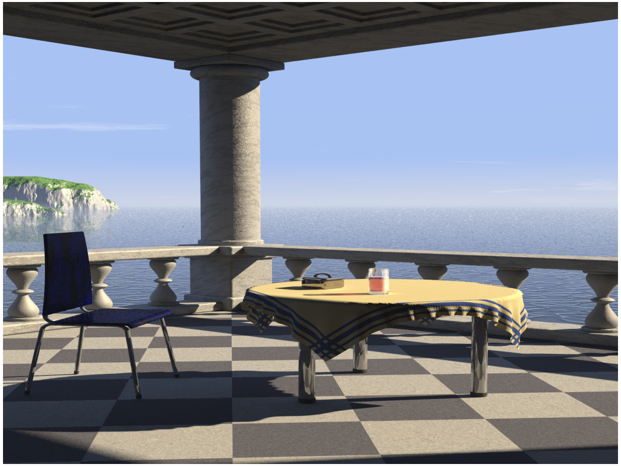
Geometric modelling provide tools that help design and manufacture of products (e.g. cars, airplanes, garments etc.) Spline (piecewise polynomial curves and surfaces) is a fundamental technique.

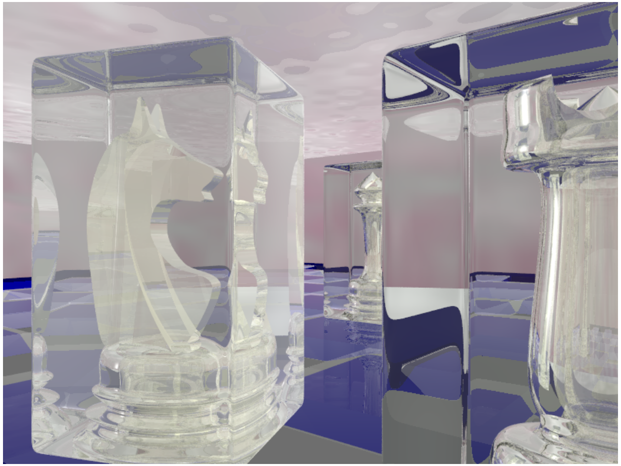




Example Application: Ray Tracing in Computer Graphics

Computer graphics aim at reproducing or creating vivid animations in computers. Ray tracing is a widely used technique for generating high-quality rendering of virtual scenes.

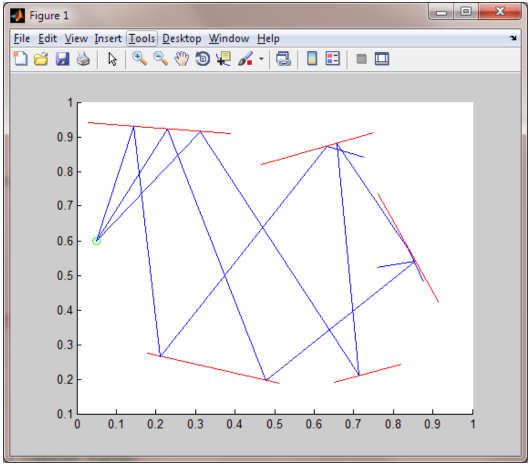


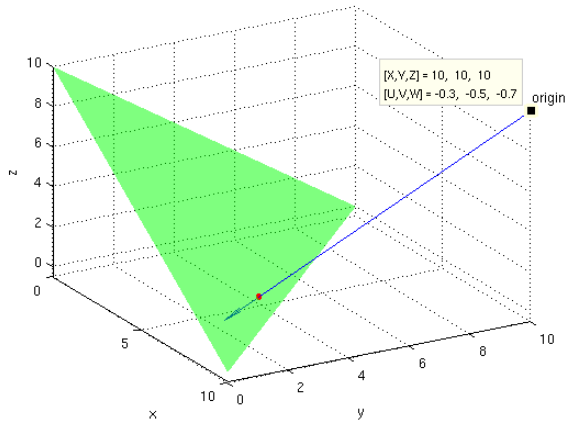


Rendered with POV-RAY

Ray Tracing in Computer Graphics (cont.)

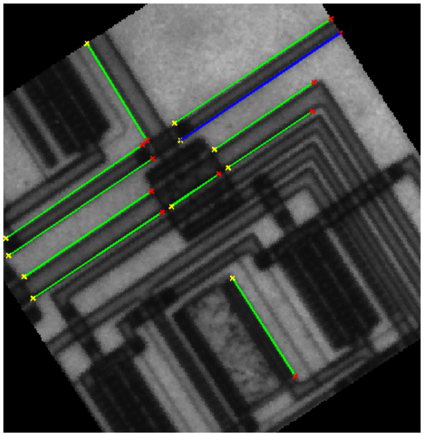
Ray object intersection is the key operation in ray tracing algorithm.
Some demos:

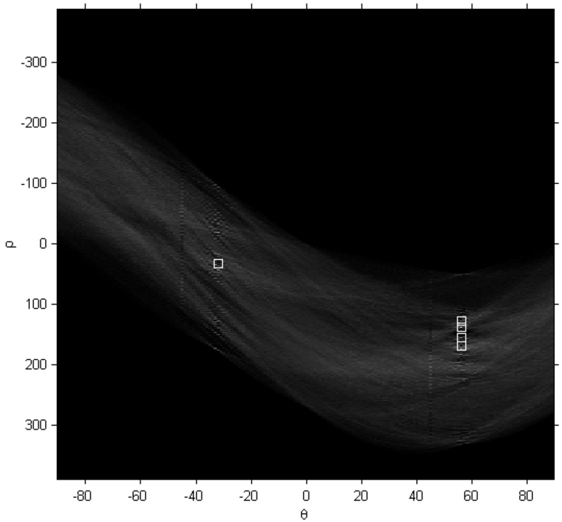




Example Application: Hough Transform in Image Processing / Computer Vision

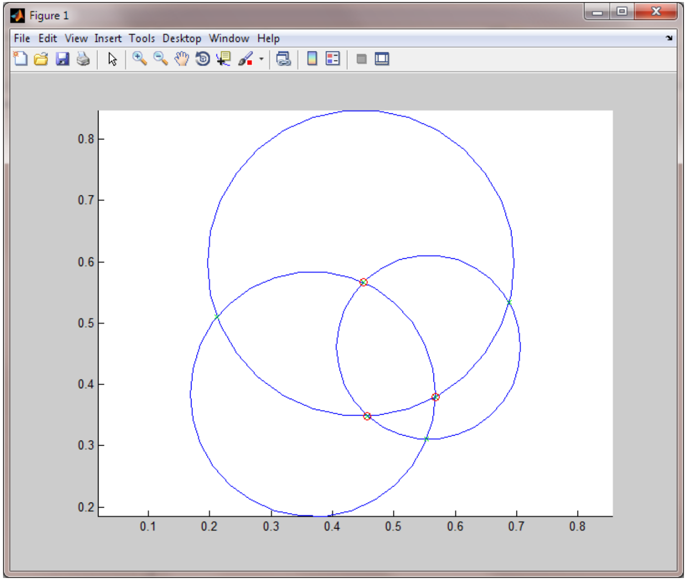
Computer Vision considers the inverse problem of "understanding" images. To identify some significant structures from images is needed by many application scenarios. Hough transform is used to find prominent features (lines, circles etc.) from images, using some voting scheme in the implicit parameter space.





Example Application: Spatial Location Sensing in Mobile Systems

With techniques such as RFID, 3D location sensing is possible. Multiple sources of information can be combined, potentially with some uncertainty. A simple 2D demo involves circle to circle intersection to identify the common region suggested by multiple sensors.

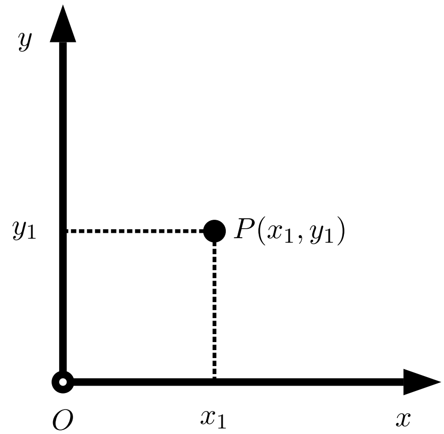


Coordinate Systems: 2D

The **Cartesian coordinate system** (also called *rectangular coordinate system*) determines each point uniquely in a plane through two numbers,usually called

- the **x-coordinate** or **abscissa**
- the **y-coordinate** or **ordinate** of the point.

with respect to two orthogonal axes, the x-axis and y axis.

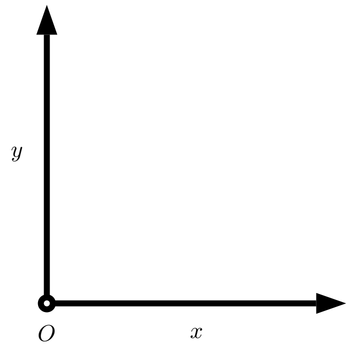


2D Coordinate Systems: Handedness

We can draw our coordinate system in one of two ways.

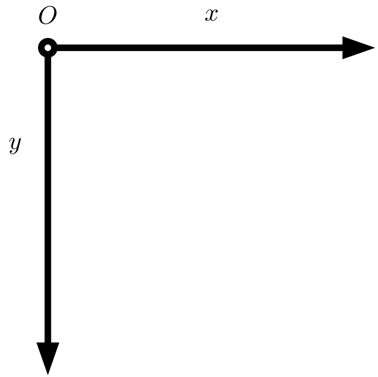
Fixing the x-axis to point horizontally from left to right, we can draw the y-axis in one of two ways:

y-axis pointing up vertically



Also called the **positive** or **standard** orientation

y-axis pointing down vertically

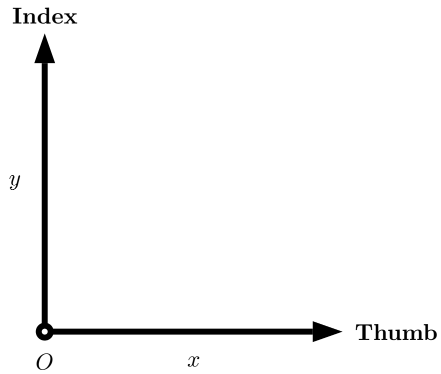


2D Coordinate Systems: Right/Left Handedness

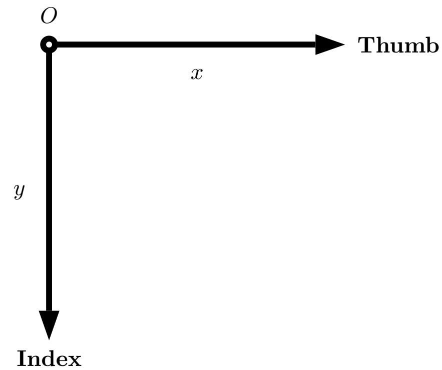
An easy way to define and remember each coordinate system is to **use your hands**:

- Assign your **thumb** to the x**-axis**
- Assign your **index finger** to the y**-axis**
- Right or left hand will align with axes accordingly (Palm facing towards you).

Right Handed System

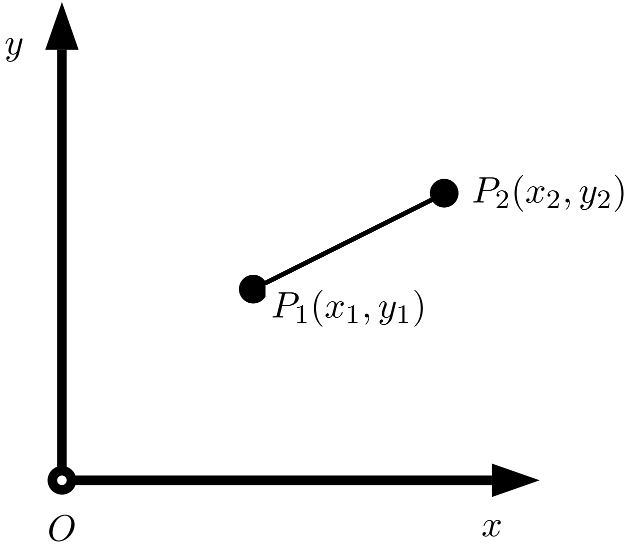


Left Handed System

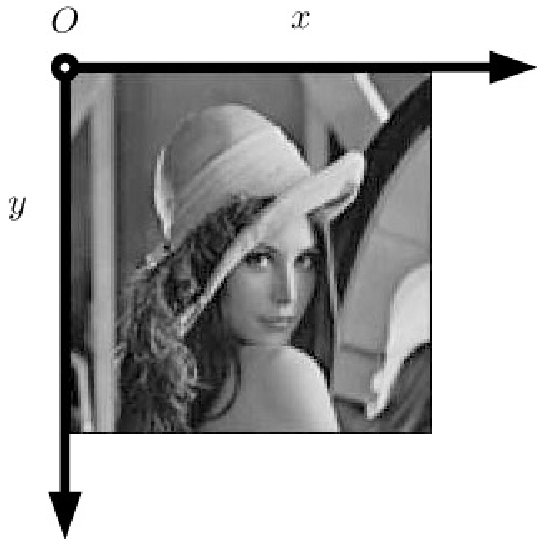


2D Coordinate Systems: Handedness Examples

Right Handed System:
Standard Graph Plotting



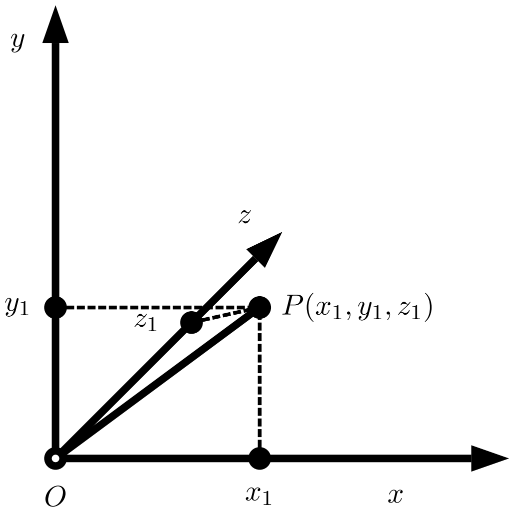
Left Handed System:
Image Pixel Coordinate Indexing



3D Coordinate Systems

3D coordinates systems build on similar ideas to the previous 2D systems, we now need to account for the **third dimension** — the **z-axis**.

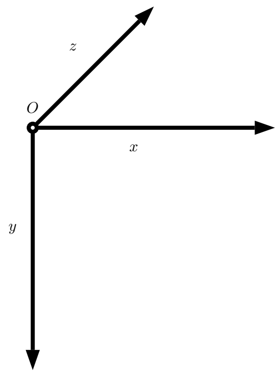
All three axes are orthogonal (perpendicular) to each other.

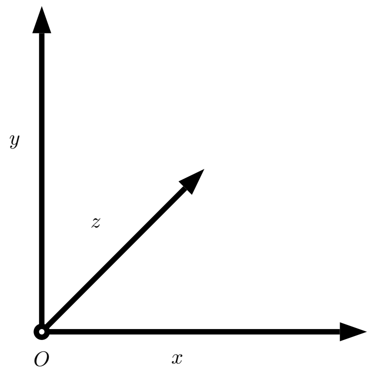


3D Coordinate Systems: Handedness

As with 2D, we can draw our coordinate system in one of two ways.

Once the x- and y-axes are specified, they determine the line along which the z-axis should lie, but there are two possible directions on this line:



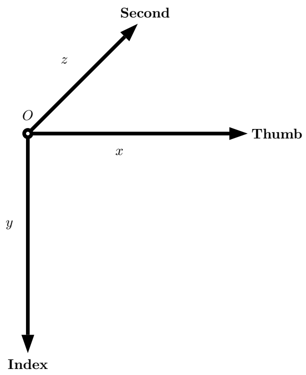


3D Coordinate Systems: Right/Left Handedness

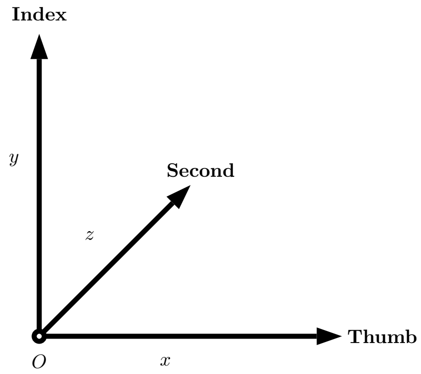
Again **use your hands**:

- Assign your **thumb** to the x**-axis**
- Assign your **index finger** to the y**-axis**
- Assign your **second finger** to the z**-axis**
- Right or left hand will align with axes accordingly (sometimes with some contortion!).

Right Handed System



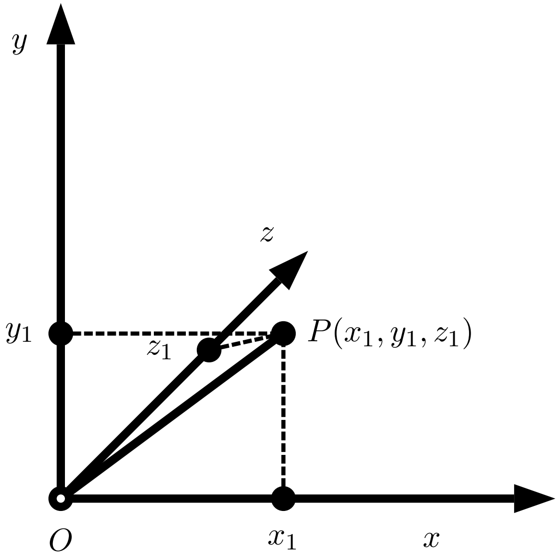
Left Handed System



3D Coordinate Systems: Handedness Examples

Right Handed System:
Video Stack Indexing:
z-axis is time

Left Handed System:

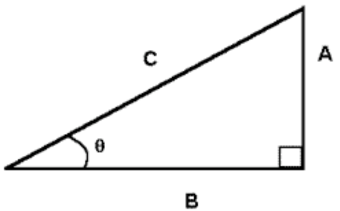


Mathematical Tools Recap

We review some simple mathematical tools used throughout the session.

Basic Trigonometric Formulae / Pythagoras' Theorem

For a right-angle triangle

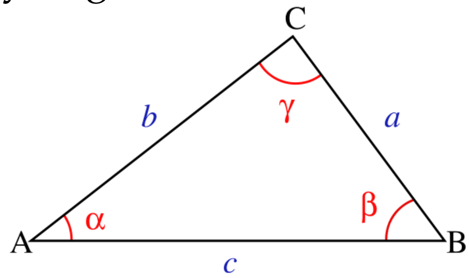


$\sin\theta = A/C$, $\cos\theta = B/C$ and $\tan\theta = A/B$
Also Pythagoras' Theorem states that

$$A^2 + B^2 = C^2$$

Law of Consines

A generalisation of Pythagoras's Theorem:



$$c^2 = a^2 + b^2 - 2ab\cos\gamma.$$

If $\gamma = 90^\circ$, $\cos\gamma = 0$, this is equivalent to Pythagoras' Theorem.

Basic Linear Algebra/Vector Formulae

For two 3D vectors $\mathbf{v}_1 = (x_1, y_1, z_1)$ and $\mathbf{v}_2 = (x_2, y_2, z_2)$.

$$\mathbf{v}_1 \pm \mathbf{v}_2 = (x_1 \pm x_2, y_1 \pm y_2, z_1 \pm z_2)$$

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = x_1 x_2 + y_1 y_2 + z_1 z_2$$

$$\mathbf{v}_1 \times \mathbf{v}_2 = \begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} = (y_1 z_2 - y_2 z_1, x_2 z_1 - x_1 z_2, x_1 y_2 - x_2 y_1).$$

Matrix operations (addition, subtraction, multiplication and division).

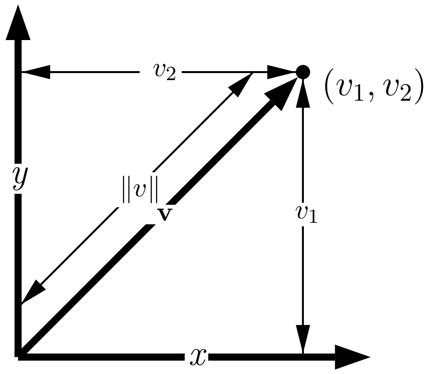
Euclidean norm of a vector

For a vector $\mathbf{v} \in \mathbb{R}^n$ we define its norm as

$$\|\mathbf{v}\| = \sqrt{\mathbf{v}.\mathbf{v}}$$

This norm is called the euclidean norm of the vector $\mathbf{v}$.

The euclidean norm of a vector coincides with the length of the vector in $\mathbb{R}^2$ and $\mathbb{R}^3$.



By Pythagoras' Theorem, $\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2} = \sqrt{\mathbf{v}.\mathbf{v}}$

Cauchy-Schwarz inequality

Let v and w be vectors in $\mathbb{R}^n$

Then they satisfy the Cauchy-Schwarz inequality

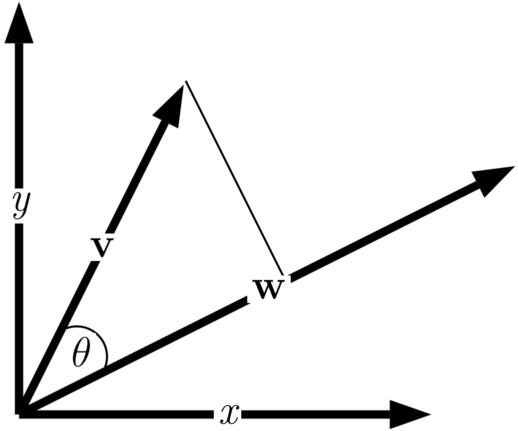
$$\mathbf{v}.\mathbf{w} \leq \|\mathbf{v}\|\|\mathbf{w}\|.$$

Angle Between Two Vectors
If $n = 2, 3$ we even have the relation

$$\mathbf{v}.\mathbf{w} = \|\mathbf{v}\|\|\mathbf{w}\| \cos \theta$$

We call θ the *angle between v and w*.

Geometric Visualisation of Angle Between Two Vectors in $\mathbb{R}^2$



Variable Substitution

To ease algebraic manipulation in deriving equations it may be useful to group variables together by substituting the group for a single variable. This may be replaced later in the derivation if needed.

For example it is far easier to expand $(x+x_t)^2$ rather than $(x+x_a+x_b+x_c)^2$.

Here we simply let $x_t = x_a + x_b + x_c$

Quadratic Equations

If $ax^2 + bx + c = 0$ then the roots of x are given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Determinants

A determinant is a number. A determinant is evaluated by scanning along one of its rows or columns and alternately adding and subtracting the value of the determinant formed by omitting the row and column corresponding to the value multiplied by that value.

A second order determinant

$$\begin{vmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{vmatrix} = d_{11}d_{22} - d_{12}d_{21}$$

A third order determinant

$$\begin{vmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{vmatrix}$$

$$= d_{11}(d_{22}d_{33} - d_{32}d_{23}) - d_{12}(d_{21}d_{33} - d_{31}d_{23}) + d_{13}(d_{21}d_{32} - d_{31}d_{22}).$$

Linear Equations

For a linear system with n unknowns, $x_1, x_2, \ldots, x_n$, to have a unique solution, n independent linear equations are needed:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\cdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

Write in matrix form with $\mathbf{A} = (a_{ij})_{n\times n}$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)^T$:

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

Cramer's rule states: the system has unique answer if and only if $|\mathbf{A}| \neq 0$. The solution is

$$x_i = \frac{|\mathbf{A}_i|}{|\mathbf{A}|}.$$

$\mathbf{A}_i$ is matrix $\mathbf{A}$ with i^{th} column replaced by b.

Apply Other Geometric Formulae

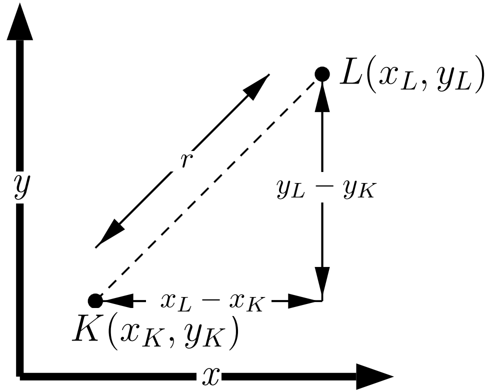
Many of the simpler derivations, such as perpendicular distance of a point to a line or derivation of a line equation, are used in more involved derivations.

Know your core Geometric derivations

Points and Lines

Distance between Two Points in 2D

Given 2 points K and L the distance, r, between them in 2D is:

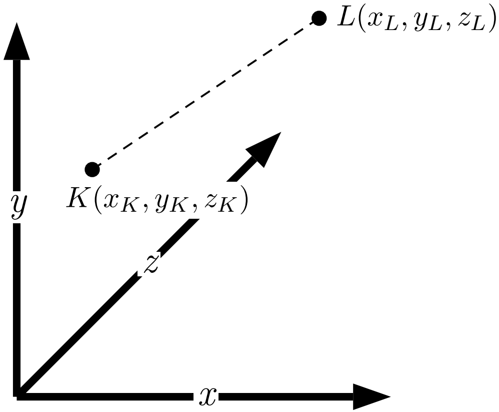


$$r = \sqrt{(x_L - x_K)^2 + (y_L - y_K)^2}$$

Proof by simple application of Pythagoras' theorem.

Distance between Two Points in 3D

Given 2 points K and L the distance, r, between them in 3D is, by simple extension from 2D:



$$r = \sqrt{(x_L - x_K)^2 + (y_L - y_K)^2 + (z_L - z_K)^2}$$

MATLAB Computation Distance between Two Points

We can write a one line MATLAB statement to compute the distance between points in n-**dimensions**, see points_dist.m:

```
dist = sqrt ( sum ( ( p1 - p2).^2 ) );
```

Alternatively we could use built in MATLAB function `norm()`

```
dist = norm(p1 - p2);
```

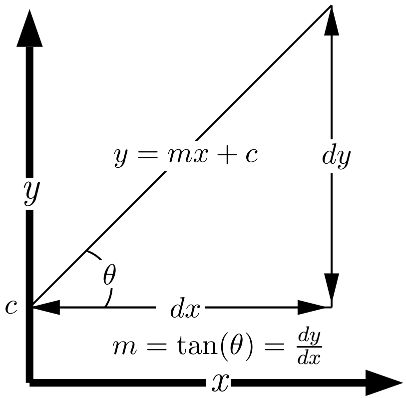
Note: MATLAB function `norm()` computes other norms also, see MATLAB function `help norm()`

Equations of a Line

Explicit Equation of a line in 2D

The best known equation of a line is:

$$y = mx + c$$



where m is the **line gradient** given by $m = \tan(\theta) = \frac{dy}{dx}$ and c is the intercept with the y-axis

Computational Problems with Explicit Equation of a line

As line becomes near vertical (parallel with the y-axis) m becomes very large as $\tan(90°) = \infty$.

So computationally this representation of a line is practically useless.

Implicit Equation of a line in 2D

A more computationally stable form of line equation is:

$$ax + by + c = 0$$

$$(a^2 + b^2 \neq 0)$$

Can you show how these representations are related?

Normalised Implicit Equation of a line in 2D

The form $ax + by + c = 0$ can be multiplied by any non-zero constant without altering its meaning — which can cause problems.

It is more useful to ***normalise***, or put the equation into **canonical** form, by imposing the constraint:

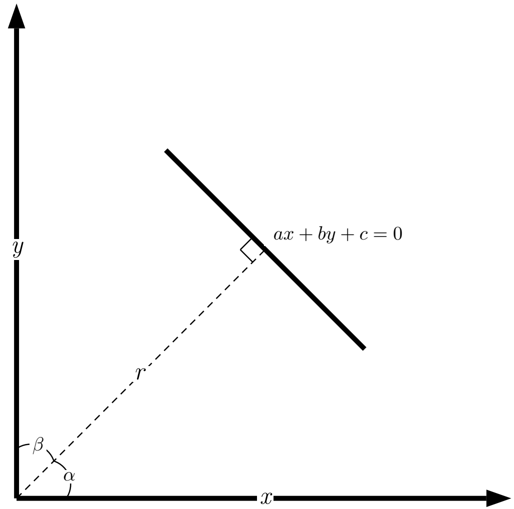
$$a^2 + b^2 = 1$$

This is most simply achieved by multiplying through by:

$$\frac{1}{\sqrt{a^2 + b^2}}$$

The normalised form also has more intuitive meaning.

Intuitive Meaning Normalised Implicit Equation of a 2D line



- The normalised form of a and b are ***direction cosines*** — the cosines of angles which the normal to the line makes with the x and y axes.
- The normalised form of c is the ***perpendicular distance*** from the line to the origin

So here: $a = \cos(\alpha)$, $b = \cos(\beta)$, and $c = -r$

Parametric Equation of a line

There is another form of line equation: ***parametric form***:

- Based on a vector representation of line
- Generalises well to higher dimensions

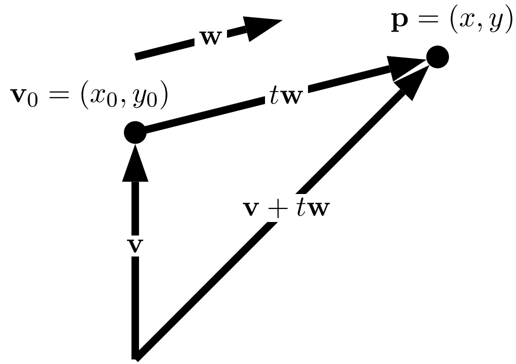
The 2D parametric form consists of 2 equations and gives x and y in terms of a third variable t:

$$
\begin{aligned}
x &= x_0 + ft \\
y &= y_0 + gt
\end{aligned}
$$

We can visualise the parametric form via vectors

Vector Visualisation of the Parametric Equation of a Line

Let point $\mathbf{v}_0 = (x_0, y_0)$ and let a vector $\mathbf{w} = (f, g)$



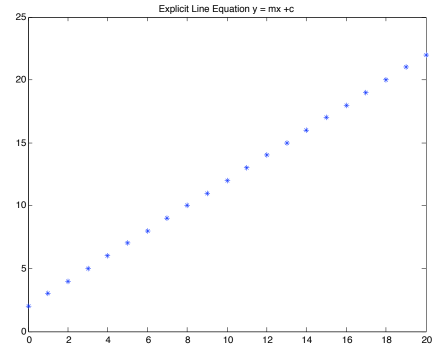
The position on any point, $P(x, y)$ can be given as $\mathbf{v}_0 + t\mathbf{w}$ where:

- $\mathbf{v}_0$ is a vector given the offset of the vectors base from the origin.
- $P(x, y)$ is some distance $t\mathbf{w}$ along the vector $\mathbf{w}$
  - $\mathbf{v}_0$ is clearly at position $t = 0$
- Vector $\mathbf{w}$ is usually specified as a normal vector (**unit length**).
- Negative t moves points in opposite direction to $\mathbf{w}$

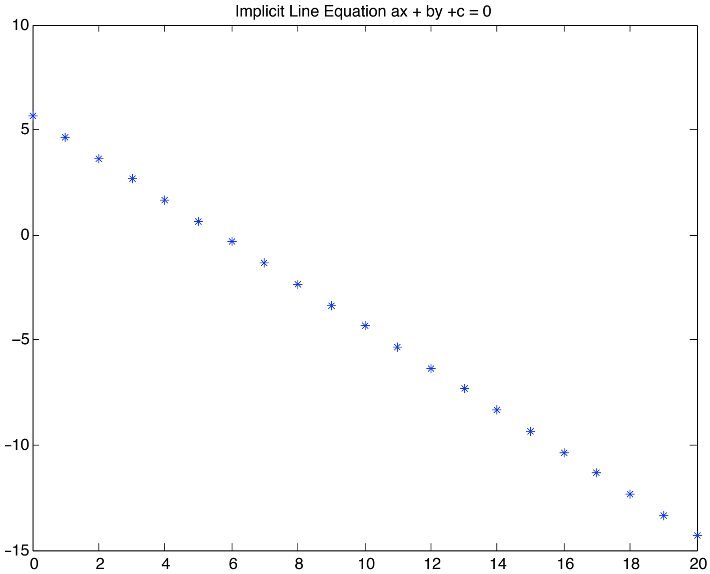
Line Representations in MATLAB

As we will see shortly, MATLAB deals with line plotting very easily. However lets look at plotting line points directly from the equations, see lines.m:

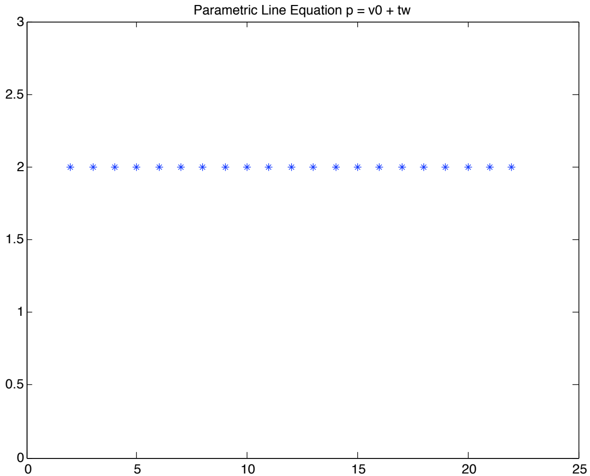
```
% Explicit form of Equation y = mx + c
n=20; % 20 points
x = 0:n; % make n x coordinate values
m = 1; c = 2; % set explicit parameters
y = m*x + c;    % compute y coordinates

figure(1) % Plot Figure
plot(x,y)
axis([0 20 0 25]) % set axes to see plot
title('Explicit Line Equation y = mx +c');
```



```
% Implicit form ax + by +c = 0
% set implicit parameters
a = cos(45*pi/180); b = cos(45*pi/180); c = -4;
y = -(a*x +c)/b; % compute y coordinates

figure(2);
plot(x,y);
title('Implicit Line Equation ax + by +c = 0');
```



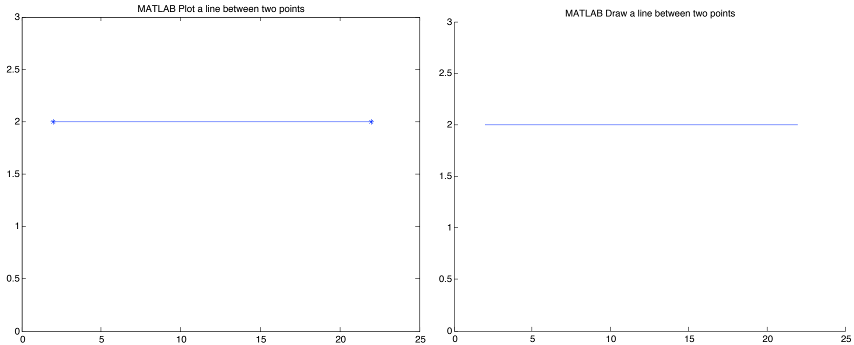
```
% Parametric form p = v0 + tw
v0 = [2,2];
w = [1,0];
t = 0:n;   % create a vector of t values

x = v0(1) + t*w(1); % Compute x
y = v0(2) + t*w(2); % Compute x

figure(3);
plot(x,y);
axis([0 25 0 3]) % set axes to see plot
title('Parametric Line Equation p = v0 + tw');
```



Drawing a line in MATLAB

To simply draw a line use the MATLAB, `plot()` or `line()` functions, see previous notes:

```
figure(4)
% just plot end points
plot([x(1) x(end)],[y(1) y(end)],'*-');
axis([0 25 0 3]) % set axes to see plot
title('MATLAB Plot a line between two points');

figure(5)
% just draw between end points
line([x(1) x(end)],[y(1) y(end)]);
axis([0 25 0 3]) % set axes to see plot
title('MATLAB Draw a line between two points');
```



Converting between Parametric and Implicit Form

Solving the simultaneous equation:

$$\begin{aligned} x &= x_0 + ft \\ y &= y_0 + gt \end{aligned}$$

for t we readily get the implicit form:

$$-gx + fy + (gx_0 - fy_0) = 0$$

So $a = -g$, $b = f$ and $c = (gx_0 - fy_0)$

A MATLAB function to achieve this simple task is line_par2imp_2d.m

Converting between Implicit and Parametric Form

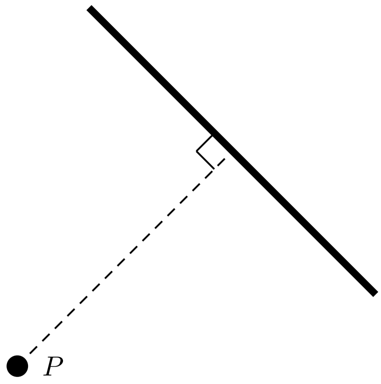
A general (but not necessarily normailsed) implicit line $ax + by + c = 0$ is parameterised as:

$$
\begin{aligned}
x &= \frac{-ac}{(a^2 + b^2)} + bt \\
y &= \frac{-bc}{(a^2 + b^2)} - at
\end{aligned}
$$

which can be coded as follows, line_imp2par_2d:

```
root = a * a + b * b;
if ( root == 0.0 )
    fprintf ( 1, ' Error!: A * A + B * B = 0.\n' );
  end

x0 = - a * c / root;
y0 = - b * c / root;
root = sqrt(root);
f =   b / root;
g = - a / root;

if ( f < 0.0 )
   f = -f;
   g = -g;
 end
```

Perpendicular Distance from a Point to a Line



For the implicit line form $ax + by + c = 0$, the shortest (**perpendicular**) from a point $P(x_p, y_p)$ to the line is given by:

$$d = \frac{ax_p + by_p + c}{\sqrt{a^2 + b^2}}$$

$a^2 + b^2$ clearly equals 1 if the line is normalised and can be omitted in this case.

Perpendicular Distance from a Point to a Line (cont.)

- If d equals 0, P is on the line.
- The sign of d indicates which side of the line the point is on.
- If this information is not required then take the absolute value of d.

The MATLAB code to achieve this is line_imp_point_dist_2d:

```
if ( a * a + b * b == 0.0 )
      fprintf ( 1, 'error! Not a Line\n' );
    end

  dist = ( a * p(1) + b * p(2) + c ) / sqrt ( a * a + b * b );
```

Perpendicular Distance from a Point to a Line (cont.)

For the parametric form,

$$\begin{aligned} x &= x_0 + ft \\ y &= y_0 + gt \end{aligned}$$

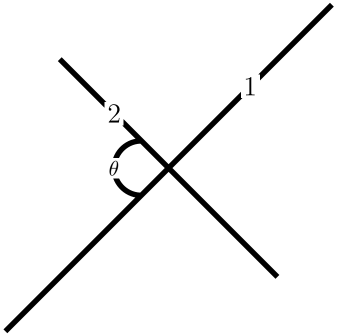
things are little more complicated, line_par_point_dist_2d:

```
dx =   g * g * ( p(1) - x0 ) - f * g * ( p(2) - y0 );
dy = - f * g * ( p(1) - x0 ) + f * f * ( p(2) - y0 );

dist = sqrt ( dx * dx + dy * dy ) / ( f * f + g * g );
```

Furthermore, the value of parameter, t, where the point intersects the line is given by:

```
t =  (f*( p(1) - x0) + g*(p(2) - y0) )/ ( f * f + g * g );
```

Angle Between Two Lines



For the implicit form $a_i x + b_i y + c = 0$ for lines $i = 1, 2$, the angle between two *normalised* lines is:

$$\theta = \cos^{-1}(a_1 a_2 + b_1 b_2)$$

For the *normalised* parametric form:

$$\begin{aligned} x &= x_{0_i} + f_i t \\ y &= y_{0_i} + g_i t \end{aligned}$$

The angle is: $\theta = \cos^{-1}(f_1 f_2 + g_1 g_2)$

Angle Between Two Unnormalised Lines

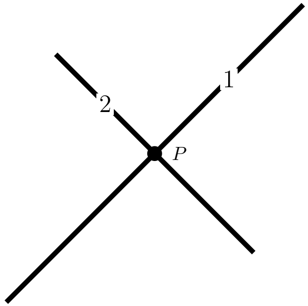
For unnormalised forms it is quicker to compute (rather normalise each seperately) as follows:

$$\theta = \cos^{-1} \frac{a_1 a_2 + b_1 b_2}{\sqrt{(a_1^2 + b_1^2)(a_2^2 + b_2^2)}} \quad \text{(implicit)}$$

$$\theta = \cos^{-1} \frac{f_1 f_2 + g_1 g_2}{\sqrt{(f_1^2 + g_1^2)(f_2^2 + g_2^2)}} \quad \text{(parametric)}$$

MATLAB functions to perform these operation are lines_imp_angle_2d and lines_par_angle_2d.m

Intersection Between Two Lines (Implicit)



For the implicit form $ax_i + by_i + c = 0$ for lines $i = 1, 2$, the intersection between two lines at point $P(x, y)$ is the solution of the two simultaneous equations for x and y. This is given by the following MATLAB code, lines_imp_int_2d.m:

```
det = a1*b2 - a2*b1
if  (abs(det) <  thresh)
% lines are parallel
end

p(1) = (b1*c2 - b2*c1)/det;
p(2) = (a2*c1 - a1*c2)/det;
```

Note: lines_imp_int_2d.m actually solve the system of equation using MATLAB linear equation solver r8mat_solve().

Intersection Between Two Lines (Parametric)

For the *normalised* parametric form:

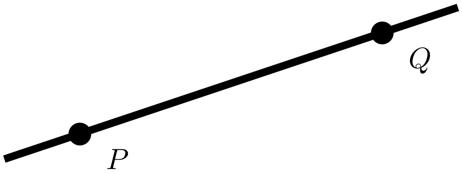
$$\begin{aligned} x &= x_{0_i} + f_i t \\ y &= y_{0_i} + g_i t \end{aligned}$$

The point of intersection is lines_par_int_2d.m:

```
det = f2 * g1 - f1 * g2;

  if ( det == 0.0 )
   % lines are parallel
  else
    t1 = ( f2 * ( y02 - y01 ) - g2 * ( x02 - x01 ) ) / det;
    t2 = ( f1 * ( y02 - y01 ) - g1 * ( x02 - x01 ) ) / det;
    pi(1) = x01 + f1 * t1;
    pi(2) = y01 + g1 * t1;
  end
```

`t1` and `t2` give the parameter values for each line. We only really need to compute one of `t1` and `t2` in most cases.

Line Through Two Points (parametric form)



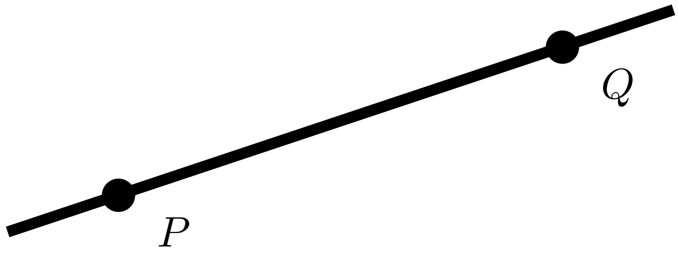
The parametric form of a line through two points, $P(x_p, y_p)$ and $Q(x_q, y_q)$ comes readily from the vector form of line:

- Set base to point P
- Vector along line is $(x_q - x_p, y_q - y_p)$
- The equation of the line is:

$$\begin{aligned} x &= x_p + (x_q - x_p)t \\ y &= y_p + (y_q - y_p)t \end{aligned}$$

- In the above, $t = 0$ gives P and $t = 1$ gives Q
- Normalise if necessary.

Line Through Two Points (implicit form)

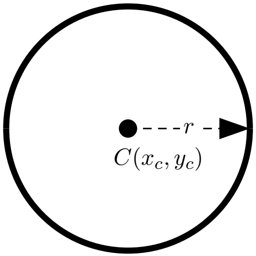


The implicit form of a line through two points, $P(x_p, y_p)$ and $Q(x_q, y_q)$ comes readily from the parametric converted to the implicit form as before:

$$(y_q - y_p)x + (x_q - x_p)y + (x_p y_q - x_q y_p) = 0$$

- This equation is not initially normalised

Circles



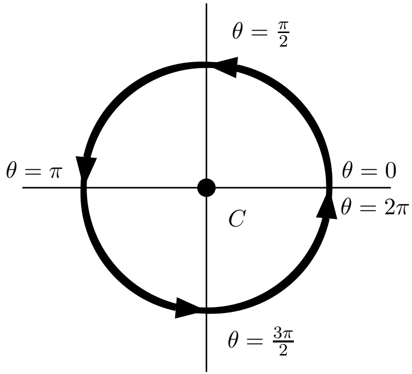
The implicit equation of a circle is the standard formula:

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$$

where the centre of the circle is $C(x_c, y_c)$ and r is the radius of the circle.

This form is commonly used for whole circles.

Circle (parametric form)



The parametric equation of a circle is given by:

$$\begin{aligned} x &= x_c + r\cos(\theta) \\ y &= y_c + r\sin(\theta) \end{aligned}$$

- Parameterisation in terms of angle subtended at the circle centre, C.

MATLAB Circle code

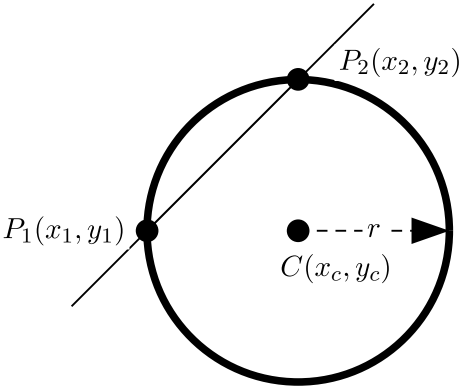
To create n points, `p`, equally space on cirlce of `centre` and radius, `r`:

Implicit form, circle_imp_points_2d:

```
for i = 1 : n
    theta = ( 2.0 * pi * ( i - 1 ) ) / n;
    p(1,i) = center(1) + r * cos ( theta );
    p(2,i) = center(2) + r * sin ( theta );
  end
```

Parametric form, is similar.

Intersections of a Line and a Circle



This problem is most easily solved if the circle is in implicit form:

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$$

and the line is parametric:

$$\begin{aligned} x &= x_0 + ft \\ y &= y_0 + gt \end{aligned}$$

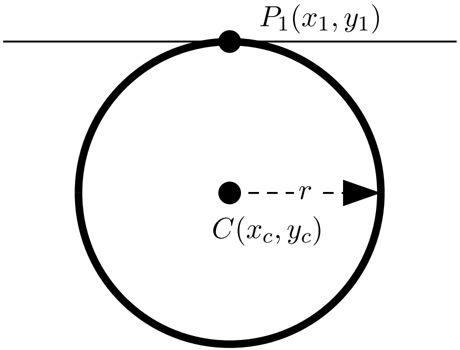
Intersections of a Line and a Circle

Substituting for (parametric line) x and y into the circle equation gives a quadratic equation in t:

- Two roots of which give points on the line where cuts the circle.

$$t = \frac{f(x_c - x_0) + g(y_c - y_0) \pm \sqrt{r^2(f^2+g^2) - (f(y_c - y_0) - g(x_c - x_0))^2}}{(f^2+g^2)}$$

- The roots maybe ***coincident*** if the line is tangential to the circle.



- If roots are ***imaginary*** then there is no intersection.

Intersections of a Line and a Circle (MATLAB)

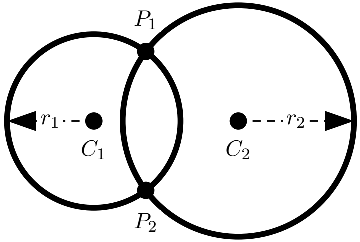
This is now a straightforward implementation in MATLAB, circle_imp_line_par_int_2d.m:

```
root = r * r * ( f * f + g * g ) - ( f * ( center(2) - y0 ) ...
    - g * ( center(1) - x0 ) ).^2;
 if ( root < 0.0 )
    num_int = 0;

  elseif ( root == 0.0 )
    num_int = 1;

    t = ( f * ( center(1) - x0 ) + g * ( center(2) - y0 ) ) / ( f * f + g * g );
    p(1,1) = x0 + f * t;
    p(2,1) = y0 + g * t;

  elseif ( 0.0 < root )
    num_int = 2;

    t = ( ( f * ( center(1) - x0 ) + g * ( center(2) - y0 ) ) ...
      - sqrt ( root ) ) / ( f * f + g * g );

    p(1,1) = x0 + f * t;
    p(2,1) = y0 + g * t;

    t = ( ( f * ( center(1) - x0 ) + g * ( center(2) - y0 ) ) ...
      + sqrt ( root ) ) / ( f * f + g * g );

    p(1,2) = x0 + f * t;
    p(2,2) = y0 + g * t;
  end
```

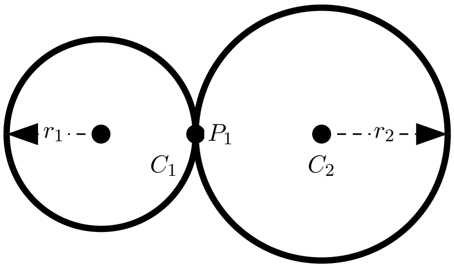
Intersections of Two Circles

Two circles may have intersections at:

- Two points



- One point at a common tangent

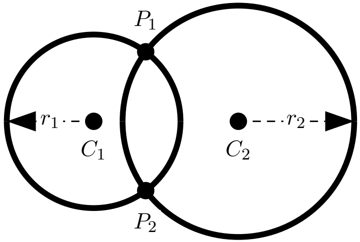


- Or they may not intersect at all.

Intersections of Two Circles

- A line between the two circles' centres and a line between the the two points of intersection make a right angle (or the tangent point's line).
- So we can find the (not normalised) implicit line between the two points of intersection $ax + by + c = 0$:

$a = x_2 - x_1, b = y_2 - y_1$ and (since $\overline{P_1P_2}$ is orthogonal to $\overline{C_1C_2}$).



To obtain c, note the distance from C_1 to $\overline{P_1P_2}$ (let $\theta = \angle P_1C_1C_2$, $d = |C_1C_2| = \sqrt{a^2 + b^2}$).

$$-r_1 \cos\theta = \frac{ax_1 + by_1 + c}{d}$$

$\cos\theta$ can be derived with law of cosines as

$$r_2^2 = r_1^2 + d^2 - 2r_1 d\cos\theta$$

So

$$\cos\theta = \frac{r_1^2 + d^2 - r_2^2}{2r_1 d}$$

$$-r_1\frac{r_1^2 + d^2 - r_2^2}{2r_1 d} = \frac{ax_1 + by_1 + c}{d}$$

We have

$$c = \frac{(r_2^2 - r_1^2) - (x_2 - x_1)^2 - (y_2 - y_1)^2}{2} - x_1(x_2 - x_1) - y_1(y_2 - y_1)$$

- Now solve as for a circle and line intersection as before.

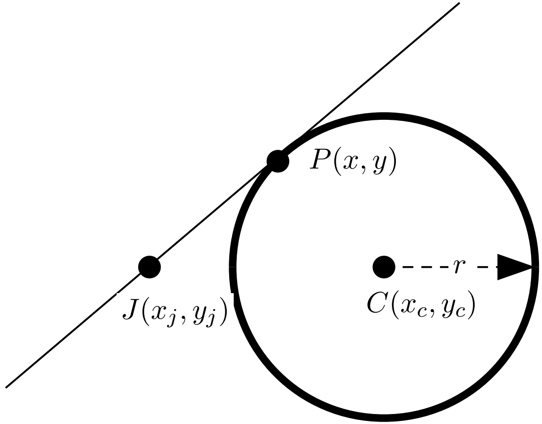
Intersections of Two Circles (MATLAB)

The intersection of two circles is implemented as follows in MATLAB, circles_imp_int_2d.m:

```
 tol = eps; % some small value tolerance
 p(1:dim_num,1:2) = 0.0;;
%
%  Take care of the case in which the circles have the same center.
%
  t1 = ( abs ( center1(1) - center2(1) ) ...
       + abs ( center1(2) - center2(2) ) ) / 2.0;
  t2 = ( abs ( center1(1) ) + abs ( center2(1) ) ...
       + abs ( center1(2) ) + abs ( center2(2) ) + 1.0 ) / 5.0;

  if ( t1 <= tol * t2 )
    t1 = abs ( r1 - r2 );
    t2 = ( abs ( r1 ) + abs ( r2 ) + 1.0 ) / 3.0;

   if ( t1 <= tol * t2 )
      num_int = 3;
    else
      num_int = 0;
    end
    return
  end
```

```
  distsq = sum ( ( center1(1:2) - center2(1:2) ).^2 );
  root = 2.0 * ( r1 * r1 + r2 * r2 ) * distsq - distsq * distsq ...
    - ( r1 - r2 ) * ( r1 - r2 ) * ( r1 + r2 ) * ( r1 + r2 );

  if ( root < -tol )
   % Circles DO NOT Intersect
    num_int = 0; % No Solution
    return
  end

  sc1 = ( distsq - ( r2 * r2 - r1 * r1 ) ) / distsq;
  if ( root < tol )
   % Circles touch at P(x,y)
   num_int = 1; % One solution
    p(1:dim_num,1) = center1(1:dim_num)' ...
      + 0.5 * sc1 * ( center2(1:dim_num) - center1(1:dim_num) )';
    return
  end

  num_int = 2;  % Two solutions
  sc2 = sqrt ( root ) / distsq;

  p(1,1) = center1(1) + 0.5 * sc1 * ( center2(1) - center1(1) ) ...
                      - 0.5 * sc2 * ( center2(2) - center1(2) );
  p(2,1) = center1(2) + 0.5 * sc1 * ( center2(2) - center1(2) ) ...
                      + 0.5 * sc2 * ( center2(1) - center1(1) );

  p(1,2) = center1(1) + 0.5 * sc1 * ( center2(1) - center1(1) ) ...
                      + 0.5 * sc2 * ( center2(2) - center1(2) );
  p(2,2) = center1(2) + 0.5 * sc1 * ( center2(2) - center1(2) ) ...
                      - 0.5 * sc2 * ( center2(1) - center1(1) );

  return
end
```

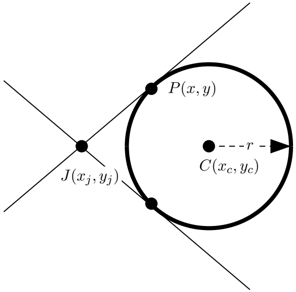
Tangents from a Point to a Circle



Demo: see Java Applet Demo

- If the point is outside the circle there are ***two tangents***
- If it lies on the circumference of the circle there is ***one tangent***
- If it lies inside circle there is ***no tangent***

Tangents from a Point to a Circle



We wish to find the implicit equation of the tangent

$$ax + by + c = 0$$

The coefficients a and b are obtained from:

$$a = \frac{\mp r(x_c - x_j) - (y_c - y_j)\sqrt{(x_c - x_j)^2 + (y_c - y_j)^2 - r^2}}{(x_c - x_j)^2 + (y_c - y_j)^2}$$

$$b = \frac{\mp r(y_c - y_j) + (x_c - x_j)\sqrt{(x_c - x_j)^2 + (y_c - y_j)^2 - r^2}}{(x_c - x_j)^2 + (y_c - y_j)^2}$$

c then obtained from the fact that the tangent passes through J:

$$c = -ax_j - by_j$$

Curves other than Circles

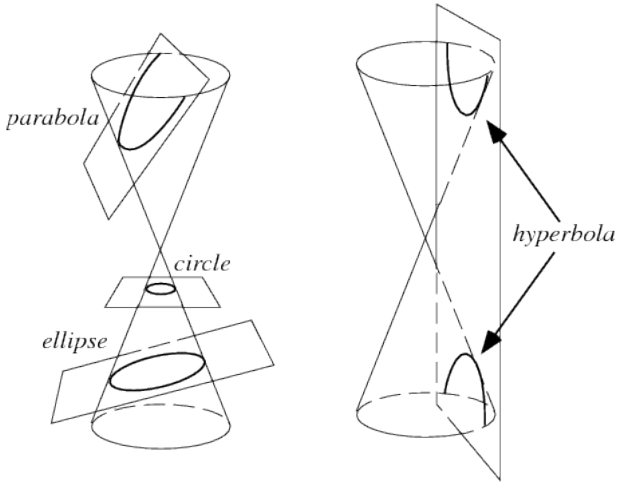
General Implicit Quadratic Equations

The **general implicit quadratic equation** form is:

$$ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0$$

Such general quadratics are called ***conic sections*** as they can represent all the shapes that can be cut from a cone with a plane.

- Ellipse
- Parabola
- Hyperbola



Ellipse, Parabola or Hyperbola

By considering three values we can determine if the quadratic is either an ellipse, parabola or hyperbola:

$$\begin{aligned}
\Delta &= a(cf - e^2) + b(bf - de) + d(be - dc) \\
\delta &= ac - b^2 \\
S &= a + c
\end{aligned}$$

- If $\Delta = 0$ then the quadratic is **degenerate** and represents two straight lines (which may not always exist!)
- ***Otherwise***:
  - If $\delta < 0$ quadratic is a hyperbola.
  - If $\delta = 0$ quadratic is a parabola.
  - If $\delta > 0$ quadratic is an ellipse ***if*** $\Delta S < 0$.

Parametric Polynomials

Implicit equations of higher order than a quadratic ($x^3, x^2y \ldots$) are not generally useful because of problems in solving them for x and y.

Parametric equations extended to higher order do not suffer such problems.

The simplest non-linear parametric curve is the **quadratic**:

$$\begin{aligned} x &= a_1 + b_1 t + c_1 t^2 \\ y &= a_2 + b_2 t + c_2 t^2 \end{aligned}$$

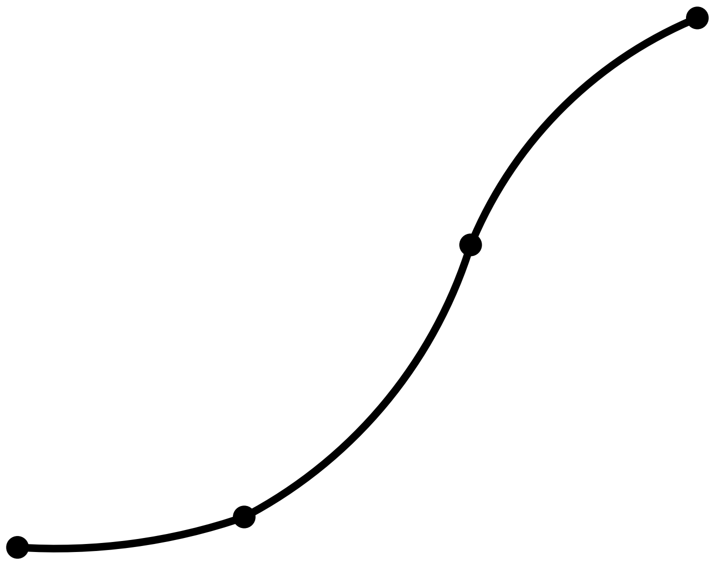
The next form is the **parametric cubic**:

$$\begin{aligned} x &= a_1 + b_1 t + c_1 t^2 + d_1 t^3 \\ y &= a_2 + b_2 t + c_2 t^2 + d_2 t^3 \end{aligned}$$

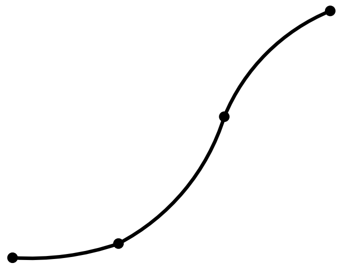
This formulation is easily extended to **3D parametric surfaces** by introducing a z equation component.

Fitting and Interpolation Using Parametric Polynomials

Parametric Polynomials are often used to interpolate data through a set of data points:



Fitting and Interpolation Using Parametric Polynomials

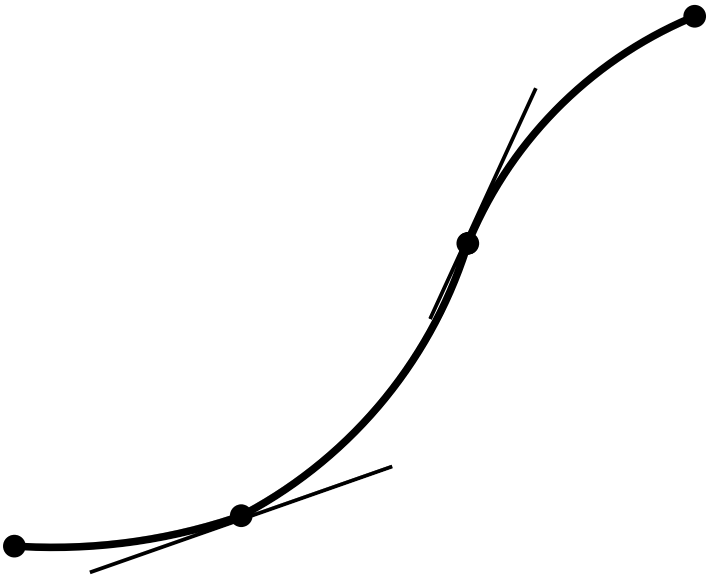


- Choose a value of t which corresponds to each given point, thus determining the order in which points occur on the curve.
- Chosen values of t and corresponding values of x and y substituted at each point, give a set of linear simultaneous equations to solve for parameters, a_i, b_i, c_i *etc.*
- If the order of the curve (highest power of t) is one less than the number of points (3 for quadratic, 4 for cubic *etc.* then the simultaneous equations can be solved.

The above procedure (interpolation through points) is called **Lagrangian Interpolation**. Lagrangian interpolation demo code

Hermite Interpolation

Here we need to introduce and fulfil some slope constraints on the parametric polynomial.



- Slope means gradient or tangent at a point here.

Hermite Interpolation

- We need to compute the **partial derivatives** of the parametric polynomial. To this we differentiate each equation in x and y with respect to t

  For example for a cubic:

$$\begin{aligned} x &= a_1 + b_1 t + c_1 t^2 + d_1 t^3 \\ y &= a_2 + b_2 t + c_2 t^2 + d_2 t^3 \end{aligned}$$

  We get the deriviatives:

$$\begin{aligned} \frac{\partial x}{\partial t} &= b_1 + 2c_1 t + 3d_1 t^2 \\ \frac{\partial y}{\partial t} &= b_2 + 2c_2 t + 3d_2 t^2 \end{aligned}$$

Hermite Interpolation

Some points to note:

- Gradients at each point need to estimated and then they can be substituted into the above equations and solved ***together*** with the original (Lagrangian) point equations.
- It is not necessary to have slope constraints at every point — position and slope constraints can be mixed as required (so long as we have enough to satisfy the simultaneous equations)
- If the points are spread evenly then the point can be parameterised at equal intervals of t.
- Setting start $t = 0$ and end $t = 1$ and having proportional values of t for unequal steps of t is a common approach.
- In Hermite interpolation there are no unique values for $\frac{\partial x}{\partial t}$ and $\frac{\partial y}{\partial t}$ for a required $\frac{dx}{dy}$, only the ratio $\frac{\partial x}{\partial t}$ / $\frac{\partial y}{\partial t}$ must correspond. This can introduce some unwanted results.
- As the order of the curves becomes higher, undesired oscillations, **waviness**, tends to occur. Higher than order 5 or 6 is not common.
- There are more elaborate parametric curve representation — Bézier curves, Spline curves.

MATLAB Hermite spline interpolation example, hermite interpolation demo code