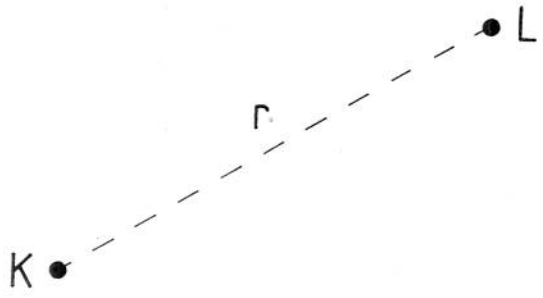
1
Points and lines

1.1 Distance between Two Points



Given two points, K and L, an elementary application of Pythagoras' theorem gives the distance between them as

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

which is simply coded:

```
XLK = XL - XK
YLK = YL - YK
RSQ = XLK*XLK + YLK*YLK
R = SQRT( RSQ )
```

However, this is not a cheap expression to compute, and many applications require a very large number of distance calculations. In most cases, these are repeated *comparisons* of inter-point distances with a reference distance. This reference distance is commonly either a fixed value, or a minimum or maximum that is being updated as an algorithm proceeds.

The simplest way of increasing the efficiency of comparisons is to compare values of r^2 instead of r. A reference distance must, of course, be squared before comparisons start, and it may be advantageous to use squared distances throughout a data structure.

If a large number of comparisons are being made, of which many will lead to the rejection of distances grossly outside the distance of interest, then pre-testing can be performed which will avoid the computation of squares as well as the square root for these cases.

If $|x_K - x_L|$ or $|y_K - y_L|$ is greater than a reference distance, then r must be as well, and further comparison can be avoided:

```
XLK = XL - XK
IF (ABS(XLK).GE.RREF) THEN

        ..... don't bother to continue.

ELSE
        YLK = YL - YK
        IF (ABS(YLK).GE.RREF) THEN

                ..... don't bother to continue.

        ELSE
                RSQ = XLK*XLK + YLK*YLK
                IF (RSQ.GE.RREFSQ) THEN

                        ..... don't bother to continue.

                ELSE

                        ..... RSQ is within the square of the
                              reference distance.

                ENDIF
        ENDIF
ENDIF
```

Note that both the reference distance, r_{ref}, and its square must be stored (and possibly, depending on the problem, updated).

If, on the other hand, the requirement is to reject quickly distances less than the reference distance, then we can use the fact that r must be less than $|x_L - x_K| + |y_L - y_K|$.

If the points under consideration are known to be (or are suspected of being) on a grid, then it may be worth dealing with the trivial cases where the two points are on the same horizontal or vertical grid line separately.

Finally, when working out the square root of the sum of two squares, it is slightly less efficient, but more numerically stable to use the code

```
TEMP1 = ABS(XLK)                    Temporary store for
TEMP2 = ABS(YLK)                    absolute values.

AMN = AMIN1(TEMP1,TEMP2)            Find the smaller and
AMX = AMAX1(TEMP1,TEMP2)            the larger.

DIV = AMN/AMX                       Prevent terms in the
R = AMX*SQRT(1.0+DIV*DIV)           bracket getting too big.
```

than the code given near the beginning of this section. This is probably only worth bothering with if you have very large and very small distances to deal with. This point is worth remembering throughout the rest of the book, however, as square rooting a sum of squares often needs to be done, and, though we use the simpler form, the above code might be needed in exceptional cases.

1.2 Equations of a Line

The choice of algebraic expressions to represent straight lines is important. Many operations are algebraically considerably simpler with one form than with another, and hence yield shorter and quicker code. In general one form should be used throughout an application, and conversions performed as necessary.

The Explicit Form

The best known line equation is this one:

$$y = mx + c$$

Because the value of m becomes very large as the line comes near to vertical, this formulation is practically useless for computation, as operations based upon it would be riddled with special cases.

We have used the common notation for this equation, as it will not be mentioned again.

The Implicit Form

This is a more stable version of the line definition above:

$$ax + by + c = 0$$

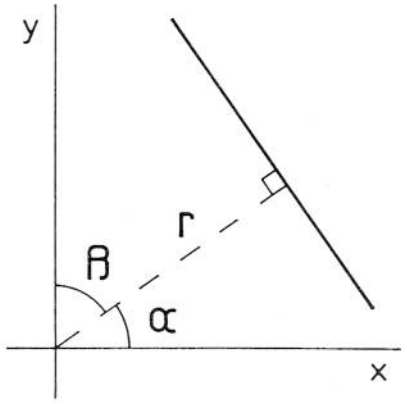
As it stands, this equation can be multiplied through by any non-zero constant without altering its meaning. It is made more useful, and potential numerical problems are avoided, by putting it into *canonical* or *normalised* form by imposing the constraint:

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

This is most simply achieved by multiplying through by:

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

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

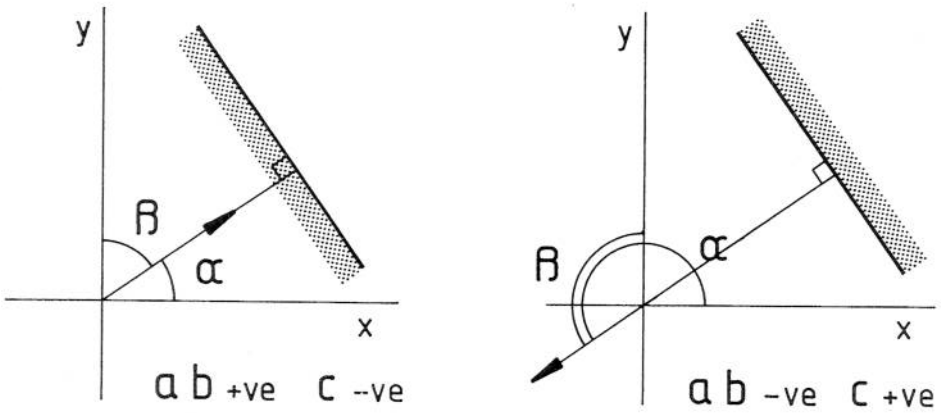


$$a = \cos\alpha\ , \quad b = \cos\beta\ , \quad c = -r$$

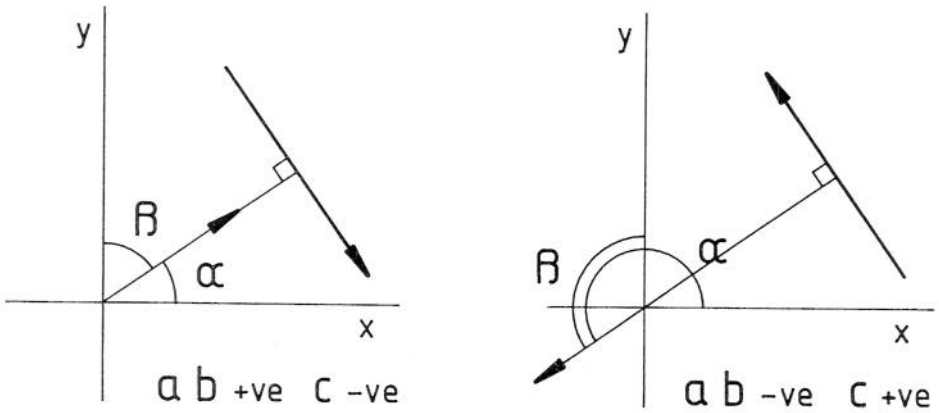
The entire equation may still be multiplied by −1 without violating the normalising $a^2 + b^2 = 1$ condition. We may choose to prefer neither form, or to have a convention that c is always positive (although this still leaves ambiguity when c = 0 and the line goes through the origin). Alternatively we may use the sign of c to convert the line to the boundary of a region in the plane, with an inside and an outside, or to impose a direction on the line. Where the line is a boundary, it may be called a *linear half-plane*, because it bisects the plane into two semi-infinite areas.

The notion of side may be expressed by a convention that the vector (see Section 6.1) formed from

the direction cosines always points towards the outside (or the inside) of the region. Direction along the line may be specified as either right- or left-handed from the normal vector.



convention: normal points towards outside



convention: line direction is to right of normal vector

The Parametric Form

This form of line equation consists of two equations which give x and y in terms of a third variable, the parameter, t:

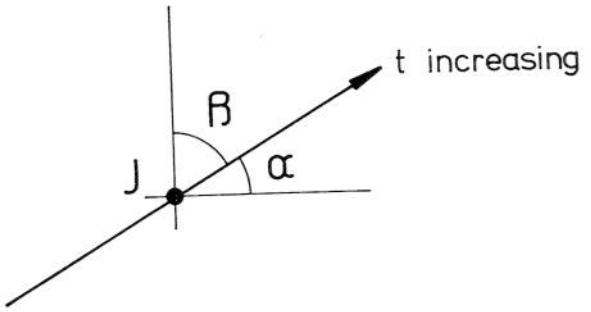
$$x = x_0 + ft$$

$$y = y_0 + gt$$

The convention x_0 and y_0 for the constant terms is adopted because it will readily be seen that (x_0, y_0) is the point on the line corresponding to a zero value of the parameter, t.

These equations have four constant terms, as opposed to three in the implicit form. The extra freedom allows us to specify just how the parameter varies along the line. There are two useful conventions. The first is that the parameter varies between 0 and 1 over a given line segment. This formulation is dealt with in the section on a line between two points (Section 1.6). The second convention (the normalised form) makes t correspond to real distance along the line.

A line through a point J, making angles α and β with the x and y axes respectively



has the parametric equations:

$$x = x_J + (\cos\alpha)t$$

$$y = y_J + (\cos\beta)t$$

In general any parametric line can be normalised by dividing the coefficients of t (but *not* the constant terms) by $\sqrt{(f^2 + g^2)}$, in a similar manner to the way that the implicit form was normalised.

Conversion from Implicit to Parametric Form.

A general, not necessarily normalised, implicit line ax + by +c = 0 is conveniently parameterised as:

$$x = \frac{-ac}{(a^2 + b^2)} + bt$$

$$y = \frac{-bc}{(a^2 + b^2)} - at$$

This operation can be coded:

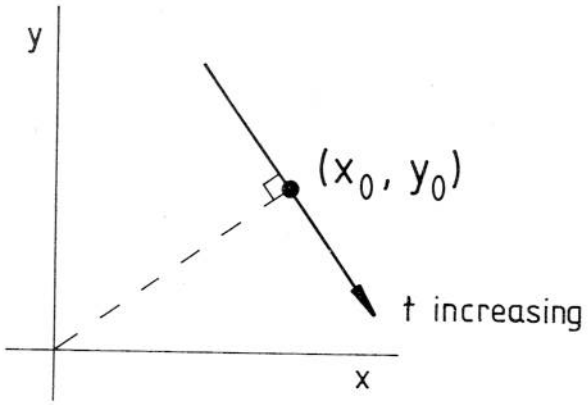
```
ROOT = 1.0/(A*A + B*B)
FACTOR = -C*ROOT
XO = A*FACTOR
YO = B*FACTOR
```

```
ROOT = SQRT( ROOT )
F = B*ROOT
G = -A*ROOT
```

(This assumes that the data structure is not corrupted, and that A and B are not both zero. It may be worth checking for this.)

This conversion also normalises the equation. If the implicit equation to be converted is known to be normalised already, then the division by $(a^2 + b^2)$ can be omitted.

The point (x_0, y_0) in this parametric form is the point on the line where the line meets the normal to the origin. The direction of parameterisation has t increasing in the line direction corresponding to the right handed convention in the implicit form. For instance, if a and b were positive, and c was negative, the parameter would behave as shown below:



Conversion from Parametric to Implicit Form

A parametric line

$$x = x_0 + ft$$

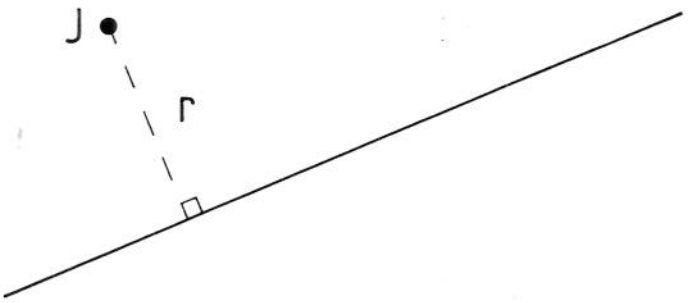
$$y = y_0 + gt$$

can readily be converted to the implicit form:

$$-gx + fy + (x_0 g - y_0 f) = 0$$

which is only normalised if the parametric form was also normalised. The increasing t direction of the parametric line will again be to the right of the normal vector from the origin to the line.

1.3 Distance from a Point to a Line



If the line is in its implicit form and has the equation

$$ax + by + c = 0$$

and the point is (x_J, y_J), then the shortest distance of that point from the line, r, is the length of a perpendicular drawn from the point to the line. r^2 is given by:

```
ABSQ = A*A + B*B
IF (ABSQ.LT.ACCY) THEN

        ..... The line is improperly defined

ELSE
        SR = A*XJ + B*YJ + C
        RSQ = SR*SR/ABSQ
ENDIF
```

If the equation of the line is in its normalised form, ABSQ will be 1.0 and this code can be simplified to the single statement

```
SR = A*XJ + B*YJ + C
```

The sign of SR indicates on which side of the line (x_J, y_J) lies. Positive values indicate that the point is on the side of the line in the direction that the vector (a,b) is pointing (see Section 6.1). Negative values indicate that it is on the other side. If this information is irrelevant, the function ABS(SR) should be calculated.

If the line is in parametric form

$$x = x_0 + ft$$

$$y = y_0 + gt$$

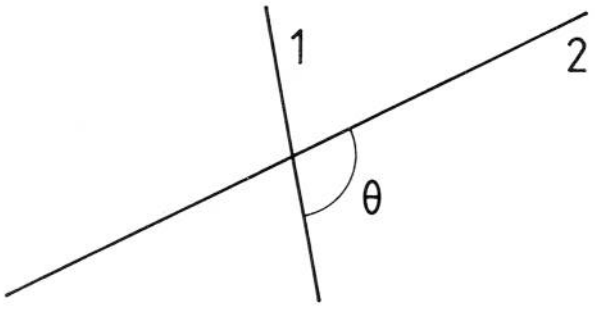
the code becomes a little more complicated:

```
FSQ = F*F
GSQ = G*G
FGSQ = FSQ + GSQ
IF (FGSQ.LT.ACCY) THEN

        ..... The line is improperly defined

ELSE
        XJO = XJ - XO
        YJO = YJ - YO
        FG = F*G
        FINV = 1.0/FGSQ
        DX = GSQ*XJO - FG*YJO
        DY = FSQ*YJO - FG*XJO
        RSQ = (DX*DX + DY*DY)*FINV*FINV
ENDIF
```

The value of the parameter, t, at the closest point on the line to (x_J, y_J) is given by:

```
TJ = (F*XJO + G*YJO)*FINV
```

1.4 Angle between Two Lines



The angle between two lines is found from their direction cosines. If the two line equations are normalised and are

$$a_1 x + b_1 y + c_1 = 0$$

$$a_2 x + b_2 y + c_2 = 0$$

then the angle between them is

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

and for two normalised parametric lines

$$x = x_1 + f_1 s$$

$$y = y_1 + g_1 s$$

$$x = x_2 + f_2 t$$

$$y = y_2 + g_2 t$$

the angle is:

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

If the lines are not normalised then, rather than normalising each separately for just this one operation, it is quicker to use the expressions:

ARCOS

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

(implicit)

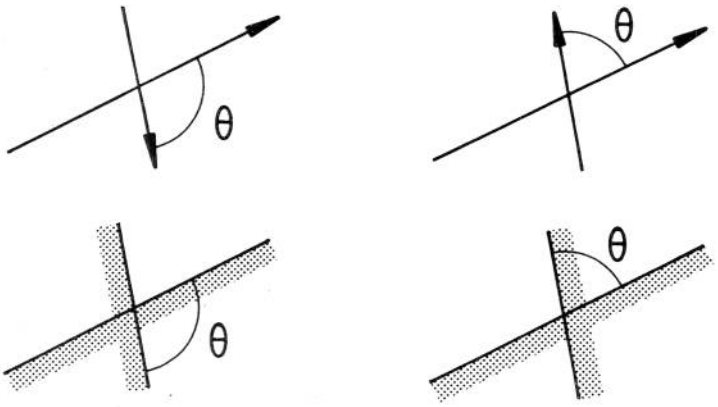
and $$\theta = \cos^{-1}\frac{f_1 f_2 + g_1 g_2}{\sqrt{[(f_1^2 + g_1^2)(f_2^2 + g_2^2)]}}$$

(parametric)

These use one square root each, instead of the two required to normalise each equation separately, and, in the case of the implicit form, the unnecessary normalisation of the constant term is also avoided.

The FORTRAN ACOS (arc cosine) function returns angles in the range 0 to π. If the acute angle

between the lines is required, then values of θ that are greater than $\pi/2$ must be subtracted from π. Alternatively, if the lines have a direction associated with them, or are half-planes, the value of θ is the angle through which one line would have to be turned in order to correspond exactly with the other.



Some elderly FORTRAN compilers have no ACOS function. An ACOS function can be written in terms of the ubiquitous ATAN2 function.

```
FUNCTION ACOS(X)
     ACOS = ATAN2(SQRT(1.0 - X*X), X)
RETURN
END
```

Both this function and the generic ACOS will give an error if the argument is greater than 1.0. In the function above this will take the form of an attempt to take the square root of a negative number. The generic ACOS routine will also give an error if its argument is less than zero, whereas the code above will return a negative angle for arguments between -1.0 and 0.0, and only give an error for arguments less than -1.0.

In any case, it is best to ensure that the argument is in the range 0.0 to 1.0, as numerical errors may produce values just outside this range even if the original data were correct. The code:

```
THETA = ACOS(AMIN1(1.0,AMAX1(0.0,X)))
```

solves this problem, but gross errors arising from corrupted data must be detected separately.

If it is required only to find pairs of lines, the angles between which fall within a particular range of values, it will be more efficient to take the cosines of the limiting values and compare these with the products of the direction cosines directly. This avoids many calls to the costly ACOS function. For instance, lines within 1^{o} of being parallel will have a sum of products of direction cosines within the range -1.0 to -0.99985 [$\cos(179^{o})$] or 0.99985 to 1.0. Lines within 1^{o} of perpendicularity will have values in the range -0.017453 [$\cos(91^{o})$] and 0.017453 [$\cos(89^{o})$].

This is an appropriate place to note that we recommend that angles are avoided wherever possible, because of the cost of trigonometric functions and the potential numerical instabilities that they introduce. However, especially when angles are required for input or output, the use of trigonometric functions cannot be avoided. It can be minimised by using the following relations between sines, cosines, tangents, and tangents of half-angles (used in describing arcs).

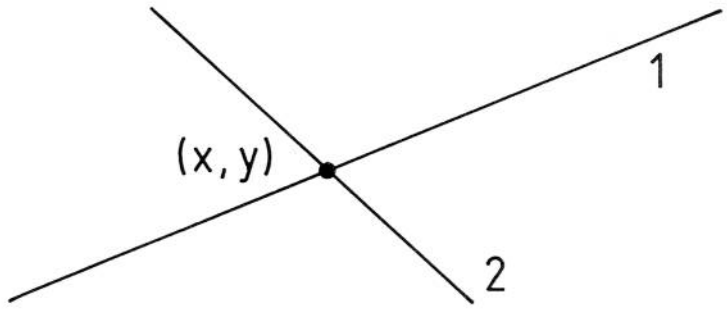
$$\sin\theta = \sqrt{(1 - \cos^2\theta)} = \frac{\tan\theta}{\sqrt{(1 + \tan^2\theta)}} = \frac{2\tan(\theta/2)}{1 + \tan^2(\theta/2)}$$

$$\cos\theta = \sqrt{(1 - \sin^2\theta)} = \frac{1}{\sqrt{(1 + \tan^2\theta)}} = \frac{1 - \tan^2(\theta/2)}{1 + \tan^2(\theta/2)}$$

$$\tan\theta = \frac{\sin\theta}{\sqrt{(1 - \sin^2\theta)}} = \frac{\sqrt{(1 - \cos^2\theta)}}{\cos\theta} = \frac{2\tan(\theta/2)}{1 - \tan^2(\theta/2)}$$

$$\tan(\theta/2) = \frac{1 - \sqrt{(1 - \sin^2\theta)}}{\sin\theta} = \sqrt{\frac{1 - \cos\theta}{1 + \cos\theta}} = \frac{\sqrt{(1 + \tan^2\theta)} - 1}{\tan\theta}$$

1.5 Intersection of Two Lines



$a_2 x + b_2 y + c_2 = 0$

and they intersect at the point (x, y) then the solution is simply coded:

```
DET = A1*B2 - A2*B1
IF (ABS(DET).LT.ACCY) THEN

        ..... The two lines are parallel

ELSE

        DINV = 1.0/DET
        X = (B1*C2 - B2*C1)*DINV
        Y = (A2*C1 - A1*C2)*DINV
ENDIF
```

If one equation is in its implicit form and the other is parametric the solution is a little more complicated.

$ax + by + c = 0$ *Implicit line equation*

$x = x_0 + ft$ *Parametric*

$y = y_0 + gt$ *line equation*

This gives the code:

```
DET = A*F + B*G
IF (ABS(DET).LT.ACCY) THEN

        ..... The two lines are parallel

ELSE

        DINV = 1.0/DET
        PDET = XO*G - YO*F
        X = (B*PDET - C*F)*DINV
        Y = -(A*PDET + C*G)*DINV
ENDIF
```

At the point of intersection:

$$t = \frac{-(c + ax_0 + by_0)}{(af + bg)}$$

If both lines are in parametric form

$$x = x_1 + f_1 s$$

$$y = y_1 + g_1 s$$

and $$x = x_2 + f_2 t$$

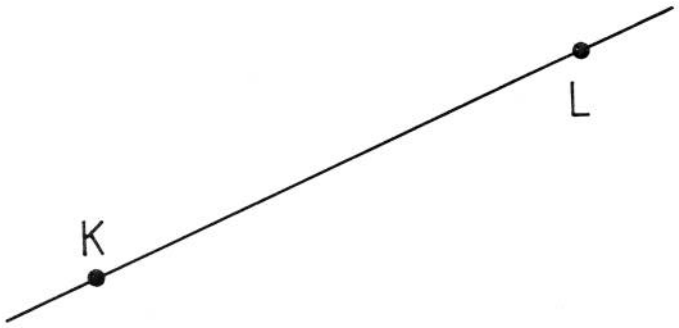
$$y = y_2 + g_2 t$$

the solution becomes

```
F1G2 = F1*G2
F2G1 = F2*G1
DET = F2G1 - F1G2
IF (ABS(DET).LT.ACCY) THEN

        ..... The two lines are parallel

ELSE
        Y21 = Y2 - Y1
        X21 = X2 - X1
        DINV = 1.0/DET
        S = (F2*Y21 - G2*X21)*DINV
        T = (F1*Y21 - G1*X21)*DINV
        X = X1 + F1*S
        Y = Y1 + G1*S
ENDIF
```

where S and T are the parameter values at the point of intersection. If only the coordinates of the point are to be calculated there is no need to include the line that calculates T.

1.6 Line through Two Points



This is the second common way of specifying a parametric line (see Section 1.2). The line is specified so that t = 0 at the first point and t = 1 at the second. The equations for this form of line are:

$$x = x_K + (x_L - x_K)t$$

$$y = y_K + (y_L - y_K)t$$

This is not normalised unless the distance between K and L is 1, of course. The implicit form is found from the conversion formula given in section 1.2, and cannot be simplified beyond this point:

$$(y_K - y_L)x + (x_L - x_K)y + (x_K y_L - x_L y_K) = 0$$

In general this equation is not normalised either.

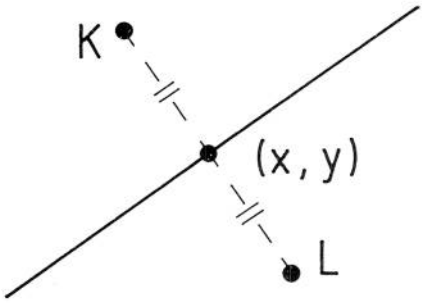
Both forms can be normalised by dividing the entire implicit equation, or the terms in t of the parametric equation, by the distance from K to L.

For example, the coefficients of a normalised implicit equation can be computed as follows:

```
XLK = XL - XK
YLK = YL - YK
RSQ = XLK*XLK + YLK*YLK
IF (RSQ.LT.ACCY) THEN

      ..... The points coincide

ELSE
      RINV = 1.0/SQRT(RSQ)
      A = -YLK*RINV
```

```
        B = XLK*RINV
        C = (XK*YL - XL*YK)*RINV
    ENDIF
```

1.7 Line Equidistant from Two Points



Both the implicit and the parametric forms of this line come out most simply in forms that are not normalised. The implicit equation is:

$$(x_L - x_K)x + (y_L - y_K)y - \frac{1}{2}[(x_L^2 + y_L^2) - (x_K^2 + y_K^2)] = 0$$

The normal vector to the line will always point from K to L.

The parametric form is chosen so that the t = 0 point is the mid-point of the line joining K and L.

$$x = \frac{(x_K + x_L)}{2} - (y_L - y_K)t$$

$$y = \frac{(y_K + y_L)}{2} + (x_L - x_K)t$$

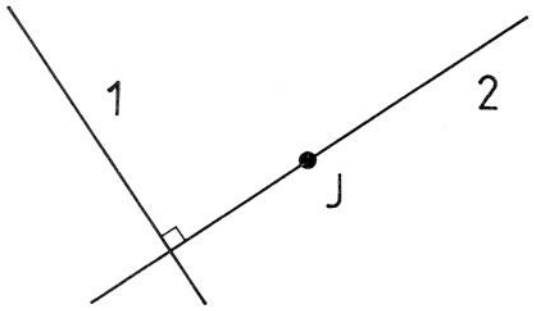
The parameter, t, increases to the right of the line from K to L.

Both forms of equation are normalised by dividing by the distance from K to L: the entire implicit equation or just the terms in t of the parametric equations.

For example, the terms of a normalised parametric representation could be computed as follows:

```
XLK = XL - XK
YLK = YL - YK
RSQ = XLK*XLK + YLK*YLK
IF (RSQ.LT.ACCY) THEN

        ..... the two points coincide.

ELSE
        RINV = 1.0/SQRT(RSQ)
        XO = 0.5*(XK + XL)
        F = -YLK*RINV
        YO = 0.5*(YK + YL)
        G = XLK*RINV
ENDIF
```

1.8 Normal to a Line Through a Point



If the line equation is in implicit form

$$a_1 x + b_1 y + c_1 = 0$$

and the point is (x_J, y_J), and the equation of a line at right angles to the first line passing through that point that we want to find will be

$$a_2 x + b_2 y + c_2 = 0$$

then the terms of that second equation are:

$$a_2 = b_1$$

$$b_2 = -a_1$$

$$c_2 = a_1 y_J - b_1 x_J$$

Note that if the original line equation was in its normalised form this property is transferred to the second line.

If the line is in parametric form

$$x = x_1 + f_1 t$$

$$y = y_1 + g_1 t$$

then the normal to it is the line

$$x = x_J - g_1 s$$

$$y = y_J + f_1 s$$