

```
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:
```

```
% 2012.04.12
```

```
% 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 = [ ...
```

```
7.8539816339744830962E-01, ...
```

```
4.6364760900080611621E-01, ...
```

```
2.4497866312686415417E-01, ...
```

```
1.2435499454676143503E-01, ...
```

```
6.2418809995957348474E-02, ...
```

```
3.1239833430268276254E-02, ...
```

```
1.5623728620476830803E-02, ...
```

```
7.8123410601011112965E-03, ...
```

```
3.9062301319669718276E-03, ...
```

```
1.9531225164788186851E-03, ...
```

```
9.7656218955931943040E-04, ...
```

```
4.8828121119489827547E-04, ...
```

```
2.4414062014936176402E-04, ...
```

```
1.2207031189367020424E-04, ...
```

```
6.1035156174208775022E-05, ...
```

```
3.0517578115526096862E-05, ...
```

```
1.5258789061315762107E-05, ...
```

```
7.6293945311019702634E-06, ...
```

```
3.8146972656064962829E-06, ...
```

```
1.9073486328101870354E-06, ...
```

```
9.5367431640596087942E-07, ...
```

```
4.7683715820308885993E-07, ...
```

```
2.3841857910155798249E-07, ...
```

```
1.1920928955078068531E-07, ...
```

```
5.9604644775390554414E-08, ...
```

```
2.9802322387695303677E-08, ...
```

```
1.4901161193847655147E-08, ...
```

```
7.4505805969238279871E-09, ...
3.7252902984619140453E-09, ...
1.8626451492309570291E-09, ...
9.3132257461547851536E-10, ...
4.6566128730773925778E-10, ...
2.3283064365386962890E-10, ...
1.1641532182693481445E-10, ...
5.8207660913467407226E-11, ...
2.9103830456733703613E-11, ...
1.4551915228366851807E-11, ...
7.2759576141834259033E-12, ...
3.6379788070917129517E-12, ...
1.8189894035458564758E-12, ...
9.0949470177292823792E-13, ...
4.5474735088646411896E-13, ...
2.2737367544323205948E-13, ...
1.1368683772161602974E-13, ...
5.6843418860808014870E-14, ...
2.8421709430404007435E-14, ...
1.4210854715202003717E-14, ...
7.1054273576010018587E-15, ...
3.5527136788005009294E-15, ...
1.7763568394002504647E-15, ...
8.8817841970012523234E-16, ...
4.4408920985006261617E-16, ...
2.2204460492503130808E-16, ...
1.1102230246251565404E-16, ...
5.5511151231257827021E-17, ...
2.7755575615628913511E-17, ...
1.3877787807814456755E-17, ...
6.9388939039072283776E-18, ...
3.4694469519536141888E-18, ...
1.7347234759768070944E-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);

xn = [0];
yn = [0];

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;

    xn = [xn; v(1)];
    yn = [yn; v(2)];

%
% 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;  
  
newplot;  
hold on;  
  
plot(xn, yn, '-*')  
  
xn = [xn(1); xn(length(xn))];  
yn = [yn(1); yn(length(yn))];  
  
plot(xn, yn, '-r')  
  
hold off;  
pause  
  
return  
end
```