```
:::::::::::::::
cordic_tb.m
:::::::::::::::
function cordic_tb();
%*********************************************************************************80
%
%  CORDIC test
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    14 June 2007
%
%  Author:
%
%    Modifications by John Burkardt
%
%    Further modified by Young W. Lim
%
%  Parameters:
%
%
%*********************************************************************************80

  pi = 3.141592653589793;
  K = 1.646760258121;
  nIter = 10;


% ------------------------------------------------------------------------
%  printf ("\nGrinding on [K, 0, 0]\n");
%  Circular (X0C, 0L, 0L);
% ------------------------------------------------------------------------
xi = 1 / K;
yi = 0.0;
zi = 0.0;
printf("-----------------------------------------------\n");
printf("xi=%f yi=%f zi=%f\n", xi, yi, zi);

[xo, yo, zo] = cordic(xi, yi, zi, nIter);

printf("xo=%f yo=%f zo=%f\n", xo, yo, zo);


% ------------------------------------------------------------------------
%  printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
%  Circular (X0C, 0L, HalfPi / 3L);
% ------------------------------------------------------------------------
xi = 1 / K;
yi = 0.0;
zi = pi / 6.0;
printf("-----------------------------------------------\n");
printf("xi=%f yi=%f zi=%f\n", xi, yi, zi);

[xo, yo, zo] = cordic(xi, yi, zi, nIter);

printf("xo=%f yo=%f zo=%f\n", xo, yo, zo);


% ------------------------------------------------------------------------
%  printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
%  Circular (X0C, 0L, HalfPi / 2L);
% ------------------------------------------------------------------------
xi = 1 / K;
yi = 0.0;
zi = pi / 4.0;
printf("-----------------------------------------------\n");
```

```
printf("xi=%f yi=%f zi=%f\n", xi, yi, zi);

[xo, yo, zo] = cordic(xi, yi, zi, nIter);

printf("xo=%f yo=%f zo=%f\n", xo, yo, zo);


% ------------------------------------------------------------------------
%  printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
%  Circular (X0C, 0L, 2L * (HalfPi / 3L));
% ------------------------------------------------------------------------
xi = 1 / K;
yi = 0.0;
zi = pi / 3.0;
printf("-----------------------------------------------\n");
printf("xi=%f yi=%f zi=%f\n", xi, yi, zi);

[xo, yo, zo] = cordic(xi, yi, zi, nIter);

printf("xo=%f yo=%f zo=%f\n", xo, yo, zo);



::::::::::::::
cordic.m
::::::::::::::
function [xo, yo, zo]  = cordic ( x, y, z, n )

%*************************************************************************80
%
%  CORDIC returns the sine and cosine using the CORDIC method.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    14 June 2007
%
%  Author:
%
%    Based on MATLAB code in a Wikipedia article.
%
%    Modifications by John Burkardt
%
%    Further modified by Young W. Lim
%
%  Parameters:
%
%    -- Input, real BETA, the angle (in radians).
%    --
%    -- Input, integer N, the number of iterations to take.
%    -- A value of 10 is low.  Good accuracy is achieved with 20 or more
%    -- iterations.
%    Input: xi, yi, zi, nIter
%
%    -- Output, real V(2), the cosine and sine of the angle.
%    Output: xo, yo, zo
%
%  Local Parameters:
%
%    Local, real ANGLES(60) = arctan ( (1/2)^(0:59) );
%
%    Local, real KPROD(33), KPROD(j) = product ( 0 <= i <= j ) K(i),
%    K(i) = 1 / sqrt ( 1 + (1/2)^(2i) ).
%
%*************************************************************************80
  angles = [  ...
    7.853981633974830962E-01, ...
```

```
  4.636476090008061162 1E-01, ...
  2.449786631268641541 7E-01, ...
  1.243549945467614350 3E-01, ...
  6.241880999595734847 4E-02, ...
  3.123983343026827625 4E-02, ...
  1.562372862047683080 3E-02, ...
  7.812341060101111296 5E-03, ...
  3.906230131966971827 6E-03, ...
  1.953122516478818685 1E-03, ...
  9.765621895593194304 0E-04, ...
  4.882812111948982754 7E-04, ...
  2.441406201493617640 2E-04, ...
  1.220703118936702042 4E-04, ...
  6.103515617420877502 2E-05, ...
  3.051757811552609686 2E-05, ...
  1.525878906131576210 7E-05, ...
  7.629394531101970263 4E-06, ...
  3.814697265606496282 9E-06, ...
  1.907348632810187035 4E-06, ...
  9.536743164059608794 2E-07, ...
  4.768371582030888599 3E-07, ...
  2.384185791015579824 9E-07, ...
  1.192092895507806853 1E-07, ...
  5.960464477539055441 4E-08, ...
  2.980232238769530367 7E-08, ...
  1.490116119384765514 7E-08, ...
  7.450580596923827987 1E-09, ...
  3.725290298461914045 3E-09, ...
  1.862645149230957029 1E-09, ...
  9.313225746154785153 6E-10, ...
  4.656612873077392577 8E-10, ...
  2.328306436538696289 0E-10, ...
  1.164153218269348144 5E-10, ...
  5.820766091346740722 6E-11, ...
  2.910383045673370361 3E-11, ...
  1.455191522836685180 7E-11, ...
  7.275957614183425903 3E-12, ...
  3.637978807091712951 7E-12, ...
  1.818989403545856475 8E-12, ...
  9.094947017729282379 2E-13, ...
  4.547473508864641189 6E-13, ...
  2.273736754432320594 8E-13, ...
  1.136868377216160297 4E-13, ...
  5.684341886080801487 0E-14, ...
  2.842170943040400743 5E-14, ...
  1.421085471520200371 7E-14, ...
  7.105427357601001858 7E-15, ...
  3.552713678800500929 4E-15, ...
  1.776356839400250464 7E-15, ...
  8.881784197001252323 4E-16, ...
  4.440892098500626161 7E-16, ...
  2.220446049250313080 8E-16, ...
  1.110223024625156540 4E-16, ...
  5.551115123125782702 1E-17, ...
  2.775557561562891351 1E-17, ...
  1.387778780781445675 5E-17, ...
  6.938893903907228377 6E-18, ...
  3.469446951953614188 8E-18, ...
  1.734723475976807094 4E-18 ];

kprod = [ ...
  0.70710678118654752440, ...
  0.63245553203367586640, ...
  0.61357199107789634961, ...
  0.60883391251775242102, ...
  0.60764825625616820093, ...
  0.60735177014129595905, ...
  0.60727764409352599905, ...
  0.60725911229889273006, ...
  0.60725447933256232972, ...
```

```
    0.60725332108987516334, ...
    0.60725303152913433540, ...
    0.60725295913894481363, ...
    0.60725294104139716351, ...
    0.60725293651701023413, ...
    0.60725293538591350073, ...
    0.60725293510313931731, ...
    0.60725293503244577146, ...
    0.60725293501477238499, ...
    0.60725293501035403837, ...
    0.60725293500924945172, ...
    0.60725293500897330506, ...
    0.60725293500890426839, ...
    0.60725293500888700922, ...
    0.60725293500888269443, ...
    0.60725293500888161574, ...
    0.60725293500888134606, ...
    0.60725293500888127864, ...
    0.60725293500888126179, ...
    0.60725293500888125757, ...
    0.60725293500888125652, ...
    0.60725293500888125626, ...
    0.60725293500888125619, ...
    0.60725293500888125617 ];
%
%  Shift angle to interval [-pi,pi].
%
%  theta = angle_shift ( beta, -pi );
%
%  Shift angle to interval [-pi/2,pi/2] and account for signs.
%
%  if ( theta < - 0.5 * pi )
%    theta = theta + pi;
%    sign_factor = -1.0;
%  elseif ( 0.5 * pi < theta )
%    theta = theta - pi;
%    sign_factor = -1.0;
%  else
%    sign_factor = +1.0;
%  end

  theta = z;
  v = [ x; y ];
  poweroftwo = 1.0;
  angle = angles(1);

  for j = 1 : n

    if ( theta < 0.0 )
      sigma = -1.0;
    else
      sigma = 1.0;
    end

    factor = sigma * poweroftwo;

    R = [ 1.0,   -factor;...
          factor, 1.0      ];

    v = R * v;
%
%  Update the remaining angle.
%
    theta = theta - sigma * angle;

    poweroftwo = poweroftwo / 2.0;
%
%  Update the angle from table, or eventually by just dividing by two.
%
    if ( length ( angles ) < ( j + 1 ) )
```

```
      angle = angle / 2.0;
    else
      angle = angles(j+1);
    end

  end
%
%  Adjust length of output vector to be [cos(beta), sin(beta)]:
%
%  KPROD is essentially constant after a certain point, so if N is
%  large, just take the last available value.
%
%  if ( 0 < n )
%    v = v * kprod ( min ( n, length ( kprod ) ) );;
%  end
%
%  Adjust for possible sign change because angle was originally
%  not in quadrant 1 or 4.
%
%  v = sign_factor * v;
  xo = v(1);
  yo = v(2);
  zo = theta;

  return
end
```