```
::::::::::::::
makefile
::::::::::::::
.SUFFIXES : .o .cpp .c

.cpp.o :
        g++ -c -g -I$HOME/include $<

.c.o :
        g++ -c -g -I$HOME/include $<

#---------------------------------------------------------------
OBJA = Angles.o                                \
       Angles.1.plot_circle_angle.o            \
       Angles.2.plot_line_angle.o              \
       Angles.3.calc_statistics.o              \
       Angles.4.plot_statistics.o              \
       Angles.5.plot_residual_errors.o         \
       Angles.6.calc_uscale_statistics.o       \
       Angles.7.plot_uscale_statistics.o       \
       Angles.8.plot_uscale_residual_errors.o  \
       Angles.9.plot_quantization.o            \
       Angles.a.compute_angle_arrays.o         \
       Angles.b.plot_angle_tree.o              \

OBJS = cordic_wt.o Angles_tb.o ${OBJA}         \

Angles.o : Angles.cpp Angles.hpp
        g++ -c -g -I$HOME/include Angles.cpp

Angles_tb : ${OBJS}
         g++ -o $@ ${OBJS} -lm

run_angles : Angles_tb
         ./Angles_tb   4 wxt 0

print_angles :
        /bin/more makefile Angles.hpp cordic_wt.hpp > print.file
        /bin/more ${OBJS:o=cpp}  >> print.file
        /bin/more QuadTree.hpp QuadTree.cpp QuadTree_tb.cpp > print.file2
        /bin/mv *.emf emf

tar_angles :
        mkdir src
        cp makefile Angles.hpp cordic_wt.hpp src
        cp ${OBJS:o=cpp} src
        tar cvf 7.cordic_accuracy.tar src
        \rm -fr src

#---------------------------------------------------------------
OBJ01 = cordic_wt.o cordic_tb01.o           \

cordic_wt.o : cordic_wt.cpp cordic_wt.hpp
        g++ -c -g -I$HOME/include cordic_wt.cpp

cordic_tb01.o : cordic_tb01.cpp cordic_wt.hpp
        g++ -c -g -I$HOME/include cordic_tb01.cpp

cordic_tb01 : ${OBJ01}
         g++ -o $@ ${OBJ01} -lm

run_tb01 : cordic_tb01
         ./cordic_tb01

print_tb01 :
        /bin/more makefile cordic_tb01.cpp cordic_wt.cpp cordic_wt.hpp > print.file
        /bin/mv *.emf emf
```

```
#---------------------------------------------------------------
OBJ02 = cordic_wt.o cordic_tb02.o ${OBJA}          \

cordic_tb02.o : cordic_tb02.cpp cordic_wt.hpp
        g++ -c -g -I$HOME/include cordic_tb02.cpp

cordic_tb02 : ${OBJ02}
         g++ -o $@ ${OBJ02} -lm

run_tb02 : cordic_tb02
         ./cordic_tb02  10 wxt 0

print_tb02 :
        /bin/more makefile cordic_tb02.cpp cordic_wt.cpp cordic_wt.hpp > print.file
        /bin/mv *.emf emf




#---------------------------------------------------------------
QuadTree.o : QuadTree.cpp
         g++ -c -g -I$HOME/include QuadTree.cpp

QuadTree_tb.o : QuadTree_tb.cpp
         g++ -c -g -I$HOME/include QuadTree_tb.cpp

QuadTree_tb : QuadTree.o QuadTree_tb.o
         g++ QuadTree_tb.o QuadTree.o -o QuadTree_tb -lm


#---------------------------------------------------------------



tar :
        mkdir src
        cp makefile Angles.hpp cordic.hpp src
        cp ${OBJS:o=cpp} src
        cp cordic_wt.hpp cordic_wt.cpp src
        tar cvf 7.cordic_accuracy.tar src
        \rm -fr src

EXES = Angles_tb cordic_tb01 cordic_tb02            \

clean :
        \rm -f *.o *~ *#
        \rm -f ${EXES}
        \rm -f *.emf



::::::::::::::
Angles.hpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <fstream>
# include <string>
// # include <cstdlib>
// # include <cmath>
# include <vector>
# include <algorithm>

using namespace std;
```

```
//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.23
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------

extern string GnuTerm;
const double pi = 3.141592653589793;
const double K = 1.646760258121;

class Angles
{

public:

  Angles();
  Angles(int nIter, int nAngle);
  ~Angles();

  void    setNIter(int nIter);
  void    setNAngle(int nAngle);
  void    setThreshold(double threshold);

  int     getNIter();
  int     getNAngle();
  double  getThreshold();


  //--------------------------------------------------------------------------
  // a. compute_angle                : compute angle and binary number string
  //    compute_angle_arrays         : init and compute array A[] & Ap[]
  // b. plot_angle_tree              : plot binary angle trees
  //--------------------------------------------------------------------------
  // 1. plot_circle_angle            : plot angle vectors on a unit circle
  // 2. plot_line_angle              : plot angle vectors on a linear scal
  // 3. calc_statistics              : find Angles Statistics  --> member data
  // 4. plot_statistics              : plot delta distribution and angle-delta
  // 5. plot_residual_errors         : plot residuals-angle and residuals-index
  // 6. calc_uscale_statistics
  // 7. plot_uscale_statistics
  // 8. plot_uscale_residual_errors
  // 9. plot_quantization            : plot non-uniform quantization of CORDIC
  //--------------------------------------------------------------------------

  /* a */ double compute_angle (int idx, int level, char *s);
  /*   */ void compute_angle_arrays ();
  /* b */ void plot_angle_tree ();
  /* 1 */ void plot_circle_angle ();
  /* 2 */ void plot_line_angle ();
  /* 3 */ void calc_statistics ();
  /* 4 */ void plot_statistics ();
  /* 5 */ void plot_residual_errors ();
  /* 6 */ void calc_uscale_statistics (double, double);
  /* 7 */ void plot_uscale_statistics ();
  /* 8 */ void plot_uscale_residual_errors ();
  /*   */ void plot_uscale_residual_errors (double, double);
  /* 9 */ void plot_quantization ();


  double *A;       // angle array
  char **Ap;       // angle path array


private:
  int     nIter;
  int     nAngle;
  int     Leaf;

  double  avg_delta;
  double  std_delta;
  double  min_delta;
  double  max_delta;
  double  min_angle;
  double  max_angle;

  double  ssr;      // sum of the squares of the residuals
  double  mse;      // mean squared error
  double  rms;      // root mean square error
  double  max_err;  // maximum of squared errors

  double threshold;


};



::::::::::::::
cordic_wt.hpp
::::::::::::::
/*------------------------------------------------------------------
void cordic_wt ( double *x, double *y, double *z, int n,
                 char *path, int *nBreak, int nBreakInit, double threshold
                 )
------------------------------------------------------------------*/

void cordic_wt ( double *x, double *y, double *z, int n,
                 char *, int *, int =0, double =0.0
                );
::::::::::::::
cordic_wt.cpp
::::::::::::::
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <cmath>
# include <ctime>

using namespace std;

# include "cordic_wt.hpp"


//--------------------------------------------------------------------------
void cordic_wt ( double *x, double *y, double *z, int n,
                 char *path, int *nBreak, int nBreakInit, double threshold
                 )
//--------------------------------------------------------------------------
//  CORDIC returns the sine and cosine using the CORDIC method.
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2012.04.17
//
//  Author:
//
//    Based on MATLAB code in a Wikipedia article.
//
//    Modifications by John Burkardt
//
//    Further modified by Young W. Lim
//
//  Parameters:
//
//    Input:
//      *x: x coord of an init vector
//      *y: y coord of an init vector
//      *z: angle (-90 <= angle <= +90)
//       n: number of iteration
//         A value of 10 is low.  Good accuracy is achieved
//         with 20 or more iterations.
//
//    Output:
//      *xo: x coord of a final vector
//      *yo: y coord of a final vector
//      *zo: angle residue
//
//  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) ).
//
//--------------------------------------------------------------------------
{
#define ANGLES_LENGTH 60
#define KPROD_LENGTH 33
#undef USE_ATAN
#define  USE_THRESHOLD



  double angles[ANGLES_LENGTH] = {
    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 };

  double kprod[KPROD_LENGTH] = {
    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 };

  double pi = 3.141592653589793;

  double angle;
  double factor;

  double sigma;
  double poweroftwo;
  double theta;

  double xn, yn;

  int j;

  //------------------------------------------------------------------------
  //  Initialize loop variables:
  //------------------------------------------------------------------------
  xn = *x;
  yn = *y;
  theta = *z;

  poweroftwo = 1.0;

#ifdef USE_ATAN
  angle = atan( 1. );
#else
  angle = angles[0];
#endif


  //------------------------------------------------------------------------
  //  Iterations
  //------------------------------------------------------------------------
  for ( j = 1; j <= n; j++ )
  {

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

      if ( theta < 0.0 ) path[j-1] = '0';
      else               path[j-1] = '1';

      factor = sigma * poweroftwo;

      *x =          xn - factor * yn;
      *y = factor * xn +          yn;

      xn = *x;
      yn = *y;

      //....................................................................
      //  Update the remaining angle.
      //....................................................................
      theta = theta - sigma * angle;
      *z = theta;


      //....................................................................
      // If residual angle is less than a given threshold, then break
      //....................................................................
#ifdef USE_THRESHOLD
      static int cntBreak = 0;
      if (nBreakInit == 0) cntBreak = 0;
      if (abs(*z) < threshold) {
        *nBreak = ++cntBreak;

#ifdef USE_THRESHOLD
        cout << "cntBreak= " << cntBreak;
        cout << "  z= " << right << setw(15) << *z;
        cout << " < " << right << setw(7) << threshold;
        cout << " j= " << right << setw(4) << j << endl;
#endif
        break;
      }
#endif

      //....................................................................
      //  Update the angle from table, or eventually by just dividing by two.
      //....................................................................
      poweroftwo = poweroftwo / 2.0;

#ifdef USE_ATAN
      if ( ANGLES_LENGTH < j+1 )  angle = angle / 2.0;
      else                        angle = angles[j];
#else
      angle = atan( 1. / (1 << j));
#endif

  }
  //------------------------------------------------------------------------
  // end of iteration
  //------------------------------------------------------------------------
  path[j-1] = '\0';


  //------------------------------------------------------------------------
  //  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 ( j > KPROD_LENGTH ) {
    *x = *x * kprod [ KPROD_LENGTH - 1 ];
    *y = *y * kprod [ KPROD_LENGTH - 1 ];
  }
  else {
    *x = *x * kprod [ j - 1 ];
    *y = *y * kprod [ j - 1 ];
  }

  //
  //  Adjust for possible sign change because angle was originally
  //  not in quadrant 1 or 4.
  //
  //   *c = sign_factor * *c;
  //   *s = sign_factor * *s;

  return;
# undef ANGLES_LENGTH
# undef KPROD_LENGTH
```

```
}

::::::::::::::
Angles_tb.cpp
::::::::::::::
# include <cstdlib>
# include <cmath>
# include <iostream>
# include <iomanip>
# include <fstream>

using namespace std;

# include "cordic_wt.hpp"
# include "Angles.hpp"

string GnuTerm;


//------------------------------------------------------------------------------
//   Purpose:
//
//     Explore Angles Space using Class Angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.23
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------



int main (int argc, char * argv[])
{

  // -------------------------------------------------------------
  // nIter   : Number of Iteration = Height of binary angle tree (< 32)
  // nAngle  : Number of Angles    = Number of Leaf Nodes
  // -------------------------------------------------------------
  int    nIter = 20;
  int    nAngle = 1 << nIter;
  double th = 0.0;



  GnuTerm = "wxt";
  // GnuTerm = "emf";

  if (argc > 1 ) {
    nIter = atoi(argv[1]);  // should be less than 31
    nAngle = 1 << nIter;
  }
  if (argc > 2) {
    GnuTerm = argv[2];
  }

  if (argc > 3) {
    th = atof(argv[3]);
  }

  cout << "----------------------------------------------------------\n";
  cout << "Angles_tb [nIter] [GnuTerm] [th]" << endl;
  cout << "----------------------------------------------------------\n";
  cout << "           nIter    = " << nIter << "  ";
  cout << "           nAngle   = " << nAngle << endl;
  cout << "           GnuTerm  = " << GnuTerm << endl;
  cout << "           th       = " << th << endl;
  cout << "----------------------------------------------------------\n";


  // -------------------------------------------------------------
  // LeafAngles : Angles Class for leaf nodes only
  // AllAngles  : Angles Class for all nodes (internal nodes included)
  // -------------------------------------------------------------
  Angles LeafAngles(nIter, nAngle);
  Angles AllAngles(nIter, 2*nAngle-1);


// #ifdef FULLSIM
  // -------------------------------------------------
  //   Plot Binary Angle Tree
  // -------------------------------------------------
  LeafAngles.plot_angle_tree ();
  AllAngles.plot_angle_tree ();



  // -------------------------------------------------
  //   Plot angle vectors on a unit circle
  // -------------------------------------------------
  LeafAngles.plot_circle_angle();
  AllAngles.plot_circle_angle();

  // -------------------------------------------------
  //   Plot angle vectors on a linear scale
  // -------------------------------------------------
  LeafAngles.plot_line_angle();
  AllAngles.plot_line_angle();
// #endif


  LeafAngles.setThreshold(th);
  AllAngles.setThreshold(th);


  // -------------------------------------------------
  //   Find Angles Statistics  --> member data
  // -------------------------------------------------
  LeafAngles.calc_statistics();
  AllAngles.calc_statistics();


  // -------------------------------------------------
  //   Plot Quantization Effects
  // -------------------------------------------------
  LeafAngles.plot_quantization();
  AllAngles.plot_quantization();
  // -------------------------------------------------
  //   Plot Delta Distribution & Angle-Delta
  // -------------------------------------------------
  LeafAngles.plot_statistics();
  AllAngles.plot_statistics();

  // -------------------------------------------------
  //   plot residual errors
  //   Residuals-Angle Plot & Residuals-Index Plot
  // -------------------------------------------------
  LeafAngles.plot_residual_errors();
  AllAngles.plot_residual_errors();



  // -------------------------------------------------
  //   Calculate Uniform Scale Statistics --> member data
  // -------------------------------------------------
  LeafAngles.calc_uscale_statistics(1.0, 1.0);
  AllAngles.calc_uscale_statistics(1.0, 1.0);


  // -------------------------------------------------
  //   Plot Uniform Scale Statistics
  // -------------------------------------------------
  LeafAngles.plot_uscale_statistics();
  AllAngles.plot_uscale_statistics();




  // -------------------------------------------------
  //   Plot residue errors at the leaf node angles
  // -------------------------------------------------
  LeafAngles.plot_uscale_residual_errors(1, 2);
  AllAngles.plot_uscale_residual_errors(1, 2);









  return 0;

}


::::::::::::::
Angles.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "cordic_wt.hpp"
# include "Angles.hpp"


using namespace std;
```

```
//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.23
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
//
// double compute_angle ( int idx, int nIter )
// Angles::Angles() : A(NULL), nIter(3), nAngle(8)
// void    Angles::setNIter(int nIter)
// void   Angles::setNAngle(int nAngle)
// void   Angles::setThreshold(double th)
// int    Angles::getNIter()
// int    Angles::getNAngle()
// double Angles::getThreshold()
// void draw_angle_tree (int nIter, int nAngle)
//
//-----------------------------------------------------------------------------







//-----------------------------------------------------------------------------
//  Class Angles' Member Functions
//-----------------------------------------------------------------------------
Angles::Angles() : nIter(3), nAngle(8)
{
  Leaf = 1;

  cout << "A is not initialized " << endl;
  cout << "nIter = " << nIter << endl;
  cout << "nAngle = " << nAngle << endl;

  avg_delta = std_delta = min_angle = max_angle = 0.0;
  ssr = mse = rms = max_err = 0.0;

  threshold = 0.0;

  compute_angle_arrays();

}


Angles::Angles(int nIter, int nAngle) :
  nIter(nIter), nAngle(nAngle)
{
  if (nAngle == (1 << nIter)) {
    Leaf = 1;
    cout << "A LeafAngles Object is created " ;
  } else {
    Leaf = 0;
    cout << "An AllAngles Object is created " ;
  }

  cout << "(nIter = " << nIter << ", ";
  cout << "nAngle = " << nAngle << ")" <<endl;

  avg_delta = std_delta = min_angle = max_angle = 0.0;
  ssr = mse = rms = max_err = 0.0;

  threshold = 0.0;

  compute_angle_arrays();

}

Angles::~Angles()
{

  free(A);
  for (int i=0; i < nAngle; i++) {
    free(Ap[i]);
  }
  free(Ap);

}

void Angles::setNIter(int nIter)
{
  nIter = nIter;
}


void Angles::setNAngle(int nAngle)
{
  nAngle = nAngle;
}

void Angles::setThreshold(double th)
{
  threshold = th;
}


int Angles::getNIter()
{
  return nIter;
}

int Angles::getNAngle()
{
  return nAngle;
}

double Angles::getThreshold()
{
  return threshold;
}



/*****
  for (i=0; i<20; i+=4) {
    for (j=0; j<4; ++j) {
      r = atan( 1. / (1 << (i+j)) ) / atan( 1. / (1 << i) ) * 100;
      cout << "index = " << i+j << " --> r = " << r << endl;
    }
  }

  return 0;
}
********************/




::::::::::::::
Angles.1.plot_circle_angle.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//     Plot angle vectors on the unit circle
//-----------------------------------------------------------------------------
void Angles::plot_circle_angle ()
{
  int i;
  ofstream myout;

  cout << "* plot_circle_angle ... " ;
  if (Leaf) cout << "(LeafAngles)" << endl;
  else cout << "(AllAngles)" << endl;

  if (nIter > 10) {
    cout << "nIter = " << nIter << " is too large to plot! " << endl;
    return;
  }


  // writing angle data on a unit circle
  myout.open("angle.dat");
  for (i=0; i<nAngle; i++) {
    myout << "0.0 0.0 " << cos(A[i]) << " " << sin(A[i]) << " " << endl;
  }
  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set output 'eg03.leaf.ang_circle.emf'" << endl;
    myout << "set title \"Leaf Angles on a unit circle \" " << endl;
  } else {
    myout << "set output 'eg03.all.ang_circle.emf'" << endl;
    myout << "set title \"All Angles on a unit circle \" " << endl;
  }
  myout << "set xlabel \"x\" " << endl;
  myout << "set ylabel \"y\" " << endl;
  myout << "set size square" << endl;
  myout << "set xrange [-1:+1]" << endl;
  myout << "set yrange [-1:+1]" << endl;
  myout << "set object 1 circle at 0, 0 radius 1" << endl;
  myout << "plot 'angle.dat' using 1:2:3:4  ";
  myout << "with vectors head filled lt 3" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");

  return;

}


::::::::::::::
Angles.2.plot_line_angle.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
```

```
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//     Plot angle vectors on a linear scale
//-----------------------------------------------------------------------------
void Angles::plot_line_angle ()
{

  ofstream myout;

  cout << "* plot_line_angle ... ";
  if (Leaf) cout << "(LeafAngles)" << endl;
  else cout << "(AllAngles)" << endl;

  if (nIter > 10) {
    cout << "nIter = " << nIter << " is too large to plot! " << endl;
    return;
  }

  // cout << "nIter  = " << nIter << endl;
  // cout << "nAngle = " << nAngle << endl;

  myout.open("angle.dat");

  for (int i=0; i<nAngle; ++i) {
    // cout << "A[" << i << "] = " << A[i] << endl;
    myout << A[i] << " 0.0 0.0 1.0" << endl;
  }

  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set title \"Leaf Angles on a linear scale\" " << endl;
    myout << "set output 'eg04.leaf.ang_line.emf'" << endl;
  } else {
    myout << "set title \"All Angles on a linear scale\" " << endl;
    myout << "set output 'eg04.all.ang_line.emf'" << endl;
  }
  myout << "set xlabel \"angles in radian\" " << endl;
  myout << "set ylabel \"\" " << endl;
  myout << "set yrange [0:+2]" << endl;
  myout << "plot 'angle.dat' using 1:2:3:4  ";
  myout << "with vectors head filled lt 3" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");

  return;
```

```
}


::::::::::::::
Angles.3.calc_statistics.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//   Find Angles Statistics  --> member data
//----------------------------------------------------------------------------
void Angles::calc_statistics ()
{

  vector <double> B, D;
  vector <double> ::iterator first, last;
  double mean, std;
  ofstream myout;


  cout << "* calc_statistics... ";
  if (Leaf) cout << "(LeafAngles)" << " nAngle = " << nAngle << endl;
  else cout << "(AllAngles)" << " nAngle = " << nAngle << endl;


  for (int i=0; i < nAngle; ++i) {
    // cout << "A[" << i << "]=" << setw(12) << setprecision(8) << A[i] << endl;
    // cout << "B[" << i << "]=" << setw(12) << setprecision(8) << B[i] << endl;
  }

  // B : sorted angles array
  for (int i=0; i < nAngle; ++i)
```

```
    B.push_back(A[i]);

  sort(B.begin(), B.end());


  // D : difference angle array
  for (int i=0; i < nAngle-1; ++i)
    D.push_back(B[i+1]- B[i]);

  sort(D.begin(), D.end());


  mean = 0.0;
  for (int i=0; i < D.size(); ++i)
    mean += D[i];
  mean /= D.size();

  std = 0.0;
  for (int i=0; i < D.size(); ++i)
    std += ((D[i]-mean) * (D[i]-mean));
  std /= D.size();
  std = sqrt(std);


  min_angle = B[0];
  max_angle = B[B.size()-1];
  min_delta = D[0];
  max_delta = D[D.size()-1];
  avg_delta = mean;
  std_delta = std;

  double udelta = (B[B.size()-1] - B[0]) /  nAngle; // computed unifrom delta


  cout << "  min angle      = " << min_angle << endl;
  cout << "  max angle      = " << max_angle << endl;
  cout << "  min delta      = " << min_delta << endl;
  cout << "  max delta      = " << max_delta << endl;
  cout << "  avg delta      = " << avg_delta << endl;
  cout << "  std delta      = " << std_delta << endl;
  cout << "  uniform delta  = " << udelta ;
  cout << "  = (max-min) / nAngle " << endl;



  return;
}


::::::::::::::
Angles.4.plot_statistics.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//------------------------------------------------------------------------------
//   Purpose:
//
```

```
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//   Plot Delta Distribution and  Angle-Delta
//-----------------------------------------------------------------------------
void Angles::plot_statistics ()
{

  vector <double> B, D;
  vector <double> ::iterator first, last;
  double mean, std;
  ofstream myout;


  cout << "* calc_statistics... ";
  if (Leaf) cout << "(LeafAngles)" << " nAngle = " << nAngle << endl;
  else cout << "(AllAngles)" << " nAngle = " << nAngle << endl;


  for (int i=0; i < nAngle; ++i) {
    // cout << "A[" << i << "]=" << setw(12) << setprecision(8) << A[i] << endl;
    // cout << "B[" << i << "]=" << setw(12) << setprecision(8) << B[i] << endl;
  }

  // B : sorted angles array
  for (int i=0; i < nAngle; ++i)
    B.push_back(A[i]);

  sort(B.begin(), B.end());


  // D : difference angle array
  for (int i=0; i < nAngle-1; ++i)
    D.push_back(B[i+1]- B[i]);

  sort(D.begin(), D.end());


  double udelta = (B[B.size()-1] - B[0]) /  nAngle; // computed unifrom delta

  // write histogram data from delta array
  myout.open("angle.dat");
  double pb ;
  for (int i=0, j, k; i<nAngle-2; i++) {
    j = i; k = 1;
    while ((D[j+1] - D[j])/D[j] < 0.01) {
      k++;
      j++;
    }
    pb = (double) k / D.size();
    myout << fixed << right << setw(12) << setprecision(7) << D[i] ;
    myout << " " << pb << endl;
    i = j;
  }
  myout.close();


  cout << "  + Delta Distribution Plot \n" ;

  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set output 'eg05.leaf.delta_dist.emf'" << endl;
    myout << "set title \"Delta Distribution of Leaf Angles\" " << endl;
  } else {
    myout << "set output 'eg05.all.delta_dist.emf'" << endl;
    myout << "set title \"Delta Distribution of All Angles\" " << endl;
  }
  myout << "set xlabel \"Delta (Adjacent Angle Difference)\" " << endl;
  myout << "set ylabel \"probability\" " << endl;
  myout << "set yrange [0:+1]" << endl;

  myout << "set arrow from " << avg_delta << ", 0";
  myout << " to " << avg_delta << ", 0.7" << endl;
  myout << "set label \"avg_delta \" at " << avg_delta;
  myout << ", 0.7 right" << endl;

  myout << "set arrow from " << udelta << ", 0";
  myout << " to " << udelta << ", 0.5" << endl;
  myout << "set label \"uniform delta \" at " << udelta;
  myout << ", 0.5 right" << endl;

  myout << "plot 'angle.dat' with linespoints" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");



  cout << "  + Angle-Delta Plot \n" ;

  // write angle-delta data
  myout.open("angle.dat");
  for (int i=0; i<B.size()-1; i++) {
    myout << B[i] << " " << B[i+1] - B[i] << endl;
  }
  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set output 'eg06.leaf.angle_delta.emf'" << endl;
    myout << "set title \"Angle-Delta Plot of Leaf Angles\" " << endl;
  } else {
    myout << "set output 'eg06.all.angle_delta.emf'" << endl;
    myout << "set title \"Angle-Delta Plot of All Angles\" " << endl;
  }
  myout << "set xlabel \"Angles in radian\" " << endl;
  myout << "set ylabel \"Delta (Adj Angle Diff) \" " << endl;
  myout << "set arrow from " << "-1.0, " << avg_delta;
  myout << " to " << "+1.0, " << avg_delta << endl;
  myout << "set label \"avg_delta \" at " << "+0.0, ";
  myout << avg_delta << " left" << endl;

  myout << "set arrow from " << "-1.0, " << udelta;
  myout << " to " << "+1.0, " << udelta << endl;
  myout << "set label \"uniform delta \" at " << "+1.0, ";
  myout << udelta << " left" << endl;

  myout << "set arrow from " << "-0.7853, " << min_delta;
  myout << "0 to -0.7853, " << max_delta  << endl;
  myout << "set label \"-pi/4 \" at " << "-0.7853, " ;
  myout << min_delta << " right " << endl;

  myout << "set arrow from " << "+0.7853, " << min_delta;
  myout << "0 to +0.7853, " << max_delta  << endl;
  myout << "set label \"+pi/4 \" at " << "+0.7853, " ;
  myout << min_delta << " left " << endl;

  myout << "plot 'angle.dat' with linespoints" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");


  return;
}


::::::::::::::
Angles.5.plot_residual_errors.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"
# include "cordic_wt.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
```

```
//    Young Won Lim
//
//  Parameters:
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//   plot residual errors
//   Residuals-Angle Plot and Residuals-Index Plot
//----------------------------------------------------------------------------
void Angles::plot_residual_errors ()
{

  int i;
  double x, y, z;
  ofstream myout;

  char path[32];


  cout << "* plot_residual_errors ... ";
  if (Leaf) cout << "(LeafAngles)" << endl;
  else cout << "(AllAngles)" << endl;

  if (nIter > 10) {
    cout << "nIter = " << nIter << " is too large to plot! " << endl;
    return;
  }


  // B : sorted angles array
  vector <double> B;
  vector <double> ::iterator first, last;

  for (int i=0; i < nAngle; ++i)
    B.push_back(A[i]);

  sort(B.begin(), B.end());


  // I=0 Use A[i] for Index-Residuals Plot
  // I=1 Use b[i] for Angle-Residuals Plot
  //..............................................................
  for (int I=0; I<2; I++) {
  //..............................................................

  // writing residue errors
  myout.open("angle.dat");

  int nBreak =0;

  // not member but local variables
  double se, ssr, mse, rms, max_err;
  se = ssr = mse = rms = max_err = 0.0;

  if (I==0) cout << "  + Index-Residuals Plot" << endl;
  else      cout << "  + Angle-Residuals Plot" << endl;

  for (i=0; i<nAngle; i++) {

    x = 1 / K;
    y = 0.0;
    if (I == 0) z = A[i];
    else        z = B[i];
    cordic_wt(&x, &y, &z, nIter, path, &nBreak, i, threshold);

    se = z * z;
    ssr += se;
    if (se > max_err) max_err = se;

    // cout << "A[" << i << "]= ";
    // cout << fixed << right << setw(10) << setprecision(7) << A[i];
    // cout << " z= " ;
    // cout << fixed << right << setw(10) << setprecision(7) << z << endl;

    myout << fixed << right << setw(10) << i;
    myout << fixed << right << setw(12) << setprecision(7);
    if (I==0) myout <<  A[i];
    else      myout <<  B[i];
    myout << fixed << right << setw(12) << setprecision(7) << z << endl;

  }

  mse = ssr / nAngle;
  rms = sqrt(mse);

  max_err = sqrt(max_err);


  cout << "  No of points = " << nAngle ;
  cout << " (nBreak = " << nBreak << " : " ;
  cout <<  100. * nBreak / nAngle << " % )" << endl;

  cout << "  SSR: Sum of Squared Residuals    = " ;
  cout << fixed << right << setw(12) << setprecision(7) << ssr << endl;
  cout << "  MSR: Mean Squared Residuals      = " ;
  cout << fixed << right << setw(12) << setprecision(7) << mse << endl;
  cout << "  RMS: Root Mean Squared Residuals = " ;
  cout << fixed << right << setw(12) << setprecision(7) << rms << endl;
  cout << "  Max Residual Error               = " ;
  cout << fixed << right << setw(12) << setprecision(7) << max_err << endl;


  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  if (I==0) {
    myout << "set terminal " << GnuTerm << endl;
    if (Leaf) {
      myout << "set output 'eg07.leaf.index_res.emf'" << endl;
      myout << "set title \"Index-Residual Plot (Leaf) \" " << endl;
    } else {
      myout << "set output 'eg07.all.index_res.emf'" << endl;
      myout << "set title \"Index-Residual Plot (All) \" " << endl;
    }
    myout << "set xlabel \"Index\" " << endl;
    myout << "set ylabel \"Residuals\" " << endl;
    myout << "plot 'angle.dat' using 1:3 with linespoints " << endl;
  } else {
    myout << "set terminal " << GnuTerm << endl;
    if (Leaf) {
      myout << "set output 'eg08.leaf.angle_res.emf'" << endl;
      myout << "set title \"Angle-Residual Plot (Leaf) \" " << endl;
    } else {
      myout << "set output 'eg08.all.angle_res.emf'" << endl;
      myout << "set title \"Angle-Residual Plot of (All)\" " << endl;
    }
    myout << "set xlabel \"Angles\" " << endl;
    myout << "set ylabel \"Residuals\" " << endl;
    myout << "plot 'angle.dat' using 2:3 with linespoints " << endl;
  }
  myout << "pause mouse keypress" << endl;

  myout.close();


  system("gnuplot command.gp");


  //..............................................................
  }
  //..............................................................

  return;

}


::::::::::::::
Angles.6.calc_uscale_statistics.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"
# include "cordic_wt.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
//   Calculate statistics on the uniform scale
//-----------------------------------------------------------------------------
//   ssr     : sum of the squares of the residuals
//   mse     : mean squared error
//   rms     : root mean square error
//   max_err : maximum of squared errors
//-----------------------------------------------------------------------------
```

```
//   ssr     : sum of the squares of the residuals
//   ang =  min_angle + avg_delta * offFactor ;
//   ang += (avg_delta / resFactor);
//-----------------------------------------------------------------------------
void Angles::calc_uscale_statistics (double resFactor, double offFactor)
{

  int n;
  double x, y, z;
  ofstream myout;

  char path[32];

  cout << "* calc_uscale_statistics ... ";
  if (Leaf) cout << "(LeafAngles Resolution)" << endl;
  else cout << "(AllAngles Resolution)" << endl;


  // sr      : square error of a data point

  double ang = min_angle + avg_delta * offFactor ;
  double se = 0.0 ;
  int nBreak =0;
  n = 0;


  ssr = mse = rms = max_err = 0.0;

  while (ang < max_angle) {
    x = 1 / K;
    y = 0.0;
    z = ang;


    cordic_wt(&x, &y, &z, nIter, path, &nBreak, n, threshold);

    se = (z * z);
    ssr += se;
    if (se > max_err) max_err = se;

    // cout << fixed << right << setw(10) << setprecision(7) << A[i];
    // cout << fixed << right << setw(10) << setprecision(7) << z << endl;

    ang += (avg_delta / resFactor);
    n++;

  }

  mse = ssr / n;
  rms = sqrt(mse);

  max_err = sqrt(max_err);


  cout << "  Angles = (" ;
  cout << min_angle << " : " << avg_delta << " : " << max_angle << ")" ;
  cout << endl << "  --> total " << n << " points" ;
  cout << " (nBreak = " << nBreak << " : " ;
  cout <<  100. * nBreak / n << " % )" << endl;

  cout << "  SSR: Sum of Squared Residuals    = " ;
  cout << fixed << right << setw(12) << setprecision(7) << ssr << endl;
  cout << "  MSR: Mean Squared Residuals      = " ;
  cout << fixed << right << setw(12) << setprecision(7) << mse << endl;
  cout << "  RMS: Root Mean Squared Residuals = " ;
  cout << fixed << right << setw(12) << setprecision(7) << rms << endl;
  cout << "  Max Residual Error               = " ;
  cout << fixed << right << setw(12) << setprecision(7) << max_err << endl;


  return;

}

::::::::::::::
Angles.7.plot_uscale_statistics.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//   Plot uniform scale statistics
//-----------------------------------------------------------------------------
//   ang =  min_angle + avg_delta * offFactor ;
//   ang += (avg_delta / resFactor);
//-----------------------------------------------------------------------------
void Angles::plot_uscale_statistics ( )
{

  int    i, j;
  double resFactor, offFactor;
  ofstream myout;

  cout << "* plot_uscale_statistics ... ";
  if (Leaf) cout << "(LeafAngles Resolution)" << endl;
  else cout << "(AllAngles Resolution)" << endl;

  if (nIter > 10) {
    cout << "nIter = " << nIter << " is too large to plot! " << endl;
    return;
  }

  // ssr : sum of the squares of the residuals
  // mse : mean squared error
  // rms : root mean square error
  // sr  : square error of a data point
  // max_err : maximum of squared errors


  int M=4;  // no of resFactor's
  int N=4;  // no of offFactor's
  double SSR[M][N];

  for (i=0; i<M; i++) {
    resFactor = i + 1.0;

    for (j=0; j<N; j++) {
      offFactor = (j+1) / N;

      cout << " ======= resFactor= " << i << ",   offFactor= " << j ;
      cout << " =======" << endl;

      calc_uscale_statistics(resFactor, offFactor);

      SSR[i][j] = ssr;
    }
  }



  // writing residue errors
  myout.open("angle.dat");

  for (i=0; i<M; i++) {
      cout << i+1  << " ";
      myout << i+1  << " ";
    for (j=0; j<N; j++) {
      cout << SSR[i][j] << " ";
      myout << SSR[i][j] << " ";
    }
    cout << endl;
    myout << endl;
  }

  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  myout << "set autoscale y" << endl;
  if (Leaf) {
    myout << "set output 'eg09.leaf.resfac_ssr.emf'" << endl;
    myout << "set title \"resFactor-SSR Plot (Leaf) \" " << endl;
  } else {
    myout << "set output 'eg09.all.resfac_ssr.emf'" << endl;
    myout << "set title \"resFactor-SSR Plot (All) \" " << endl;
  }
  myout << "set xlabel \"resFactor\" " << endl;
  myout << "set ylabel \"SSR (Sum of Squared Residuals)\" " << endl;
  myout << "plot " ;
  for (j=0; j<N; j++) {
    myout << " 'angle.dat' using 1:" << j+2 ;
    myout << " title 'offFactor=" << (j+1) << "/" << N << "' ";
    myout << " with linespoints" ;
    if (j<N-1) myout << ", "; else myout << endl;
  }
```

```
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");




  // writing residue errors
  myout.open("angle.dat");

  for (j=0; j<N; j++) {
      cout << j+1  << " ";
      myout << j+1  << " ";
    for (i=0; i<M; i++) {
      cout << SSR[i][j] << " ";
      myout << SSR[i][j] << " ";
    }
    cout << endl;
    myout << endl;
  }

  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  myout << "set autoscale y" << endl;
  myout << "set autoscale y" << endl;
  if (Leaf) {
    myout << "set output 'eg10.leaf.offfac_ssr.emf'" << endl;
    myout << "set title \"offFactor-SSR Plot (Leaf) \" " << endl;
  } else {
    myout << "set output 'eg10.all.offfac_ssr.emf'" << endl;
    myout << "set title \"offFactor-SSR Plot (All) \" " << endl;
  }
  myout << "set xlabel \"offFactor\" " << endl;
  myout << "set ylabel \"SSR (Sum of Squared Residuals)\" " << endl;
  myout << "plot " ;
  for (i=0; i<M; i++) {
    myout << " 'angle.dat' using 1:" << i+2 ;
    myout << " title 'resFactor=" << (i+1) << "' ";
    myout << " with linespoints" ;
    if (i<M-1) myout << ", "; else myout << endl;
  }

  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");





  return;

}



::::::::::::::
Angles.8.plot_uscale_residual_errors.cpp
```

```
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"
# include "cordic_wt.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//   Plot residual errors on the uniform scale
//-----------------------------------------------------------------------------
//   ang =  min_angle + avg_delta * offFactor ;
//   ang += (avg_delta / resFactor);
//-----------------------------------------------------------------------------
void Angles::plot_uscale_residual_errors
(double resFactor, double offFactor )
{

  int n;
  double x, y, z;
  ofstream myout;

  char path[32];

  cout << "* plot_uscale_residual_errors ... ";
  if (Leaf) cout << "(LeafAngles Resolution)" << endl;
  else cout << "(AllAngles Resolution)" << endl;

  if (nIter > 10) {
    cout << "nIter = " << nIter << " is too large to plot! " << endl;
    return;
  }

  double ang = min_angle + avg_delta * offFactor ;
  double se = 0.0 ;
  int nBreak =0;
  n = 0;

  // writing residue errors
  myout.open("angle.dat");

  while (ang < max_angle) {
    x = 1 / K;
    y = 0.0;
    z = ang;


    cordic_wt(&x, &y, &z, nIter, path, &nBreak, n, threshold);


    se = (z * z);

    // cout << fixed << right << setw(10) << setprecision(7) << A[i];
    // cout << fixed << right << setw(10) << setprecision(7) << z << endl;

    myout << fixed << right << setw(10) << n;
    myout << fixed << right << setw(22) << setprecision(7) << ang;
    myout << fixed << right << setw(22) << setprecision(7) << z;
    myout << fixed << right << setw(22) << setprecision(7) << se << endl;

    ang += (avg_delta / resFactor);
    n++;

  }

  cout << "  Angles = (" ;
  cout << min_angle << " : " << avg_delta << " : " << max_angle << ")" ;
  cout << endl << "  --> total " << n << " points" ;
  cout << " (nBreak = " << nBreak << " : " ;
  cout <<  100. * nBreak / n << " % )" << endl;

  myout.close();



  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set autoscale y" << endl;
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set output 'eg11.leaf.idx_res_us.emf'" << endl;
    myout << "set title \"Index-Residual Plot (uScale-Leaf)\"" << endl;
  } else {
    myout << "set output 'eg11.all.idx_res_us.emf'" << endl;
    myout << "set title \"Index-Residual Plot (uScale-All)\"" << endl;
  }
  myout << "set xlabel \"Index\" " << endl;
  myout << "set ylabel \"Residuals\" " << endl;
  myout << "plot 'angle.dat' using 1:3 with linespoints " << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set autoscale y" << endl;
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set output 'eg12.leaf.ang_res_us.emf'" << endl;
    myout << "set title \"Angle-Residual Plot (uScale-Leaf)\"" << endl;
  } else {
    myout << "set output 'eg12.all.ang_res_us.emf'" << endl;
    myout << "set title \"Angle-Residual Plot (uScale-All)\"" << endl;
  }
  myout << "set xlabel \"Angle\" " << endl;
  myout << "set ylabel \"Residuals\" " << endl;
  myout << "plot 'angle.dat' using 2:3 with linespoints " << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");


  return;

}



void Angles::plot_uscale_residual_errors ()
{

  plot_uscale_residual_errors (1.0, 1.0);

}


::::::::::::::
Angles.9.plot_quantization.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"
# include "cordic.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.17
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------
//   Plot Non-uniform Quantization of CORDIC
//-----------------------------------------------------------------------------
void Angles::plot_quantization ()
{


  vector <double> B, D;
  vector <double> ::iterator first, last;
  double mean, std;
  ofstream myout;


  cout << "* calc_statistics... ";
  if (Leaf) cout << "(LeafAngles)" << " nAngle = " << nAngle << endl;
  else cout << "(AllAngles)" << " nAngle = " << nAngle << endl;


  for (int i=0; i < nAngle; ++i) {
    // cout << "A[" << i << "]=" << setw(12) << setprecision(8) << A[i] << endl;
    // cout << "B[" << i << "]=" << setw(12) << setprecision(8) << B[i] << endl;
  }

  // B : sorted angles array
  for (int i=0; i < nAngle; ++i)
    B.push_back(A[i]);

  sort(B.begin(), B.end());


  // D : difference angle array
  for (int i=0; i < nAngle-1; ++i)
    D.push_back(B[i+1]- B[i]);

  sort(D.begin(), D.end());


  double udelta = (B[B.size()-1] - B[0]) /  nAngle; // computed unifrom delta

  // write histogram data from delta array
  myout.open("angle.dat");
  double pb ;
  for (int i=0; i<nAngle; i++) {
    myout << fixed << right << setw(12) << setprecision(7) << B[0] + udelta*i ;
    myout << fixed << right << setw(12) << setprecision(7) << B[0] + udelta*i ;
    myout << fixed << right << setw(12) << setprecision(7) << B[i] ;
    myout << " " << endl;
  }
  myout.close();


  cout << "  + Quantization Effect Plot \n" ;

  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  if (Leaf) {
    myout << "set output 'eg12.leaf.quantization.emf'" << endl;
    myout << "set title \"Quantization Effect of Leaf Angles\" " << endl;
  } else {
    myout << "set output 'eg12.all.quantization.emf'" << endl;
    myout << "set title \"Quantization Effect of All Angles\" " << endl;
  }
  myout << "set xlabel \"Delta (Adjacent Angle Difference)\" " << endl;
  myout << "set ylabel \"Quantized Angles\" " << endl;
  myout << "set yrange [" << B[0] << ":" << B[B.size()-1] << "]" << endl;

  myout << "plot 'angle.dat' using 1:2 with lines, ";
  myout << " 'angle.dat' using 1:3 with lines" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();


  system("gnuplot command.gp");



  return;

}


::::::::::::::
Angles.a.compute_angle_arrays.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//------------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.23
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//  Compute an angle value and binary string based on the binary tree
//    idx - index for leaf nodes [0..2^level -1]
//    level - the level of the binary angle tree
//    s[] - binary number string for the number idx
//------------------------------------------------------------------------------
double Angles::compute_angle (int idx, int level, char *s)
{
  int    i, j;
  double angle;
  // i - bit position starting from msb
  // j = 2^i
  // (idx & (1 << (level-i-1))) - i-th bit of idx from msb
  // if each bit is '1', add atan(1/2^i)
  // if each bit is '0', sub atan(1/2^i)
  // s[32] contains the binary representation of idx

  angle = 0.0;
  for (i=0; i<level; i++) {
    j = 1 << i;
    if (idx & (1 << (level-i-1))) {
      angle += atan( 1. / j );
      s[i] = '1';
    } else {
      angle -= atan( 1. / j );
      s[i] = '0';
    }
    // cout << "i=" << i << " j=" << j << " 1/j=" << 1./j
    //      << " atan(1/j)=" << atan(1./j)*180/3.1416 << endl;
  }
  s[i] = '\0';

  // cout << level << " " << idx << " " << s
  //      << " ---> " << angle*180/3.1416 << endl;

  return angle;
}

//-----------------------------------------------------------------------------
//  Initialize and compute the arrays A[] and Ap[][]
//-----------------------------------------------------------------------------
void Angles::compute_angle_arrays ()
{

  A   = (double *) malloc (nAngle * sizeof (double));
  Ap   = (char **) malloc (nAngle * sizeof (char *));
  for (int i=0; i < nAngle; i++) {
    Ap[i] = (char *) malloc (256 * sizeof (char));
  }

  char   s[256];
  int    i, j;
  int    k, level, leaves;

  if (Leaf) {
    for (j=0; j<nAngle; ++j) {
      A[j] = compute_angle(j, nIter, Ap[j]);
      // cout << "A[" << j << "]=" << setw(12) << setprecision(8) << A[j] << endl;
    }
  }
  else {
    k=0;
    for (i=0; i<=nIter; ++i) {
      level = i;
      leaves = 1 << level;
      // cout << "level = " << level << "leaves = " << leaves << endl;
      for (j=0; j<leaves; ++j) {
        A[k+j] = compute_angle(j, level, Ap[k+j]);
        // cout << "A[" << j+k << "] = " << A[j+k] << endl;
      }
      k += leaves;
    }
  }

}
```

```
::::::::::::::
Angles.b.plot_angle_tree.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>

# include "Angles.hpp"

using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.01.23
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
//  Plot binary angle trees for Leaf and All node cases
//-----------------------------------------------------------------------------
void Angles::plot_angle_tree ()
{

  int level, leaves;
  int i, j, k;
  ofstream myout;


  if (nIter > 10) {
    cout << "nIter = " << nIter << " is too large to plot! " << endl;
    return;
  }

  // cout << "nIter  = " << nIter << endl;
  // cout << "nAngle = " << nAngle << endl;

  //---------------------------------------
  if (Leaf) {
  //---------------------------------------
  // Binary Angle Tree (Leaf)
  //---------------------------------------
  myout.open("angle.dat");

  level = nIter;
  for (i=0; i<level; ++i) {
    leaves = 1 << nIter;
    for (j=0; j<leaves; ++j) {
      // cout << "A[" << j << "] = " << A[j] << endl;
      myout << A[j]*180/pi << " " <<  i << " 0.0 1.0" << endl;
    }
  }

  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  myout << "set output 'eg01.bin_ang_tree.emf'" << endl;
  myout << "set title \"Binary Angle Tree\" " << endl;
  myout << "set xlabel \"Angles in degree\" " << endl;
  myout << "set ylabel \"Levels \" " << endl;
  myout << "set format x \"%.0f\" " << endl;
  myout << "set format y \"%.0f\" " << endl;
  myout << "plot 'angle.dat' using 1:2:3:4  ";
  myout << "with vectors head filled lt 3" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();

  system("gnuplot command.gp");



  //---------------------------------------
  } else {
  //---------------------------------------
  // Cumulative Angle Tree (All)
  //---------------------------------------

  myout.open("angle.dat");

  k=0;
  for (i=0; i<=nIter; ++i) {
    level = i;
    leaves = 1 << level;
    // cout << "level = " << level << "leaves = " << leaves << endl;
    for (j=0; j<leaves; ++j) {
      // cout << "A[" << k+j << "] = " << A[k+j] << endl;
      myout << A[k+j]*180/pi << " " <<  i << " 0.0 1.0" << endl;
    }
    k += leaves;
  }

  myout.close();


  // writing gnuplot commands
  myout.open("command.gp");
  myout << "set terminal " << GnuTerm << endl;
  myout << "set output 'eg02.cumul_ang_tree.emf'" << endl;
  myout << "set title \"Cumulative Binary Angle Tree\" " << endl;
  myout << "set xlabel \"Angles in degree\" " << endl;
  myout << "set ylabel \"Levels \" " << endl;
  myout << "set format x \"%.0f\" " << endl;
  myout << "set format y \"%.0f\" " << endl;
  myout << "plot 'angle.dat' using 1:2:3:4  ";
  myout << "with vectors head filled lt 4" << endl;
  myout << "pause mouse keypress" << endl;
  myout.close();

  system("gnuplot command.gp");

  //---------------------------------------
  }
  //---------------------------------------

  return;

}
```