Aviation Formulary V1.24

By Ed Williams

Table of Contents

Introduction

This introduction is written for pilots (and others) who are interested in great circle navigation and would like to know how to compute courses, headings and other quantities of interest. These formulae can be programmed into your calculator or spreadsheet. I'll attempt to include enough information that those familiar with plane trigonometry can derive additional results if required.

It is a well known that the shortest distance between two points is a straight line. However anyone attempting to fly from Los Angeles to New York on the straight line connecting them would have to dig a very substantial tunnel first. The shortest distance, following the earth's surface lies vertically above the aforementioned straight line route. This route can be constructed by slicing the earth in half with an imaginary plane through LAX and JFK. This plane cuts the (assumed spherical) earth in a circular arc connecting the two points, called a great circle. Only planes through the center of the earth give rise to great circles. Any plane will cut a sphere in a circle, but the resulting little circles are not the shortest distance between the points they connect. A little thought will show that lines of longitude (meridians) are great circles, but lines of latitude, with the exception of the equator, are not.

I will assume the reader is familiar with latitude and longitude as a means of designating locations on the earth's surface. For the convenience of North Americans I will take North latitudes and West longitudes as positive and South and East negative. The longitude is the opposite of the usual mathematical convention. True course is defined as usual, as the angle between the course line and the local meridian measured clockwise.

The first important fact to realise is that in general a great circle route has a true course that varies from point to point. For instance the great circle route between two points of equal (non-zero) latitude does not follow the line of latitude in an E-W direction, but arcs towards the pole. It is possible to fly between two points using an unvarying true course, but in general the resulting route differs from the great circle route and is called a rhumb line. Unlike a great circle which encircles the earth, a pilot flying a rhumb line would spiral indefinitely poleward.

Natural questions are to seek the great circle distance between two specified points and true course at points along the route. The required spherical trigonometric formulae are greatly simplified if angles and distances are measured in the appropriate natural units, which are both radians! A radian, by definition, is the angle subtended by a circular arc of unit length and unit radius. Since the length of a complete circular arc of unit radius is 2*pi, the conversion is 360 degrees equals 2*pi radians, or:

```
angle_radians=(pi/180)*angle_degrees
angle_degrees=(180/pi)*angle_radians
```

Great circle distance can be likewise be expressed in radians by defining the distance to be the angle subtended by the arc at the center of the earth. Since by definition, one nautical mile subtends one minute (=1/60 degree) of arc, we have:

```
distance_radians=(pi/(180*60))*distance_nm
distance_nm=((180*60)/pi)*distance_radians
```

In all subsequent formulae all distances and angles, such as latitudes, longitudes and true courses will be assumed to be given in radians, greatly simplifying them, and in applications the above formulae and their inverses are necessary to convert back and forth between natural and practical units. Examples of this process are given later.

Some great circle formulae:

Distance between points

The great circle distance d between two points with coordinates {lat1,lon1} and {lat2,lon2} is given by:

```
d=acos(sin(lat1)*sin(lat2)+cos(lat1)*cos(lat2)*cos(lon1-lon2))
```

A mathematically equivalent formula, which is less subject to rounding error for short distances is:

```
d=2*asin(sqrt((sin((lat1-lat2)/2))^2 +
                 cos(lat1)*cos(lat2)*(sin((lon1-lon2)/2))^2))
```

Course between points

We obtain the initial course, tc1, (at point 1) from point 1 to point 2 by the following. The formula fails if the initial point is a pole. We can special case this with:

```
IF (cos(lat1) < EPS)   // EPS a small number ~ machine precision
  IF (lat1 > 0)
     tc1= pi        //  starting from N pole
  ELSE
     tc1= 0         //  starting from S pole
  ENDIF
ENDIF


IF sin(lon2-lon1)<0
   tc1=acos((sin(lat2)-sin(lat1)*cos(d))/(sin(d)*cos(lat1)))
ELSE
   tc1=2*pi-acos((sin(lat2)-sin(lat1)*cos(d))/(sin(d)*cos(lat1)))
ENDIF
```

An alternative formula, not requiring the pre-computation of d, the distance between the points, is:

```
   tc1=mod(atan2(sin(lon1-lon2)*cos(lat2),
           cos(lat1)*sin(lat2)-sin(lat1)*cos(lat2)*cos(lon1-lon2)), 2*pi)
```

Latitude of point on GC

Intermediate points {lat,lon} lie on the great circle connecting points 1 and 2 when:

```
lat=atan((sin(lat1)*cos(lat2)*sin(lon-lon2)
     -sin(lat2)*cos(lat1)*sin(lon-lon1))/(cos(lat1)*cos(lat2)*sin(lon1-lon2)))
```

(not applicable for meridians. i.e if sin(lon1-lon2)=0)

Lat/lon given radial and distance

A point {lat,lon} is a distance d out on the tc radial from point 1 if:

```
     lat=asin(sin(lat1)*cos(d)+cos(lat1)*sin(d)*cos(tc))
     IF (cos(lat)=0)
        lon=lon1      // endpoint a pole
     ELSE
        lon=mod(lon1-asin(sin(tc)*sin(d)/cos(lat))+pi,2*pi)-pi
     ENDIF
```

This algorithm is limited to distances such that dlon <pi/2, i.e those that extend around less than one quarter of the circumference of the earth in longitude. A completely general, but more complicated algorithm is necessary if greater distances are allowed:

```
     lat =asin(sin(lat1)*cos(d)+cos(lat1)*sin(d)*cos(tc))
     dlon=atan2(sin(tc)*sin(d)*cos(lat1),cos(d)-sin(lat1)*sin(lat))
     lon=mod( lon1-dlon +pi,2*pi )-pi
```

Intersecting radials

Now how to compute the latitude, lat3, and longitude, lon3 of an intersection formed by the crs13 true bearing from point 1 and the crs23 true bearing from point 2:

```
dst12=2*asin(sqrt((sin((lat1-lat2)/2))^2+
```

```
                   cos(lat1)*cos(lat2)*sin((lon1-lon2)/2)^2))
IF sin(lon2-lon1)<0
   crs12=acos((sin(lat2)-sin(lat1)*cos(dst12))/(sin(dst12)*cos(lat1)))
ELSE
   crs12=2.*pi-acos((sin(lat2)-sin(lat1)*cos(dst12))/(sin(dst12)*cos(lat1)))
ENDIF
IF sin(lon1-lon2)<0
   crs21=acos((sin(lat1)-sin(lat2)*cos(dst12))/(sin(dst12)*cos(lat2)))
ELSE
   crs21=2.*pi-acos((sin(lat1)-sin(lat2)*cos(dst12))/(sin(dst12)*cos(lat2)))
ENDIF
   ang1=mod(crs13-crs12+pi,2.*pi)-pi
   ang2=mod(crs21-crs23+pi,2.*pi)-pi
IF (sin(ang1)*sin(ang2)<=sqrt(TOL))
   "no intersection exists"
ELSE
   ang1=abs(ang1)
   ang2=abs(ang2)
   ang3=acos(-cos(ang1)*cos(ang2)+sin(ang1)*sin(ang2)*cos(dst12))
   dst13=asin(sin(ang2)*sin(dst12)/sin(ang3))
   dst23=asin(sin(ang1)*sin(dst12)/sin(ang3))
   lat3=asin(sin(lat1)*cos(dst13)+cos(lat1)*sin(dst13)*cos(crs13))
   lon3=mod(lon1-asin(sin(crs13)*sin(dst13)/cos(lat3))+pi,2*pi)-pi
ENDIF
```

TOL is a small number of order machine precision. 10^-15 would be OK for standard double precision arithmetic.

Clairaut's formula:

This relates the latitude (lat) and true course (tc) along any great circle, namely: sin(tc)*cos(lat)=constant. That is, for any two points on the GC:

```
           sin(tc1)*cos(lat1)=sin(tc2)*cos(lat2)
```

Since at the highest latitude (latmx) reached the tc must be 90/270, we also have:

```
latmx=acos(abs(sin(tc)*cos(lat)))
```

where lat and tc are the latitude and true course at *any* point on the great circle.

Crossing parallels:

Any given great circle (excepting one over the poles) crosses each meridian once and only once. However, any given great circle has a maximum latitude reached at its apex. It crosses lower latitudes twice and higher latitudes never. Thus the algorithm for finding the longitudes at which a given great circle crosses a given parallel is a little more complex.

Suppose a great circle passes through (lat1,lon1) and (lat2,lon2). It crosses the parallel lat3 at longitudes lon3_1 and lon3_2 given by:

```
   l12 = lon1-lon2
   A = sin(lat1)*cos(lat2)*cos(lat3)*sin(l12)
   B = sin(lat1)*cos(lat2)*cos(lat3)*cos(l12) - cos(lat1)*sin(lat2)*cos(lat3)
   C = cos(lat1)*cos(lat2)*sin(lat3)*sin(l12)
   lon = atan2(B,A)                      ( atan2(y,x) convention)
  IF (C >sqrt(A^2 + B^2))
    "no crossing"
```

```
  ELSE
    dlon = acos(C/sqrt(A^2+B^2))
    lon3_1=mod(lon1+dlon+lon+pi, 2*pi)-pi
    lon3_2=mod(lon1-dlon+lon+pi, 2*pi)-pi
  ENDIF
```

Cross track error:

Suppose you are proceeding on a great circle route from A to B (course =crs_AB) and end up at D, perhaps off course. You can calculate the course from A to D (crs_AD) and the distance from A to D (dist_AD) using the formulae above. In terms of these the cross track error, XTD, (distance off course) is given by

```
           XTD =asin(sin(dist_AD)*sin(crs_AD-crs_AB))
```

(positive XTD means right of course, negative means left)

Implementation notes:

Notes on mathematical functions

Note: ^ denotes the exponentiation operator, sqrt is the square root function, acos the arc-cosine (or inverse cosine) function and asin is the arc-sine function. If asin or acos are unavailable they can be implemented using the atan2 function:

```
acos(x)=atan2(sqrt(1-x^2),x)
  acos returns a value in the range 0 <= acos <= pi
asin(x)=atan2(x,sqrt(1-x^2))}
  asin returns a value in the range -pi/2 <= asin <= pi/2
```

Note: Here atan2 has the conventional (C) ordering of arguments, namely atan2(y,x). This is not universal, Excel for instance uses atan2(x,y), but it has asin and acos anyway. Be warned. It returns a value in the range -pi < atan2 <= pi.

Further note: if your calculator/programming language is so impoverished that only atan is available then use:

```
atan2(y,x)=atan(y/x)       x>0
atan2(y,x)=atan(y/x)+pi    x<0, y>=0
atan2(y,x)=pi/2            x=0, y>0
atan2(y,x)=atan(y/x)-pi    x<0, y<0
atan2(y,x)=-pi/2           x=0, y<0
atan2(0,0) is undefined and should give an error.
```

Another potential implementation problem is that the arguments of asin and/or acos may, because of rounding error, exceed one in magnitude. With perfect arithmetic this can't happen. You may need to use "safe" versions of asin and acos on the lines of:

```
  asin_safe(x)=asin(max(-1,min(x,1)))
  acos_safe(x)=acos(max(-1,min(x,1)))
```

Note on the mod function. This appears to be implemented differently in different languages. Mod(y,x) is the remainder on dividing y by x and always lies in the range 0 <=mod <x. The following should be bulletproof:

```
  FUNCTION mod(y,x)
```

```
  IF y>=0
    mod=y- x*int(y/x)
  ELSE
    mod=y+ x*(int(-y/x)+1)
  ENDIF
```

Sign Convention

As stated in the introduction, North latitudes and West longitudes are treated as positive, and South latitudes and East longitudes negative. It's easier to go with the flow, but if you prefer another convention you can change the signs in the formulae.

Worked Examples:

```
  Suppose point 1 is LAX: (33deg 57min N, 118deg 24min W)
  Suppose point 2 is JFK: (40deg 38min N,  73deg 47min W)
```

In radians LAX is

(33+57/60)*pi/180=0.592539, (118+24/60)*pi/180=2.066470

and JFK is

(0.709186,1.287762)

The distance from LAX to JFK is

```
   d=acos(sin(lat1)*sin(lat2)+cos(lat1)*cos(lat2)*cos(lon1-lon2))
    =acos(sin(0.592539)*sin(0.709186)+
                     cos(0.592539)*cos(0.709186)*cos(0.778708))
    =acos(0.811790)
    =0.623585 radians
    =0.623585*180*60/pi=2144nm
```

The initial true course out of LAX is:

```
   sin(-0.778708)=-0.702<0 so

       tc1=acos((sin(lat2)-sin(lat1)*cos(d))/(sin(d)*cos(lat1)))
          =acos((sin(0.709186)-sin(0.592539)*cos(0.623585))/
                (sin(0.623585)*cos(0.592535))
          =acos(0.408455)
          =1.150035 radians
          =66 degrees
```

An enroute waypoint 100nm from LAX on the 66 degree radial (100nm along the GC to JFK) has lat and long given by:

```
  100nm = 100*pi/(180*60)=0.0290888radians
     lat=asin(sin(lat1)*cos(d)+cos(lat1)*sin(d)*cos(tc))
        =asin(sin(0.592539)*cos(0.0290888)
              +cos(0.592539)*sin(0.0290888)*cos(1.150035))
        =asin(0.568087)
        =0.604180radians
        =34degrees 37min N

     lon=lon1-asin(sin(tc)*sin(d)/cos(lat))
        =2.066470- asin(sin(1.150035)*sin(0.0290888)/cos(0.604180))
        =2.034206 radians
        =116 degrees 33min W
```

The great circle route from LAX to JFK crosses the 111degree W meridian at a latitude of:

```
    (111degrees=1.937315 radians)

  lat=atan((sin(lat1)*cos(lat2)*sin(lon-lon2)
     -sin(lat2)*cos(lat1)*sin(lon-lon1))/(cos(lat1)*cos(lat2)*sin(lon1-lon2)))
     =atan((sin(0.592539)*cos(0.709186)*sin(0.649553)
     -sin(0.709186)*cos(0.592539)*sin(-0.129154))/(cos(0.592539)*cos(0.709186)
                                                      *sin(0.778708)))
     =atan(0.737110)
     =0.635200radians
     =36 degrees 24min
```

Cross track error

Suppose enroute from JFK to LAX you find yourself at (D) N34:30 W116:30, which in radians is (0.6021386,2.033309) (See earlier for LAX, JFK coordinates and course)

From LAX to D the distance is:

```
 dist_AD=acos(sin(0.592539)*sin(0.6021386)+
       cos(0.592539)*cos(0.6021386)*cos(2.066470-2.033309))
        =0.02905 radians (99.8665 nm)
```

From LAX to D the course is:

```
 crs_AD=acos((sin(0.6021386)-sin(0.592539)*cos(0.02905))/
     (sin(0.02905)*cos(0.592539)))
         =1.22473 radians  (70.17 degrees)
```

At point D the cross track error is:

```
 xtk= asin(sin(0.02905)*sin(1.22473-1.15003))
    =0.00216747 radians
    = 0.00216747*180*60/pi =7.4512 nm right of course
```

Example of an intersection calc (briefly):

Let point 1 be REO (42.60N,117.866W)=(0.74351,2.05715)rad
Let point 2 be BKE (44.84N,117.806W)=(0.782606,2.056103)rad

The 51 degree (=0.890118rad) bearing from REO intersects with 137 degree (=2.391101rad) from BKE at (lat3,lon3):

Then:

```
dst12=0.039103
crs12=0.018996
crs21=3.161312
ang1=0.871122
ang2=0.770211
ang3=1.500667
dst13=0.02729
dst23=0.029986
lat3=0.760473      =43.5N
lon3=2.027876      =116.2W  at BOI!
```

Some general spherical triangle formulae.

A spherical triangle is one whose sides are all great circular arcs. Let the sides have lengths a,b and c radians, and the opposite angles be A, B and C radians.

```
                c
           A -------B
            \       |
             \      |
              \b    |a
               \    |
                \   |
                 \  |
                  \C|
                   \|

    (The angle at B is not necessarily a right angle)

      sin(a)  sin(b)   sin(c)
      ----- = ------ = ------
      sin(A)  sin(B)   sin(C)

  cos(a)=cos(b)*cos(c)+sin(b)*sin(c)*cos(A)
  cos(b)=cos(c)*cos(a)+sin(c)*sin(a)*cos(B)
  cos(c)=cos(a)*cos(b)+sin(a)*sin(b)*cos(C)

  cos(A)=-cos(B)*cos(C)+sin(B)*sin(C)*cos(a)
  cos(B)=-cos(C)*cos(A)+sin(C)*sin(A)*cos(b)
  cos(C)=-cos(A)*cos(B)+sin(A)*sin(B)*cos(c)

  Some useful consequences of these are:

  tan(A)=sin(B)*sin(a)/(sin(c)*cos(a)-cos(B)*cos(c)*sin(a))
  tan(B)=sin(C)*sin(b)/(sin(a)*cos(b)-cos(C)*cos(a)*sin(b))
  tan(C)=sin(A)*sin(c)/(sin(b)*cos(c)-cos(A)*cos(b)*sin(c))

  tan(a)=sin(b)*sin(A)/(sin(C)*cos(A)+cos(b)*cos(C)*sin(A))
  tan(b)=sin(c)*sin(B)/(sin(A)*cos(B)+cos(c)*cos(A)*sin(B))
  tan(c)=sin(a)*sin(C)/(sin(B)*cos(C)+cos(a)*cos(B)*sin(C))
```

Given *any* three of {a,b,c,A,B,C} the remaining sides and angles can be found from the above formulae.

```
  To solve a spherical triangle (requiring 0< a,b,c,A,B,C <pi
to get rid of pathological cases):
  Given {A,b,c}:  // Two sides, included angle
   a=acos(cos(b)*cos(c)+sin(b)*sin(c)*cos(A))
   B=acos((cos(b) - cos(c)*cos(a))/(sin(c)*sin(a)))
   C=acos((cos(c) - cos(a)*cos(b))/(sin(a)*sin(b)))

  Given {a,B,C}:  // Two angles, included side
   A=acos(-cos(B)*cos(C)+sin(B)*sin(C)*cos(a))
   b=atan2(sin(a)*sin(B)*sin(C),cos(B)+cos(C)*cos(A))
   c=atan2(sin(a)*sin(B)*sin(C),cos(C)+cos(A)*cos(B))

  Given {a,b,c}:  // Three sides
   A=acos((cos(a) - cos(b)*cos(c))/(sin(b)*sin(c)))
   B=acos((cos(b) - cos(c)*cos(a))/(sin(c)*sin(a)))
   C=acos((cos(c) - cos(a)*cos(b))/(sin(a)*sin(b)))

  Given {A,B,C}:  // Three angles (this has an infinity of solutions
for plane triangles and so is numerically inaccurate for small
spherical triangles)
   delta=(A+B+C-pi)/2
```

```
   a=2*asin(sqrt(sin(delta)*sin(A-delta)/(sin(B)*sin(C))))
   b=2*asin(sqrt(sin(delta)*sin(B-delta)/(sin(C)*sin(A))))
   c=2*asin(sqrt(sin(delta)*sin(C-delta)/(sin(A)*sin(B))))

  Given {A,a,b}:  // Two sides, non-included angle
   x=sin(A)*sin(b)/sin(a)
   if (x=1) {
     B=pi/2    // One spherical triangle exists
   } else if (x < 1) {
     B= asin(x) and pi-asin(x) // Two triangles exist
   } else{
     // No triangles exist
   }
   For each triangle
   c=mod(2*atan2(cos((A+B)/2)*sin((a+b)/2),cos((A-B)/2)*cos((a+b)/2)),2*pi)
   C=mod(2*atan2(cos((a-b)/2)*cos((A+B)/2),cos((a+b)/2)*sin((A+B)/2)),2*pi)

  Given {a,A,B}:  // Two angles, non-included side
   x=sin(a)*sin(B)/sin(A)
   if (x=1) {
     b=pi/2    // One spherical triangle exists
   } else if (x < 1) {
     b=asin(x) and pi-asin(x) // Two triangles exist
   } else{
     // No triangles exist
   }
   For each triangle
   c=mod(2*atan2(cos((A+B)/2)*sin((a+b)/2),cos((A-B)/2)*cos((a+b)/2)),2*pi)
   C=mod(2*atan2(cos((a-b)/2)*cos((A+B)/2),cos((a+b)/2)*sin((A+B)/2)),2*pi)
```

Note that for a spherical triangle A+B+C is not pi (180 degrees) but greater.
The difference is called the spherical excess E, defined as E=A+B+C-pi.
In terms of which the surface area enclosed by a spherical triangle is given by

```
Area = E*R^2
```

In terms of the sides:

```
    E = 4*sqrt(atan(tan(s/2)*tan((s-a)/2)*tan((s-b)/2)*tan((s-c)/2)))
```

where

```
    s = (a+b+c)/2
```

analogous to Heron's formula for a plane triangle.

Some other formulae that may occasionally be useful are:

```
   sin(A/2) = sqrt((sin(s-b)*sin(s-c))/(sin(b)*sin(c)))
   cos(A/2) = sqrt((sin(s)*sin(s-a))/(sin(b)*sin(c)))
   tan(A/2) = sin((b-c)/2)/(sin((b+c)/2)*tan((B-C)/2))
            = cos((b-c)/2)/(cos((b+c)/2)*tan((B+C)/2))
   tan(a/2) = cos((B+C)/2)*tan((b+c)/2)/cos((B-C)/2)
            = sin((B+C)/2)*tan((b-c)/2)/sin((B-C)/2)
   tan((A-B)/2)=cot(C/2)*sin((a-b)/2)/sin((a+b)/2)
   tan((A+B)/2)=cot(C/2)*cos((a-b)/2)/cos((a+b)/2)
   sin(a)*cos(B)=cos(b)*sin(c)-sin(b)*cos(c)*cos(A)
   cos(a)*cos(C)=sin(a)*cot(b)-sin(C)*cot(B)
```

In these formulae, A, B and C can be interchanged, provided a, b and c change with them.

```
  ie a->b, b->c, c->a, A->B, B->C, C->A.
In addition, the formulae hold if pi-a is written for A,
pi-b for B and pi-c for C, etc.
  ie A->pi-a, B->pi-b, C->pi-c, a->pi-A, b->pi-B, c->pi-C
----------------------------------------------------------------------------
```

Rhumb Line Navigation

Rhumb lines or loxodromes are tracks of constant true course. With the exception of meridians and the equator, they are not the same as great circles. They are not very useful approaching either pole, where they become tightly wound spirals. The formulae below fail if any point actually is a pole.

When two points (lat1,lon1), (lat2,lon2) are connected by a rhumb line with true course tc :

```
 lon2-lon1=-tan(tc)*(log((1+sin(lat2))/cos(lat2))-
                    log((1+sin(lat1))/cos(lat1)))
          =-tan(tc)*(log((1+tan(lat2/2))/(1-tan(lat2/2)))-
                    log((1+tan(lat1/2))/(1-tan(lat1/2))))
          =-tan(tc)*(log(tan(lat2/2+pi/4)/tan(lat1/2+pi/4)))
```

(logs are "natural" logarithms to the base e.)

The true course between the points is given by:

```
     tc= mod(atan2(lon1-lon2,log(tan(lat2/2+pi/4)/tan(lat1/2+pi/4))),2*pi)
```

The dist, d between the points is given by:

```
         if (abs(lat2-lat1) < sqrt(TOL)){
             q=cos(lat1)
         } else {
             q= (lat2-lat1)/log(tan(lat2/2+pi/4)/tan(lat1/2+pi/4))
         }
         d=sqrt((lat2-lat1)^2+ q^2*(lon2-lon1)^2)
```

This formula fails if the rhumb line in question crosses the 180 E/W meridian. Allowing this as a possibility, the true course tc, and distance d, for the shortest rhumb line connecting two points is given by:

```
  dlon_W=mod(lon2-lon1,2*pi)
  dlon_E=mod(lon1-lon2,2*pi)
  dphi=log(tan(lat2/2+pi/4)/tan(lat1/2+pi/4))
  if (abs(lat2-lat1) < sqrt(TOL)){
     q=cos(lat1)
  } else {
     q= (lat2-lat1)/dphi
  }
  if (dlon_W < dlon_E){// Westerly rhumb line is the shortest
      tc=mod(atan2(-dlon_W,dphi),2*pi)
      d= sqrt(q^2*dlon_W^2 + (lat2-lat1)^2)
  } else{
      tc=mod(atan2(dlon_E,dphi),2*pi)
      d= sqrt(q^2*dlon_E^2 + (lat2-lat1)^2)
      }
```

To find the lat/lon of a point on true course tc, distance d from (lat1,lon1) along a rhumbline (initial point cannot be a pole!):

```
  lat= lat1+d*cos(tc)
  dphi=log(tan(lat/2+pi/4)/tan(lat1/2+pi/4))
  IF (abs(lat-lat1) < sqrt(TOL)){
```

```
    q=cos(lat1)
 } ELSE {
    q= (lat-lat1)/dphi
 }
 dlon=-d*sin(tc)/q
 lon=mod(lon1+dlon+pi,2*pi)-pi
```

TOL is a small number of order machine precision- say 1e-15. The tests avoid 0/0 indeterminacies on E-W courses.

Example:

```
  Suppose point 1 is LAX: (33deg 57min N, 118deg 24min W)
  Suppose point 2 is JFK: (40deg 38min N,  73deg 47min W)
Rhumb line course from LAX to JFK:
LAX (0.592539,2.066470) radians and JFK is (0.709185,1.287762) radians

dlon_W=mod(1.287762-2.066470,2*pi)=5.504478
dlon_E=mod(2.066470-1.287762,2*pi)=0.778708

dphi=log(tan(0.709185/2+pi/4)/tan(0.592539/2+pi/4))
    =0.146801
q= (0.709185-0.592539)/0.146801 =0.794586
  dlon_E < dlon_W: East is shorter!
tc=mod(atan2(0.778708,0.146801),2*pi)= 1.384464 radians = 79.32 degrees
d=sqrt(0.794586^2*0.778708^2 + (0.709185-0.592539)^2)
= 0.629650 radians = 2164.6 nm
```

Compare this with the great circle course of 66 degrees and distance of 2144 nm.

Conversely, if we proceed 2164.6nm (0.629650 radians) on a rhumbline course of 79.3 degrees (1.384464 radians) starting at LAX, our final point will be given by:

```
lat=0.592539 + 0.629650 * cos(1.384464)
   = 0.709185
dphi=log(tan(0.709185/2+pi/4)/tan(0.592539/2+pi/4))
    =0.146801
q= (0.709185-0.592539)/0.146801 =0.794586
dlon=-0.629650*sin(1.384464)/0.794586=-0.778708
lon=mod(2.066470-0.778708+pi,2*pi)-pi
   =1.287762
```

which is the lat/lon of JFK- as required.

Wind Triangles

In all formulae, all angles are in radians. Convert back and forth as in the Great Circle section. [This is unnecessary on calculators which have a "degree mode" for trig functions. Most programming languages provide only "radian mode".]

```
      angle_radians=(pi/180)*angle_degrees
      angle_degrees=(180/pi)*angle_radians
```

A further conversion is required if using degrees/minutes/seconds:

```
 angle_degrees=degrees+(minutes/60.)+(seconds/3600.)

 degrees=int(angle_degrees)
 minutes=int(60*(angle_degrees-degrees))
```

```
 seconds=60*(60*(angle_degrees-degrees)-minutes))
```

[You may have a built-in HH <-> HH:MM:SS conversion to do this efficiently]

Let CRS=course, HD=heading, WD=wind direction (from), TAS=True airpeed, GS=groundspeed, WS=windspeed.

Units of the speeds do not matter as long as they are all the same.

(1) Unknown Wind:

```
 WS=sqrt( (TAS-GS)^2+ 4*TAS*GS*(sin((HD-CRS)/2))^2 )
 WD=CRS + atan2(TAS*sin(HD-CRS), TAS*cos(HD-CRS)-GS)  (**)
 IF (WD<0) THEN WD=WD+2*pi
 IF (WD>2*pi) THEN WD=WD-2*pi
```

((**) assumes atan2(y,x), reverse arguments if your implementation has atan2(x,y))

(2) Find HD, GS

```
 SWC=(WS/TAS)*sin(WD-CRS)
 IF (abs(SWC)>1)
      "course cannot be flown-- wind too strong"
 ELSE
      HD=CRS+asin(SWC)
      if (HD<0) HD=HD+2*pi
      if (HD>2*pi) HD=HD-2*pi
      GS=TAS*sqrt(1-SWC^2)-WS*cos(WD-CRS)
 ENDIF
```

Note:
The purpose of the "if (HD<0) HD=HD+2*pi; if (HD>2*pi) HD=HD-2*pi" is to ensure the final heading ends up in the range (0, 2*pi). Another way to do this, with the MOD function available is:

```
   HD=MOD(HD,2*pi)
```

(3) Find CRS, GS

```
  GS=sqrt(WS^2 + TAS^2 - 2*WS*TAS*cos(HD-WD))
  WCA=atan2(WS*sin(HD-WD),TAS-WS*cos(HD-WD))  (*)
  CRS=MOD(HD+WCA,2*pi)
```

(*) WCA=asin((WS/GS)*sin(HD-WD)) works if the wind correction angle is less than 90 degrees, which will always be the case if WS < TAS. The listed formula works in the general case

----

Approximate variation formulae.

I did a least squares polynomial fit to the NFDC airport database.

x=latitude (N degrees) y=longitude (W degrees) var= variation (degrees)

```
   var=  -65.6811 + 0.99*x + 0.0128899*x^2 - 0.0000905928*x^3 + 2.87622*y -
        0.0116268*x*y - 0.00000603925*x^2*y - 0.0389806*y^2 -
        0.0000403488*x*y^2 + 0.000168556*y^3
```

Continental US only, 3771 points, RMS error 1 degree All within 2 degrees except for the following airports: MO49 MO86 MO50 3K6 02K and KOOA

(24 < x < 50, 66 < y < 125)

Alaska Fit, better than 1 degree, all points:

```
  var=  618.854 + 2.76049*x - 0.556206*x^2 + 0.00251582*x^3 - 12.7974*y +
        0.408161*x*y + 0.000434097*x^2*y - 0.00602173*y^2 -
        0.00144712*x*y^2 + 0.000222521*y^3
```

55 points (x > 54, 130 < y < 172)

For Western Europe, fitting to the 1997 IGRF reference field:

```
 var =10.4768771667158 -0.507385322418858*lon +0.00753170031703826*lon^2-
    1.40596203924748e-05*lon^3 -0.535560699962353*lat +
    0.0154348808069955*lat*lon -8.07756425110592e-05*lat*lon^2 +
    0.00976887198864442*lat^2 -0.000259163929798334*lat^2*lon-
    3.69056939266123e-05*lat^3;
```

Here *East* lon is positive! In the range -10 < lon < 28, 36 < lat < 68 RMS error = 0.04 degrees, max error 0.20 degrees.

I've written software that computes magnetic variation anywhere on (or above) the earth's surface, using either the WMM or IGRF reference models. There are Mac , DOS and Linux executables available.

Standard Atmosphere and Altimetry

The following contains some formulae concerning altimetry and the standard atmosphere (1976 International Standard Atmosphere).

At sea-level on a standard day:

```
    the temperature, T_0 = 59F = 15C = 288.15K (C=Celsius K=Kelvin,
                                     T(Kelvin)=T(Celsius)+273.15)

    the pressure, P_0 = 29.92126 "Hg = 1013.250 mB = 2116.2166 lbs/ft^2
                      = 760.0 mmHg   = 101325.0 Pa = 14.69595 psi = 1.0 atm

    the air density, rho_0 = 1.2250 kg/m^3 = 0.002376892 slugs/ft^3
```

The standard lapse rate is T_r= 0.0065C/m = .0019812C/ft below the tropopause h_Tr= 11.0km= 36089.24ft

Above the tropopause, standard temperature is T_Tr= -56.5C= 216.65K (up to an altitude of 20km) Standard temperature at altitude h is thus given by:

```
   T_s= T_0- T_r*h  (h < h_Tr)
      = T_Tr        (h > h_Tr)
      = 15-.0019812*h(ft) C (h < 36089.24ft)
```

Variation of pressure with altitude:

```
  p= P_0*(1-6.8755856*10^-6 h)^5.2558797    h<36,089.24ft
  p_Tr= 0.2233609*P_0
  p=p_Tr*exp(-4.806346*10^-5(h-36089.24)) h>36,089.24ft
```

Variation of density with altitude:

```
  rho=rho_0*(1.- 6.8755856*10^-6 h)^4.2558797 h<36,089.24ft
  rho_Tr=0.2970756*rho_0
  rho=rho_Tr*exp(-4.806346*10^-5(h-36089.24)) h>36,089.24ft
```

Relationship of pressure and indicated altitude:

```
  alt_set in inches, heights in feet
  P_alt_corr= 145442.2*(1- (alt_set/29.92126)^0.190261)  or
  P_alt_corr= (29.92-alt_set)*1000 (simple approximation)
  P_alt= Ind_Alt + P_alt_corr
```

Relationship of pressure and density altitude:

```
  D_Alt=P_alt+(T_s/T_r)*(1.-(T_s/T)^0.2349690)
     (Standard temp T_s and actual temp T in Kelvin)
```

An approximate, but fairly accurate formula is:

```
  D_Alt=P_Alt+118.6*(T-T_s)
  where T and T_s may (both) be either Celsius or Kelvin
```

Density altitude example:

Let pressure altitude (P_alt) be 8000 ft, temperature 18C.

Standard temp (T_s) is given by

```
   T_s=15-.0019812*8000=-0.85C = (273.15-0.85)K=272.30K
```

Actual temperature (T) is

```
 18C=(273.15+18)K=291.15K

  Density altitude (D_Alt) = 8000 +(272.30/.0019812)*(1-
(272.30/291.15)^0.2349690)
                           = 8000 + 2145 = 10145ft
```

or approximately:

```
  Density Altitude=8000 +118.6*(18+0.85)=10236ft
```

Relationship of true and calibrated (indicated) altitude:

```
 TA= CA + (CA-FE)*(ISADEV)/(273+OAT)
```

where

```
 TA= True Altitude above sea-level
 FE= Field Elevation of station providing the altimeter setting
 CA= Calibrated altitude= Altitude indicated by altimeter when set to the
      altimeter setting, corrected for calibration error.

 ISADEV= Average deviation from standard temperature from standard in the air
       column between the station and the aircraft (in C)
```

OAT= Outside air temperature (at altitude)

The above is more precise than provided by the E6B or similar.

Mach numbers, true vs calibrated airspeeds etc.

```
 Mach Number (M) = TAS/CS
 CS = sound speed= 38.967854*sqrt(T+273.15)  where T is the OAT in celsius.
 TAS is true airspeed in knots.
```

Because of compressibility, the measured IAT (indicated air temperature) is higher than the actual true OAT. Approximately:

```
  IAT=OAT+K*TAS^2/7592
```

The recovery factor K, depends on installation, and is usually in the range 0.95 to 1.0, but can be as low as 0.7. Temperatures are Celsius, TAS in knots.

Also:

```
    OAT = (IAT + 273.15) / (1 + 0.2*K*M^2) - 273.15
```

The airspeed indicator measures the differential pressure, DP, between the pitot tube and the static port, the resulting indicated airspeed (IAS), when corrected for calibration and installation error is called "calibrated airspeed" (CAS).

For low-speed (M<0.3) airplanes the true airspeed can be obtained from CAS and the density altitude, DA.

```
  TAS = CAS*(rho_0/rho)^0.5=CAS/(1-6.8755856*10^-6 * DA)^2.127940
(DA<36,089.24ft)
```

Roughly, TAS increases by 1.5% per 1000ft.

When compressibility is taken into account, the calculation of the TAS is more elaborate:

```
  DP=P_0*((1+0.2*(IAS/CS_0)^2)^3.5 -1)
  M=(5*( (DP/P+1)^(2/7) -1) )^0.5
  TAS= M*CS
```

P_0 is is (standard) sea-level pressure, CS_0 is the speed of sound at sea-level, CS is the speed of sound at altitude, and P is the pressure at altitude.

These are given by earlier formulae:

```
    P_0= 29.92126 "Hg = 1013.25 mB = 2116.2166 lbs/ft^2
    P= P_0*(1-6.8755856*10^-6*PA)^5.2558797, pressure altitude, PA<36,089.24ft
    CS= 38.967854*sqrt(T+273.15)  where T is the (static/true) OAT in Celsius.
    CS_0=38.967854*sqrt(15+273.15)=661.4786 knots
```

```
  [Example: CAS=250 knots, PA=10000ft, IAT=2C, recovery factor=0.8
  DP=29.92126*((1+0.2*(250/661.4786)^2)^3.5 -1)= 3.1001 "
  P=29.92126*(1-6.8755856*10^-6 *10000)^5.2558797= 20.577 "
  M= (5*( (3.1001/20.577 +1)^(2/7) -1) )^0.5= 0.4523 Mach
  OAT=(2+273.15)/(1 + 0.2*0.8*0.4523^2) - 273.15= -6.72C
  CS= 38.967854*sqrt(-6.7+273.15)=636.08 knots
  TAS=636.08*0.4523=287.7 knots]
```

Some notes on the origins of some of the "magic" number constants in the preceeding section:

6.8755856*10^-6 = T'/T_0, where T' is the standard temperature lapse rate and T_0 is the standard sea-level temperature.

5.2558797 = Mg/RT_0, where M is the (average) molecular weight of air, g is the acceleration of gravity and R is the gas constant.

0.2233609 = ratio of the pressure at the tropopause to sea-level pressure.

4.806346*10^-5 = Mg/RT_tr, where T_tr is the temperature at the tropopause.

4.2558797 = Mg/RT_0 -1

0.2970756 = ratio of the density at the tropopause to the density at SL (rho_0)

145442 = T_0/T'

38.967854 = sqrt(gamma R T_0/M)

Relative humidity, dewpoint, frostpoint etc.

The relative humidity, f (as a fraction) is related to the temperature, T and dewpoint Td by:

```
       f= exp(17.27(Td/(Td+237.3)-T/(T+237.3)))
```

and to the frostpoint temperature Tf by:

```
       f= exp(21.87(Tf/(Tf+265.5)-T/(T+265.5)))
```

Temperatures are in Celsius. Multiply f by 100 if you want a percentage. The above are based on an empirical fit to the saturation vapor pressure of water due to O. Tetens in Zeitschrift fur Geophysik, Vol VI (1930), quoted in "Principles of Meteorological Analysis" by W. J. Saucier (Dover NY 1983).

This fit is:

```
 e_s=6.11 * exp(bT/(T+a))   for the saturation vapor pressure e_s in mbar
   over water a=237.3, b=17.27
   over ice a=265.5, b=21.87
  An alternative slightly more accurate fit (over water) is:
 e_s = 6.10779 + T * (4.43652e-1 + T * (1.42894e-2 + T * (2.65064e-4 + T *
        (3.03124e-6 + T * (2.03408e-8 + (6.13682e-11 * T))))))
 (from Lowe, JAM (1977), 103)
```

Tables of Relative Humidity and Dewpoint vs Temperature and Wet Bulb Temperature can be found in "Introduction to Meteorology" by Franklyn Cole (Wiley NY 1975).

Inverting this to find dewpoint in terms of temp and RH:

```
 Dewpoint   Td=237.3/(1/(ln(f)/17.27+T/(T+237.3))-1)
 Frostpoint Tf=265.5/(1/(ln(f)/21.87+T/(T+265.5))-1)
```

Given the wet bulb temperature Tw (C), the dry bulb temperature T (C), and the pressure, p in mbar one gets the (approximate) relative humidity and dewpoint by the following:

```
 ed= 6.11*exp(17.27*T/(T+237.3))       /* SVP at dry-bulb temp
 ew= 6.11*exp(17.27*Tw/(Tw+237.3))       /* SVP at wet-bulb temp
 wd=0.62197*ed/(p-ed)                 /* saturation mixing ratio at T
 ww=0.62197*ew/(p-ew)                 /* saturation mixing ratio at Tw
 w=(2500.0*ww-1.0046*(T-Tw))/(2500.0+1.81*(T-Tw)) /* mixing ratio
 f= w/wd                              /* relative humidity as a fraction
 e= p*w/(0.62197+w)                   /* vapor pressure (mb)
 Td=(237.3*log10(e)-186.527)/(8.286-log10(e)) /* the dewpoint (C)
```

This uses the Tetens fit for the saturated vapor pressure and treat water vapor as an ideal gas, both of which are pretty good approximations. If you want better refer to the Smithsonian Meteorological Tables (Smithsonian Institute 1963)

A related formula gives the increase in effective density altitude due to humidity. It only addresses the reduction of air density, and not the effect on engine power output:

```
Increase(ft)=0.267*RH*(T+273)*exp(17.3*T/(T+237))*(1-0.00000688*H)^(-5.26)
```

RH (f above) is the relative humidity expressed as a fraction, T is the temperature in Celsius and H is the pressure altitude in feet.

Examples are:

```
 SL/30C/100% -> 565' increase in DA
 10000/5C/80% -> 124' increase in DA
 5000/40C/80% -> 977' increase in DA.
```

In terms of the dewpoint, Td the formula is:

```
  Increase(ft)=0.267*(T+273)*exp(17.3*Td/(Td+237))*(1-0.00000688*H)^(-5.26)
```

which clearly agrees with the above when T=Td and RH=1.

Bellamy's formula.

Bellamy's formula for the wind drift and (single) wind correction angle is as follows:

```
   Drift (nm) = 21500*(p2-p1)/(sin(latitude)*TAS)  (p2-p1 in inches)
              = 635  *(p2-p1)/(sin(latitude)*TAS)  (p2-p1 in mB)

   Wind Correction Angle= 1230000*(p2-p1)/(sin(latitude)*TAS*Dist)  (inches)
                        = 36300*  (p2-p1)/(sin(latitude)*TAS*Dist)   (mB)
```

p2-p1 is the difference between the destination and departure pressures. latitude is the average latitude on the route. TAS is the true airspeed in knots. Dist is the distance in nm.

If the destination pressure is higher, the drift is to the left, and the required WCA is to the right (and vice-versa).

Example:

SFO -> LAX 300nm at 100 knots, latitude 36 degrees. Suppose the LAX altimeter setting is 0.2" higher (better the actual pressure difference at cruise altitude if you can get it).

```
    Drift = 21500*0.2/(sin(36)*100)= 73nm left
    WCA=1230000*0.2/(sin(36)*100*300)= 14 degrees right
```

A discussion of this is in Barry Schiff's "Proficient Pilot I".

Unit conversions, etc.

```
 1 knot = 1.150779 mph
 1 mph  = 0.868976 knot
 1 knot = 1.852000 km/hr*
 1 km/hr= 0.539968 knot
 1 mph  = 1.609344 km/hr*
 1 km/hr= 0.621371 mph
```

* = exact conversion factor

Ellipsoidal parameters:

Name	Major axis, a (km)	Flattening (f)
WGS84	6378.13700	1/298.257223563
GRS80/NAD83	6378.13700	1/298.257222101
WGS66	6378.145	1/298.25
GRS67/IAU68	6378.16000	1/298.2472
WGS72	6378.135	1/298.26
Krasovsky	6378.245	1/298.3
Clarke66/NAD27	6378.2064	1/294.9786982138

Reference: Coordinate Systems and Map Projections, D. H. Maling (Pergamon 1992) (except Clarke66 !)

To convert between geocentric (radius r, geocentric latitude u) and geodetic coordinates (geodetic latitude v, height above the ellipsoid h):

```
  tan(u) = tan(v)*(h*sqrt((a*cos(v))^2+(b*sin(v))^2) +b^2)/
                  (h*sqrt((a*cos(v))^2+(b*sin(v))^2) +a^2)

  r^2 = h^2 + 2*h*sqrt((a*cos(v))^2+(b*sin(v))^2)+
               (a^4-(a^4-b^4)*(sin(v))^2)/(a^2-(a^2-b^2)*(sin(v))^2)
```

a and b are the semi-major axes of the ellipsoid, and b=a*(1-f), where f is the flattening. Note that geocentric and geodetic longitudes are equal.

Turns and pivotal altitude

In a steady turn, in no wind, with bank angle, b at an airspeed v

```
  tan(b)= v^2/(R g)
  v= w R
```

where g is the acceleration due to gravity, R is the radius of turn and w is the rate of turn.

Pivotal altitude h_p is given by

```
   h = v^2/g
```

With R in feet, v in knots, b in degrees and w in degrees/sec (inconsistent units!), numerical constants are introduced:

R =v^2/(11.23*tan(0.01745*b))

(Example) At 100 knots, with a 45 degree bank, the radius of turn is 100^2/(11.23*tan(0.01745*45))= 891 feet.

The rate of turn w is given by:

w = 96.7*v/R

(Example) = 96.7*100/891= 10.9 degs/sec

The bank angle b_s for a standard rate turn is given by:

b_s = 57.3*atan(v/362.1)

(Example) for 100 knots, b_s = 57.3*atan(100/362.1) = 15.4 degrees
A useful rule-of-thumb, accurate to ~1 degree for speeds up to 250 knots, is b_s= v/7 (v in knots).

The pivotal altitude is given by:

h_p = v^2/11.23

(Example) At 100 knots groundspeed the pivotal altitude is 100^2/11.23 = 890 feet.

Revision History

Version 1.24 2/10/99
Corrected some last digit rounding errors in the rhumb line examples
1.23
Additions to spherical triangle section
1.22
Course between points formula fails if the initial point was exactly a pole. This has to be special-cased.
1.21
Added Mach -> IAS formulae

1.20

Third numerical example of the effect of humidity on density altitude corrected.
Added standard rate turn bank angle rule-of-thumb.

1.19

Another bug in the intersection section... The test for input data where "no intersection exists", or more precisely, when it's ambiguous which of the two great circle intersections is desired, was misplaced. With valid data, no problem...
The Clarke66/NAD27 inverse flattening was incorrect in my reference book. Corrected. Thanks to Larry Lewis.

1.18

Corrected equation for dst12 in intersection calculation. It should have been the same as for the distance between points given earlier. A factor of 2 was dropped. The numerical example used the correct formula.

1.17

(1/26/98) Converted from 1962 to 1976 US Standard Atmosphere (=ICAO standard atmosphere). Made unit conversions more accurate. (by Doug Haluza)

1.16

(10/26/97) Corrected conversion to hh:mm:ss
seconds=60*(60*(angle_degrees-degrees)-minutes))

1.15

(9/11/97) Added European variation fit

1.14

(9/2/97) Added warnings about arguments of asin and acos being out of range from rounding error.

1.13

(8/31/97) The rhumb line section was rewritten. Erroneously corrected one formula, then changed it back! Added a numerical example for the calculation of the endpoint of a rhumb line. Added some more spherical triangle formulae.

1.12

Somehow I dropped a line in the the 1.08 atan2 fix. Sigh! Added turn radius, pivotal altitude formulae.

1.11

Made "Lat/lon given radial and distance" handle the pole endpoint case more elegantly.

1.10

Add "find CRS, GS" to wind triangle section

1.09

Added geodetic/geocentric coordinate conversion

1.08

Added an alternative method for calculation of course between two points, not requiring pre-computation of the distance between them.

Changed the definition of atan2 to the ANSI standard one where it is defined to have a range of -pi < atan2 <= pi, rather than 0 <= atan2 < 2pi. This was a bug only if had you used the previous version to define asin in terms of atan via atan2. No one reported it though...

Corrected some damaged formulae in the intersection section of the html version.

1.07 (4/1/97)

Add additional spherical triangle formulae. Correct the condition (dlon<pi/2) for the validity of the short range formula in the "lat/lon given radial and distance" section.

1.06 (3/3/97)

Correct typo in html version of HDG/GS formula. (minus sign) Definitions of a and b swapped in Tejen's fit to saturation vapor pressure.

1.05 (12/17/96)

Correct test for pole in formula for computing lat/long of a point a given radial and distance: lat=0 => cos(lat)=0

1.04 (11/11/95)

Add formula for computing lat/long of a point a given radial and distance valid when the distance can exceed one quarter of the earth's circumference.

Note that atan2(0,0) should return an error.

Add rhumb line formulae and example.

Change intersection calculation to only provide result when intersection of radials exists.

Comments, corrections, suggestions to:
Ed Williams
72347.1516@CompuServe.COM

Home page http://www.best.com/~williams