```
::::::::::::::
Angles.make
::::::::::::::
#---------------------------------------------------------------
# copy include files    ${INC} into the directory ${INCD}
# copy library files    ${LIB} into the directory ${LIBD}
# copy executable files ${EXE} into the directory ${EXED}
# include files in ${INCS} directories to compile this module
#---------------------------------------------------------------
INCD = /home/young/MyWork/inc
LIBD = /home/young/MyWork/lib
EXED = /home/young/MyWork/exe

VPATH = ../Class.Core:../Class.Figures:../Class.GPData

INCS = -I../Class.Core    \
       -I../Class.Figures \
       -I../Class.GPData  \


.SUFFIXES : .o .cpp .c

.cpp.o :
        g++ -c -Wall -g ${INCS} $<

.c.o :
        g++ -c -Wall g ${INCS} $<


#---------------------------------------------------------------
# Classes
#---------------------------------------------------------------
SRC  = Angles.cpp   Angles.hpp                          \
       Angles.1.b1.plot_angle_tree.cpp                  \
       Angles.1.b2.plot_circle_angle.cpp                \
       Angles.1.b3.plot_line_angle.cpp                  \
       Angles.1.b4.plot_quantization.cpp                \
       Angles.2.t1.calc_tscale_statistics.cpp           \
       Angles.2.t2.plot_tscale_statistics.cpp           \
       Angles.2.t3.plot_tscale_residual_angles.cpp      \
       Angles.3.u1.calc_uscale_statistics.cpp           \
       Angles.3.u2.plot_uscale_statistics.cpp           \
       Angles.3.u3.plot_uscale_residual_angles.cpp      \
       Angles.3.u4.plot_uscale_histogram.cpp            \
       Angles.a.compute_angle_arrays.cpp                \

OBJ  = Angles.o                                         \
       Angles.1.b1.plot_angle_tree.o                   \
       Angles.1.b2.plot_circle_angle.o                 \
       Angles.1.b3.plot_line_angle.o                   \
```

```
       Angles.1.b4.plot_quantization.o            \
       Angles.2.t1.calc_tscale_statistics.o       \
       Angles.2.t2.plot_tscale_statistics.o       \
       Angles.2.t3.plot_tscale_residual_angles.o  \
       Angles.3.u1.calc_uscale_statistics.o       \
       Angles.3.u2.plot_uscale_statistics.o       \
       Angles.3.u3.plot_uscale_residual_angles.o  \
       Angles.3.u4.plot_uscale_histogram.o        \
       Angles.a.compute_angle_arrays.o            \


INC = Angles.hpp                        \

LIB = libcordic-angles.a                \

EXE = Angles_tb                         \


#---------------------------------------------------------------
Angles.o : ${SRC}
        g++ -c -Wall -g ${INCS} Angles.cpp

#---------------------------------------------------------------
all : ${OBJ}
#       ar -rcs libcordic-angles.a ${OBJ}
        ar -cvq libcordic-angles.a ${OBJ}
        \cp -f ${INC} ${INCD}
        \cp -f ${LIB} ${LIBD}
        \rm -f ${OBJ}


print : Angles.make ${SRC}
        /bin/more $? > Angles.print

tar : Angles.make ${SRC}
        tar cvf Angles.tar $?

clean :
        \rm -f *.o *~ *#
::::::::::::::
Angles.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstring>
```

```
#include <string>

#include "Core.hpp"
#include "Angles.hpp"


using namespace std;


//------------------------------------------------------------------------------
//   Purpose:
//
//     Angles Class Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.02.20
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------
//
//        Angles::Angles() : A(NULL), nIters(3), nAngles(8)
// void   Angles::setnIters(int nIters)
// void   Angles::setnAngles(int nAngles)
// void   Angles::setThreshold(double th)
// int    Angles::getnIters()
// int    Angles::getnAngles()
// double Angles::getThreshold()
//
// double compute_threshold(int nIters)
//
//------------------------------------------------------------------------------



//------------------------------------------------------------------------------
//  Class Angles' Member Functions
//------------------------------------------------------------------------------
```

```
Angles::Angles() : nIters(10), nAngles(1024)
{
  Leaf = 1;

  cout << "Default LeafAngles Object is created " ;

  Angles(nIters, nAngles);

}

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

  cout << "(nIters = " << nIters << ", ";
  cout << "nAngles = " << nAngles << ")" <<endl;

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

  threshold = 0.0;



  //................................................
  A   = (double *) calloc (nAngles, sizeof (double));
  B   = (double *) calloc (nAngles, sizeof (double));
  Ap   = (char **) calloc (nAngles, sizeof (char *));
  for (int i=0; i < nAngles; i++) {
    Ap[i] = (char *) calloc (256, sizeof (char));
  }
  //................................................
  compute_angle_arrays();
  //................................................


}


//................................................
Angles::~Angles()
{
```

```
  S.ARm.clear();   // map       : angle - residual
  S.ADm.clear();   // map       : angle - difference (of adjacent residuals)
  S.RAmm.clear();  // multimap  : residual   - angle
  S.DAmm.clear();  // multimap  : difference - angle
  S.HRCm.clear();  // map       : residual   - count for a histogram
  S.HDCm.clear();  // map       : difference  -count for a histogram

  S.R.clear();


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

}

//..................................................
uStat::uStat()
{

}

uStat::~uStat()
{


}


//--------------------------------------------------------------------------------
double compute_threshold(int nIters)
{

  int nAngles = 1 << nIters;

  Angles AllAngles(nIters, 2*nAngles-1);

  AllAngles.calc_tscale_statistics();  /* 3  */

  double th = AllAngles.get_avg_delta();

  // th = (AllAngles.get_max_angle() - AllAngles.get_max_angle();
  // th /= AllAngles.getnAngles();

  cout << "* Computed threshold is to be used : " << th << endl;
  return th;
```

```
}


//------------------------------------------------------------------------------
int Angles::checkNIters(string str)
{

   printf("* %s ...\n", str.c_str());

   if (Leaf) printf("(LeafAngles) nAngles=%d nIters=%d \n", nAngles, nIters);
   else      printf("(AllAngles)  nAngles=%d nIters=%d \n", nAngles, nIters);

   if (nIters > 20) {
     printf("nIters=%d is too large to plot!!! \n", nIters);
     return -1;
   } else {
     return 0;
   }
}



/*****
  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.hpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <fstream>
# include <string>
// # include <cstdlib>
// # include <cmath>
# include <vector>
# include <algorithm>
# include <map>
# include <list>
```

```
using namespace std;


//------------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.02.20
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------
//  Defined Classes
//  class XRange
//  class uStat
//  class Angles
//------------------------------------------------------------------------------

extern string GnuTerm;
extern string ofExt;


const double pi = 3.141592653589793;
const double K = 1.646760258121;



// to pass parameters use class uStat
typedef map<double, double> Map;
typedef Map::iterator mI;

typedef multimap<double, double> MMap;
typedef MMap::iterator mmI;


//------------------------------------------------------------------------
```

```
class uStat {
  public:
  uStat();
  ~uStat();

  Map  ARm;   // map       : angle - residual
  Map  ADm;   // map       : angle - difference (of adjacent residuals)
  MMap RAmm;  // multimap  : residual   - angle
  MMap DAmm;  // multimap  : difference - angle
  Map  HRCm;  // map       : residual   - count for a histogram
  Map  HDCm;  // map       : difference  -count for a histogram

  vector<double> R;

  double min_ang;
  double max_ang;

  double min_res;
  double max_res;
  double avg_res;
  double std_res;

  double min_diff;
  double max_diff;
  double avg_diff;
  double std_diff;

  double step_ang;
  double rms_res;
  double max_freq_res;
  double max_freq_diff;
};


//----------------------------------------------------------------------
class XRange {
  public:
    float xmin;
    float xmax;
    int   nPartitions;
    int   partitionIndex;
};



//----------------------------------------------------------------------
class Angles
{

public:
```

```
Angles();
Angles(int nIters, int nAngles);
~Angles();


void    setnIters     (int    val) { nIters     = val;};
void    setnAngles    (int    val) { nAngles    = val;};
void    setnPoints    (int    val) { nPoints    = val;};
void    setThreshold (double val) { threshold = val;};
void    setUseTh     (int    val) { useTh     = val;};
void    setUseThDisp (int    val) { useThDisp = val;};
void    setUseATAN   (int    val) { useATAN   = val;};

int     getnIters     () { return nIters;     };
int     getnAngles    () { return nAngles;    };
int     getnPoints    () { return nPoints;    };
double  getThreshold () { return threshold; };
int     getUseTh     () { return useTh;     };
int     getUseThDisp () { return useThDisp; };
int     getUseATAN   () { return useATAN;   };

int     getLeaf      () { return Leaf;      };

//---------------------------------------------------------------------------
// 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_tscale_statistics       : find Angles Statistics  --> member data
// 4. plot_tscale_statistics       : plot delta distribution and angle-delta
//*5. plot_tscale_residual_angles  : plot residuals-angle and residuals-index
//*6. calc_uscale_statistics
// 7. plot_uscale_statistics
//*8. plot_uscale_residual_angles
// 9. plot_quantization            : plot non-uniform quantization of CORDIC
//---------------------------------------------------------------------------
//*:  call cordic()
//---------------------------------------------------------------------------

/* a */ double compute_angle (int idx, int level, char *s);
/*   */ void compute_angle_arrays ();
/* 1b1 */ void plot_angle_tree (int, int);
/* 1b2 */ void plot_circle_angle ();
/* 1b3 */ void plot_line_angle ();
/* 1b4 */ void plot_quantization ();
/* 2t1 */ void calc_tscale_statistics ();
/* 2t2 */ void plot_tscale_statistics (int);
```

```
/* 2t3 */ void plot_tscale_residual_angles ();
/* 3u1 */ void calc_uscale_statistics (int);
/* 3u2 */ void plot_uscale_statistics (int);
/* 2u3 */ void plot_uscale_residual_angles (int);
/* 3u4 */ void plot_uscale_histogram (int);


uStat S;

double *A;                    // angle array
double *B;                    // sorted angle array

char **Ap;                    // angle path array


double  get_avg_delta () { return avg_delta; };
double  get_std_delta () { return std_delta; };
double  get_min_delta () { return min_delta; };
double  get_max_delta () { return max_delta; };
double  get_min_angle () { return min_angle; };
double  get_max_angle () { return max_angle; };

double  get_ssr ()      { return ssr; };          // sum of the squares of the residuals
double  get_mse ()      { return mse; };          // mean squared error
double  get_rms ()      { return rms; };          // root mean square error
double  get_max_err ()  { return max_err; };      // maximum of squared errors

double  get_threshold () { return threshold; };



void  set_avg_delta (double val) { avg_delta = val; };
void  set_std_delta (double val) { std_delta = val; };
void  set_min_delta (double val) { min_delta = val; };
void  set_max_delta (double val) { max_delta = val; };
void  set_min_angle (double val) { min_angle = val; };
void  set_max_angle (double val) { max_angle = val; };

void  set_ssr (double val)      { ssr = val; };          // sum of the squares of the residuals
void  set_mse (double val)      { mse = val; };          // mean squared error
void  set_rms (double val)      { rms = val; };          // root mean square error
void  set_max_err (double val)  { max_err = val; };      // maximum of squared errors

void  set_threshold (double val) { threshold = val; };


int   is_tscale_stat_done()             {return tscale_stat_done; };
int   is_uscale_stat_done()             {return uscale_stat_done; };

void  set_tscale_stat_done(int val)     { tscale_stat_done =1; };
```

```
  void  set_uscale_stat_done(int val)     { uscale_stat_done =1; };

  int   checkNIters(string str);

  list<string> epsList;

private:
  int     nIters;    // number of iterations (levels)
  int     nAngles;   // numuber of nodes in binary angle tree (leaves or all nodes)
  int     nPoints;   // number of angle points (uniform scale)
  int     Leaf;

  int     useTh;
  int     useThDisp;
  int     useATAN;

  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;

  int tscale_stat_done;
  int uscale_stat_done;

};


double compute_threshold(int nIters);
void make_tex_output();

  //---------------------------------------------------------------------------
  //
  // ____ : leaf/all
  //
  //---------------------------------------------------------------------------
  // 1.b1 plot_angle_tree              : plot binary angle tree
  //      egb1.____.ang_tree.eps
  //---------------------------------------------------------------------------
  // 1.b2 plot_circle_angle            : plot angle vectors on a circle
  //      egb2.____.circle_ang.eps
  //---------------------------------------------------------------------------
```

```
  // 1.b3 plot_line_angle              : plot angle vectors on a line
  //      egb3.____.ang_line.i1.eps
  //----------------------------------------------------------------------------
  // 1.b4 plot_quantization            : plot quantization effects
  //      egb4.____.quantization.eps
  //----------------------------------------------------------------------------
  //
  //
  //----------------------------------------------------------------------------
  // 2.t1 calc_tscale_statistics       : find Angles Statistics  --> member data
  //----------------------------------------------------------------------------
  // 2.t2 plot_tscale_statistics       : plot delta distribution and angle-delta
  //      egt2.____.t_delta_dist_0.[th0.001].eps  (histogram of the delta's)
  //      egt2.____.t_delta_dist_1.[th0.001].eps  (histogram of the delta's)
  //      egt2.____.t_delta_vs_angle.[th0.001].eps (delta vs angle)
  //----------------------------------------------------------------------------
  //*2.t3 plot_tscale_residual_angles  : plot residuals-angle and residuals-index
  //      egt3.____.t_res_vs_angle.n1000.x1.[th0.001].eps (residual angles vs angles)
  //      egt3.____.t_res_vs_index.n1000.x1.[th0.001].eps (residual angles vs index)
  //----------------------------------------------------------------------------
  //
  //
  //----------------------------------------------------------------------------
  //*3.u1 calc_uscale_statistics
  //----------------------------------------------------------------------------
  // 3.u2 plot_uscale_statistics
  //      egu2.____.u_dff_hist.n%d.x%d.--%f.ext
  //      egu2.____.u_res_hist.n%d.x%d.--%f.ext
  //      egu2.____.u_angle_vs_dff.n%d.x%d.--%f.ext
  //      egu2.____.u_angle_vs_res.n%d.x%d.--%f.ext
  //----------------------------------------------------------------------------
  //*3.u4 plot_uscale_histogram
  //      egu4.____.u_dff_vs_angle.n%d.x%d.--%f.ext
  //      egu4.____.u_res_vs_angle.n%d.x%d.--%f.ext
  //      egu4.____.u_dff_corr_vs_angle.n%d.x%d.--%f.ext
  //      egu4.____.u_res_corr_vs_angle.n%d.x%d.--%f.ext
  //----------------------------------------------------------------------------


::::::::::::::
Angles.1.b1.plot_angle_tree.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstring>
```

```
#include "Angles.hpp"
#include "GPData.hpp"

using namespace std;

//------------------------------------------------------------------------------
//  Purpose: Class Angles Implementation Files
//
//      [Angles.1.b1.plot_angle_tree.cpp]
//
//       Angles::plot_angle_tree()
//       - to plot a binary tree angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//       m : m-th level
//       n : n-th node in the m-th level
//
//  Outputs:
//       egb1.____.ang_tree.eps
//
//------------------------------------------------------------------------------
void B1_plot_subtree_leaf(int m, int mode, char * fname, int nIters, double * A);
void B1_plot_subtree_all(int m, int n, char * fname, int nIters, double * A);
void B1_run_gnuplot(Angles *Ang, GPData *G, int flag);

//------------------------------------------------------------------------------
//  Plot a binary angle tree
//------------------------------------------------------------------------------
//  the [n]-th node in the [m]-th level
//------------------------------------------------------------------------------
void Angles::plot_angle_tree (int m, int n)
{

  int level, leaves;
```

```
int i, j, k;


if (checkNIters("plot_angle_tree")) return;

// cout << "nIters  = " << nIters << endl;
// cout << "nAngles = " << nAngles << endl;


ofstream myout;
char fname[256];

int cond1, cond2;
int minj, maxj;
int gsize;
//----------------------------------------
if (Leaf) {
  // the [n]-th node in the [m]-th level
  // in the [m]-th level, there are 2^m nodes, so 2^m subtrees(subblocks)
  // plot leaf arrows for each of 2^m subtrees(subblocks)
  // (2^nIters) / (2^m) leaves belong to each subtree
  // to see if overlapped angle ranges between subtrees

  for (i=0; i<=1; ++i) {
    // angle1.dat, angle2.dat
    sprintf(fname, "angle%d.dat", i+1);

    // mode=0: block index, mode=1: offset index
    B1_plot_subtree_leaf(m, i, fname, nIters, A);

    GPData G(GnuTerm, nAngles);
    //----------------------------------------
    B1_run_gnuplot(this, &G, i+1);  // flag=1,2
    //----------------------------------------
  }

//----------------------------------------
} else {

  for (i=-1; i<=1; ++i) {
    // angle1.dat, angle2.dat, angle3.dat
    sprintf(fname, "angle%d.dat", i+2);

    // (n-1, n, n+1)-th subtree
    B1_plot_subtree_all(m, n+i, fname, nIters, A);

    GPData G(GnuTerm, nAngles);
    //----------------------------------------
    B1_run_gnuplot(this, &G, i+2); // flag=1,2,3
    //----------------------------------------
```

```
    }

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



  return;

}


//-------------------------------------------------------------------------------
// B1_plot_subtree_leaf
//-------------------------------------------------------------------------------
//  the n-th node in the [m]-th level
//  mode=0: height --> i: block index
//  mode=1: height --> j: offset index
//-------------------------------------------------------------------------------
void B1_plot_subtree_leaf(int m, int mode, char * fname, int nIters, double * A)
{
    int i, j, k, leaves, gsize;

    ofstream myout;

    myout.open(fname);

    // 2^m nodes (subtrees) in the [m]-th level
    gsize = 1 << m;

    for (i=0; i<gsize; ++i) {
      leaves = 1 << nIters; // no of leaves
      for (j=0; j<leaves/gsize; ++j) {

        // mode=0: height --> i: block index
        // mode=1: height --> j: offset index
        k = ((mode==0) ? i : j);

        myout << A[i*(leaves/gsize)+j]*180/pi << " ";
        myout << k << " 0.0 1.0" << endl;
      }
    }

    myout.close();
}



//-------------------------------------------------------------------------------
// B1_plot_subtree_all
```

```
//----------------------------------------------------------------------------
//  the [n]-th node in the [m]-th level
//----------------------------------------------------------------------------
void B1_plot_subtree_all(int m, int n, char * fname, int nIters, double * A)
{
    int i, j, k, level, leaves;
    int cond1, cond2, minj, maxj;

    ofstream myout;

    myout.open(fname);

    k=0;

    // i: the tree level index
    for (i=0; i<=nIters; ++i) {
      level = i;
      leaves = 1 << level;
      for (j=0; j<leaves; ++j) {
        // ancestor condition
        cond1 = (i <= m) && (j == n / (1 << (m-i))) ;

        // descendant condition
        minj = n * (1 << (i-m));
        maxj = (n+1) * (1 << (i-m));
        cond2 = (i >  m) && (minj <= j) && (j < maxj);

        if (cond1 || cond2 ) {
        // printf("[i=%d j=%d] \n", i, j);

        myout << A[k+j]*180/pi << " " <<  i << " 0.0 1.0" << endl;
        }
      }
      k += leaves;
    }

    myout.close();
}



//----------------------------------------------------------------------------
// run_gnuplot
//----------------------------------------------------------------------------
// Leaf: flag=1 : block index view
// Leaf: flag=2 : offset index view
//----------------------------------------------------------------------------
// All:  flag=1 : (m, n-1) descendants
// All:  flag=2 : (m, n-1) & (m, n) descendants
// All:  flag=3 : (m, n-1) & (m, n) & (m, n+1) descendants
```

```
//------------------------------------------------------------------------------
void B1_run_gnuplot(Angles *Ang, GPData *G, int flag)
{
  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");

  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;

  char fstr[256];

  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    sprintf(fstr, "ang_tree%d", flag);
    G->set_fname(Ang, "egb1", fstr);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    // cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }

  G->set_title(Ang, "Binary Angle Tree");
  myout << "set title '" << G->title << "' " << 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 << "set xrange [-100:100]" << endl;

  //-------------------------------------------------------
  // Leaf: flag=1 : block index view
  // Leaf: flag=2 : offset index view
  //-------------------------------------------------------
  if (Ang->getLeaf()) {
    if (flag == 1) {
      G->set_title(Ang, "Binary Angle Tree - block index view");

      myout << "plot 'angle1.dat' using 1:2:3:4    ";
      myout << "with vectors head filled lt 3  " << endl;

      if (strcmp(GnuTerm.c_str(), "wxt") == 0)
        myout << "pause mouse keypress" << endl;
    } else if (flag == 2) {
      G->set_title(Ang, "Binary Angle Tree - offset index view");
```

```
      myout << "plot 'angle2.dat' using 1:2:3:4   ";
      myout << "with vectors head filled lt 3  " << endl;

      if (strcmp(GnuTerm.c_str(), "wxt") == 0)
        myout << "pause mouse keypress" << endl;
    }

  //------------------------------------------------------------
  // All: flag=1 : (m, n-1) descendants
  // All: flag=2 : (m, n-1) & (m, n) descendants
  // All: flag=3 : (m, n-1) & (m, n) & (m, n+1) descendants
  //------------------------------------------------------------
  } else {
    if (flag == 1) {
      G->set_title(Ang, "Binary Angle Tree - consecutive subtrees 1");

      myout << "plot 'angle1.dat' using 1:2:3:4   ";
      myout << "with vectors head filled lt 3  " << endl;

      if (strcmp(GnuTerm.c_str(), "wxt") == 0)
        myout << "pause mouse keypress" << endl;
    } else if (flag == 2) {
      G->set_title(Ang, "Binary Angle Tree - consecutive subtrees 1,2");

      myout << "plot 'angle1.dat' using 1:2:3:4   ";
      myout << "with vectors head filled lt 3 ,  " ;
      myout << "     'angle2.dat' using 1:2:3:4  ";
      myout << "with vectors head filled lt 4  " << endl;

      if (strcmp(GnuTerm.c_str(), "wxt") == 0)
        myout << "pause mouse keypress" << endl;
    } else if (flag == 3) {
      G->set_title(Ang, "Binary Angle Tree - consecutive subtrees 1,2,3");

      myout << "plot 'angle1.dat' using 1:2:3:4   ";
      myout << "with vectors head filled lt 3 ,  " ;
      myout << "     'angle2.dat' using 1:2:3:4  ";
      myout << "with vectors head filled lt 4  , ";
      myout << "     'angle3.dat' using 1:2:3:4  ";
      myout << "with vectors head filled lt 5" << endl;

      if (strcmp(GnuTerm.c_str(), "wxt") == 0)
        myout << "pause mouse keypress" << endl;
    }
  //------------------------------------------------------------
  }

  myout.close();
```

```
  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  system("gnuplot command.gp");

}
```

```
::::::::::::::
Angles.1.b2.plot_circle_angle.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstring>
#include <string>

#include "Angles.hpp"
#include "GPData.hpp"

using namespace std;

//----------------------------------------------------------------------------------
//  Purpose: Class Angles Implementation Files
//
//     [Angles.1.b2.plot_circle_angle.cpp]
//
//      Angles::plot_circle_angle ()
//
//      - to plot angle vectors on the unit circle
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
```

```
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//  Outputs:
//      egb2.____.circle_ang.eps
//
//--------------------------------------------------------------------------
void B2_run_gnuplot(Angles *Ang, GPData *G, int ksize);


//--------------------------------------------------------------------------
//     Plot angle vectors on the unit circle
//--------------------------------------------------------------------------
void Angles::plot_circle_angle ()
{

  int i;


  if (checkNIters("plot_circle_angle")) return;


  ofstream myout;

  int k;
  double x0, y0, xd, yd;


  // B : sorted angles array
  vector <double> BV;

  for (int i=0; i < nAngles; ++i)  BV.push_back(A[nAngles-i-1]);
  sort(BV.begin(), BV.end());
  for (int i=0; i < nAngles; ++i) B[i] = BV[i];


  // int nPoints = getnAngles();
  // double ang  = get_min_angle();
  // double rng  = get_max_angle() - get_min_angle());
  double binnum = 256;
  double step = (B[nAngles-1] - B[0]) / nAngles;
  int    ksize = 64;


  // writing angle data on a unit circle
  myout.open("angle.dat");
```

```
  for (i=0; i<nAngles; i++) {

    k = (int) (i % ksize);
    // if (k%2 == 0) k = 2;
    // else k = 3;

    x0 = k*cos(A[i]);
    y0 = k*sin(A[i]);
    xd = cos(A[i]);
    yd = sin(A[i]);

    myout << x0 << " " << y0 << " " << xd << " " << yd << " " << endl;
  }
  myout.close();


  GPData G(GnuTerm, nAngles);
  //---------------------------------------
  B2_run_gnuplot(this, &G, ksize);
  //---------------------------------------

  return;

}


//--------------------------------------------------------------------------------
// run_gnuplot
//--------------------------------------------------------------------------------
void B2_run_gnuplot(Angles *Ang, GPData *G, int ksize)
{
  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");

  // Ang->epsList.clear();
  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    G->set_fname(Ang, "egb2", "circle_ang");
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }
```

```
  G->set_title(Ang, "Circular angle vectors by the offset in a block");
  myout << "set title '" << G->title << "' " << endl;

  myout << "set xlabel \"x\" " << endl;
  myout << "set ylabel \"y\" " << endl;
  myout << "set size square" << endl;
  myout << "set xrange [-" << ksize << ":+" << ksize << "]" << endl;
  myout << "set yrange [-" << ksize << ":+" << ksize << "]" << 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;
  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();

  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  system("gnuplot command.gp");

  return;

}



::::::::::::::
Angles.1.b3.plot_line_angle.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>
# include <cstring>

# include "Angles.hpp"
#include "GPData.hpp"

using namespace std;

//----------------------------------------------------------------------------
//  Purpose: Class Angles Implementation Files
//
//     [Angles.1.b3.plot_line_tree.cpp]
//
```

```
//      Angles::plot_line_angle ()
//
//      - to plot angle vectors on the x axis
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//  Output :
//      egb3.____.line_ang.i1.eps
//
//-----------------------------------------------------------------------------
void plot_partition(Angles *Ang);
void B3_run_gnuplot(Angles *Ang, GPData *G, XRange *Rng);



//-----------------------------------------------------------------------------
//  Plot angle vectors on the x axis
//-----------------------------------------------------------------------------
void Angles::plot_line_angle ()
{

  if (checkNIters("plot_line_angle")) return;

  // B : sorted angles array
  vector <double> BV;

  for (int i=0; i < nAngles; ++i)  BV.push_back(A[nAngles-i-1]);
  sort(BV.begin(), BV.end());
  for (int i=0; i < nAngles; ++i) B[i] = BV[i];


  // int nPoints = getnAngles();
  // double ang  = get_min_angle();
  // double rng  = get_max_angle() - get_min_angle());
  double binnum = 256;
```

```
  double step = (B[nAngles-1] - B[0]) / binnum;
  double ang = 0.0;
  double xpos;
  int    hpos;

printf("* max=%f \n", B[0]);
printf("* min=%f \n", B[nAngles-1]);
printf("* step=%f \n", step);

  ofstream myout;

  myout.open("angle.dat");

  for (int i=0; i<nAngles; ++i) {
    ang  = B[i] - B[0];
    hpos = int (ang / step);
    xpos = fmod(ang,  step);
    myout << scientific << xpos << " " << hpos << " 0.0 1.0" << endl;

    if (hpos == 0) {
    myout << scientific << xpos << " " << hpos << " 0.0 " << binnum << endl;
    }

  }

  myout.close();

  //..............................
  plot_partition(this);
  //..............................


}


//------------------------------------------------------------------------------
//  Subplot angle vectors on the x axis on the range [xmin,xmax]
//------------------------------------------------------------------------------
void plot_partition(Angles *Ang)
{
  int nPartitions = 1;

  XRange        Rng;
  GPData        G(GnuTerm, Ang->getnAngles());


  Rng.nPartitions = 1;

  if (Ang->getnIters() < 10) {
    Rng.partitionIndex = 0;
```

```
    Rng.xmin = -2;
    Rng.xmax = +2;

    //...............................................
    B3_run_gnuplot(Ang, &G, &Rng);
    //...............................................

  } else if (Ang->getnIters() < 21 ) {
    Rng.partitionIndex = 0;
    Rng.xmin = -2;
    Rng.xmax = +2;

    cout << "Enter the number of x partitions : ";
    // cin >>  nPartitions;
    cout << endl;

    nPartitions = 1;

    Rng.nPartitions = nPartitions;
    Rng.partitionIndex = 0;

    if (nPartitions > 1) {
      G.useSubRange = useXPartition;
      G.valSubRange = nPartitions;
    }

    for (int i=0; i<nPartitions; ++i) {
      Rng.xmin = -2 + 4./nPartitions *i;
      Rng.xmax = -2 + 4./nPartitions *(i+1);
      Rng.partitionIndex = i;

      //...............................................
      B3_run_gnuplot(Ang, &G, &Rng);
      //...............................................

    }

  } else {
    cout << "nIters = " << Ang->getnIters() << " is too large to plot! " << endl;
    return;
  }

  // cout << "nIters  = " << nIters << endl;
  // cout << "nAngles = " << nAngles << endl;

  return;

}
```

```
//----------------------------------------------------------------------------
//  run_gnuplot
//----------------------------------------------------------------------------
void B3_run_gnuplot(Angles *A, GPData *G, XRange *Rng)
{
  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");

  G->set_prefix(A);
  G->set_suffix(A);

  myout << "set terminal " << GnuTerm << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    char str[256];
    sprintf(str, "line_ang.i%d", Rng->partitionIndex);
    G->set_fname(A, "egb3", str);
    A->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }

  G->set_title(A, "Linear angle vectors showing jitter");
  myout << "set title '" << G->title << "' " << endl;

  myout << "set xlabel \"angles in radian\" " << endl;
  myout << "set ylabel \"\" " << endl;
  //myout << "set yrange [0:+2]" << endl;
  //myout << "set xrange [" << Rng->xmin << ":" ;
  //myout                   << Rng->xmax << "]" << endl;
  myout << "plot 'angle.dat' using 1:2:3:4  ";
  myout << "with vectors head filled lt 3" << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();

  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  system("gnuplot command.gp");

  return;
}
```

```
::::::::::::::
Angles.1.b4.plot_quantization.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstring>

#include "Angles.hpp"
#include "GPData.hpp"

using namespace std;

//----------------------------------------------------------------------------
//   Purpose: Class Angles Implementation Files
//
//     [Angles.1.b4.plot_quantization.cpp]
//
//      Angles::plot_quantization ()
//
//      - to plot quantization errors
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//      egb4.____.quantization.eps
//
//----------------------------------------------------------------------------
void B4_run_gnuplot(Angles *Ang, GPData *G);

//----------------------------------------------------------------------------
```

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


  vector <double> BV, DV;
  vector <double> ::iterator first, last;
  ofstream myout;

  cout << "* plot_quantization... ";
  if (Leaf) cout << "(LeafAngles)" << " nAngles = " << nAngles << endl;
  else      cout << "(AllAngles)"  << " nAngles = " << nAngles << endl;


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

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


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

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


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


  // write histogram data from delta array
  myout.open("angle.dat");


  for (int i=0; i<nAngles; i++) {
    myout << scientific << BV[0] + udelta*i << " ";
    myout << scientific << BV[0] + udelta*i << " ";
    myout << scientific << BV[i] << endl;
  }
  myout.close();


  GPData G(GnuTerm, nAngles);
  //----------------------------------------
  B4_run_gnuplot(this, &G);
  //----------------------------------------

  return;
```

```
}


//------------------------------------------------------------------------------
// run_gnuplot
//------------------------------------------------------------------------------
void B4_run_gnuplot(Angles *Ang, GPData *G)
{
  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");

  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    G->set_fname(Ang, "egb4", "quantization");
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }

  G->set_title(Ang, "Quantization Effect");
  myout << "set title '" << G->title << "' " << endl;


  myout << "set xlabel \"Angles \" " << endl;
  myout << "set ylabel \"Quantized Angles\" " << endl;
  // myout << "set yrange [" << BV[0] << ":" << BV[BV.size()-1] << "]" << endl;
  myout << "plot 'angle.dat' using 1:2 with lines, ";
  myout << "     'angle.dat' using 1:3 with lines" << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;


  myout.close();

  system("gnuplot command.gp");

}




::::::::::::::
```

```
Angles.2.t1.calc_tscale_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
//
//     [Angles.2.t1.calc_tscale_statistics ()
//
//      Angles::calc_tscale_statistics ()
//
//      from tree scale angles,
//      compute the sorted vector BV - min, max
//      compute the difference vector DV - min, max, avg, std
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is Distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//    min_angle, max_angle,
//    min_delta, max_delta, avg_delta, std_delta
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//   Find Angles Statistics  --> member DVata
//----------------------------------------------------------------------------
```

```
void Angles::calc_tscale_statistics ()
{


  if (checkNIters("calc_tscale_statistics")) return;

  //----------------------------------------------------------------------------
  // BV - the sorted angle vector of the angle array A
  // DV - the delta angle vector of BV
  //----------------------------------------------------------------------------
  vector <double> BV, DV;
  vector <double> ::iterator first, last;


  // BV : sorted angle vector
  for (int i=0; i < nAngles; ++i)
    BV.push_back(A[i]);

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


  // DV : difference angle vector --> delta distribution
  for (int i=0; i < nAngles-1; ++i)
    DV.push_back(BV[i+1]- BV[i]);

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


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



  // mean & std of the delta distribution
  double mean, std;

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

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

```
  set_min_angle( BV[0]            );
  set_max_angle( BV[BV.size()-1] );

  cout << "  min angle      = " << get_min_angle() << endl;
  cout << "  max angle      = " << get_max_angle() << endl;
  cout << "  -----------------" << endl;


  set_min_delta( DV[0]            );
  set_max_delta( DV[DV.size()-1] );
  set_avg_delta( mean            );
  set_std_delta( std             );

  cout << "  min delta      = " << get_min_delta() << endl;
  cout << "  max delta      = " << get_max_delta() << endl;
  cout << "  avg delta      = " << get_avg_delta() << endl;
  cout << "  std delta      = " << get_std_delta() << endl;
  cout << "  -----------------" << endl;

  double udelta = (BV[BV.size()-1] - BV[0]) /  nAngles; // computed unifrom DVelta

  cout << "  uniform delta  = " << udelta << "  = (max-min) / nAngles " << endl;


  return;
}


::::::::::::::
Angles.2.t2.plot_tscale_statistics.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstring>
#include <string>
#include <map>

#include "Angles.hpp"
#include "GPData.hpp"


using namespace std;

//----------------------------------------------------------------------------
//   Purpose: Class Angles Implementation Files
```

```
//
//      [Angles.2.t2.plot_tscale_statistics.cpp]
//
//       Angles::plot_tscale_statistics ()
//
//       plot statistics on residue angles
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//      egt2.____.delta_dist_bin??_0.[th0.001].eps  (hist with bins)
//      egt2.____.delta_dist_val??_1.[th0.001].eps  (hist with values)
//      egt2.____.delta_vs_angle.[th0.001].eps (delta vs angle)
//
//-------------------------------------------------------------------------------
// void Angles::plot_tscale_statistics (int binNum = 50)
//   void P4A_make_plot_data(Angles *Ang, int binNum, int *H)
//   void P4B_make_plot_data (Angles *Ang)
//   void P4C_make_plot_data(int binNum)
//   void P4A_run_gnuplot(int binNum, Stat & S, Angles *Ang, GPData *G)
//   void P4B_run_gnuplot (Stat & S, Angles *Ang, GPData *G)
//-------------------------------------------------------------------------------
// to pass parameters use class Stat
class Stat {
  public:
  double avg;
  double median;
  double udelta;
  double mind;
  double maxd;
};


//-------------------------------------------------------------------------------
void P4A_make_plot_data(Angles *Ang, int binNum, Stat & S, int *H);
void P4B_make_plot_data (Angles *Ang);
void P4C_make_plot_data(Angles *Ang, int binNum);
```

```
void P4A_run_gnuplot(int binNum, Stat & S, Angles *Ang, GPData *G);
void P4B_run_gnuplot (Stat & S, Angles *Ang, GPData *G);

void makeBV(Angles * Ang, vector <double> & BV);
void makeDV(Angles * Ang, vector <double> & BV, vector <double> & DV);



//------------------------------------------------------------------------------
//   Plot Delta Distribution and  Angle-Delta
//------------------------------------------------------------------------------
void Angles::plot_tscale_statistics (int binNum = 50)
{


  if (checkNIters("plot_tscale_statistics")) return;


  if (~is_tscale_stat_done())
  //..........................................
    calc_tscale_statistics();
  //..........................................


  //----------------------------------------------------------------------------
  // H  - the histogram array
  // S  - avg, median, udelta, mind, maxd;
  //----------------------------------------------------------------------------
  int *H = (int *) calloc (binNum, sizeof (int));
  Stat S;

  GPData G(GnuTerm, getnAngles());

  cout << "  + Delta distribution plot with bins \n" ;
  //..........................................
  P4A_make_plot_data(this, binNum, S, H);
  //..........................................
  P4A_run_gnuplot(binNum, S, this, &G);
  //..........................................


  cout << "  + Delta distribution plot with actual values \n" ;
  //..........................................
  P4B_make_plot_data (this);
  //..........................................
  P4A_run_gnuplot(0, S, this, &G);
  //..........................................


  cout << "  + Delta vs. angle plot \n" ;
```

```
  //.........................................
  P4C_make_plot_data(this, binNum);
  //.........................................
  P4B_run_gnuplot (S, this, &G);
  //.........................................


}


//------------------------------------------------------------------------------
// void P4A_make_plot_data (Angles *Ang, int binNum, Stat & S, int *H)
// void P4B_make_plot_data (Angles *Ang)
// void P4C_make_plot_data (Angles *Ang, int binNum)
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//   make plot data for delta distribution (histogram by a given bin size)
//------------------------------------------------------------------------------
void P4A_make_plot_data(Angles *Ang, int binNum, Stat & S, int *H)
{

  vector <double> BV;           // the sorted angle vector of the array A
  vector <double> DV;           // the delta angle vector of BV

  makeBV(Ang, BV);
  makeDV(Ang, BV, DV);


  // for a median, 0.5 should be used ***
  double frac = 0.25, findex = frac * DV.size();
  int    index = (int) findex;

  S.avg = Ang->get_avg_delta();
  S.median = DV[index];
  S.udelta = (BV[BV.size()-1] - BV[0]) / Ang->getnAngles();
  S.mind = Ang->get_min_delta();
  S.maxd = Ang->get_max_delta();

  cout << "    DV.size()/2= " << DV.size()/2;
  cout << "    median: DV[DV.size()/2]= " << DV[DV.size()/2] << endl;
  cout << "    S.median= DV[DV.size()*" << frac << "]= " << S.median << endl;


  // computed unifrom delta & bin size
  // double udelta  = (BV[BV.size()-1] - BV[0]) / Ang->getnAngles();
  double binSize = (DV[DV.size()-1] - DV[0]) / binNum;


  // compute the histogram array H
```

```
  double pb ;
  double lbound, ubound;

  for (int i=0; i< (int) DV.size(); i++)
    for (int j=0; j<binNum; ++j)  {
      lbound = DV[0] + binSize * j;
      ubound = DV[0] + binSize * (j+1);
      if ((lbound  <= DV[i]) && (DV[i] < ubound)) {
        H[j]++;
      }
    }


  //-------------------------------------------------------
  ofstream myout;

  // write histogram data from delta array
  myout.open("angle.dat");

  for (int j=0; j<binNum; j++) {
    pb = H[j] * (1. / DV.size());
    lbound = DV[0] + binSize * j;
    myout << scientific << lbound << " " ;
    myout << scientific << pb << endl;
  }

  myout.close();
  //-------------------------------------------------------

}


//--------------------------------------------------------------------------------
// typedef map<double, double> Map;
// typedef Map::iterator mI;
// typedef multimap<double, double> MMap;
// typedef MMap::iterator mmI;

//--------------------------------------------------------------------------------
//   List all the distinct delta angles (histogram for all distinct delta's)
//--------------------------------------------------------------------------------
void P4B_make_plot_data (Angles *Ang)
{

  vector <double> BV;           // the sorted angle vector of the array A

  makeBV(Ang, BV);


  MMap deltaMMap;
```

```
Map  deltaMap;

double angle, delta;
// char dStr[80];

// BV : sorted angle vector
for (int i=0; i < Ang->getnAngles(); ++i) {
  angle = BV[i];
  if (i == Ang->getnAngles()-1) delta = BV[i] - BV[i-1];
  else                          delta = BV[i+1] - BV[i];

  deltaMMap.insert(make_pair(delta, angle));
  deltaMap.insert(make_pair(delta, angle));
}

mmI it1, it2;

for (it1=deltaMMap.begin(); it1!=deltaMMap.end(); it1++)
{

  // cout << " delta =" << delta <<"  angles =" <<  angle << endl;

}

mI i1, i2;

int sum =0;
int index =0;

//-------------------------------------------------------
ofstream myout;

myout.open("angle.dat");

for (i1=deltaMap.begin(); i1!=deltaMap.end(); i1++)
{
  double delta = (*i1).first;
  // double angle = (*i1).second;
  int count = deltaMMap.count(delta);

  sum += count;
  index++;

  // cout << " d =" << delta <<"  a =" <<  angle <<  " count=" <<  count << endl;
  myout << scientific << delta  << " ";
  myout << scientific << (double) count/Ang->getnAngles() << endl;

}

myout.close();
```

```
  //-------------------------------------------------------

  cout << "    the number of distinct delta's = " << index << endl;
  cout << "    total count: " << sum << " =  nAngles:" << Ang->getnAngles() << endl;

}



//--------------------------------------------------------------------------------
//   make plot data for delta angles vs. angles (to find dense area)
//--------------------------------------------------------------------------------
void P4C_make_plot_data(Angles *Ang, int binNum)
{

  vector <double> BV;           // the sorted angle vector of the array A
  vector <double> DV;           // the delta angle vector of BV

  makeBV(Ang, BV);
  makeDV(Ang, BV, DV);


  //-------------------------------------------------------
  ofstream myout;

  // write histogram data from delta array
  myout.open("angle.dat");

  // double pb, lbound;
  // double binSize = (DV[DV.size()-1] - DV[0]) / binNum;

  for (int i=0; i < (int) BV.size()-1; i++) {
    myout << scientific << BV[i] << " ";
    myout << scientific << BV[i+1] - BV[i] << endl;
  }

  myout.close();
  //-------------------------------------------------------

}

//--------------------------------------------------------------------------------
void makeBV(Angles * Ang, vector <double> & BV)
{
  // BV : sorted angle vector
  for (int i=0; i < Ang->getnAngles(); ++i)
    BV.push_back(Ang->A[i]);
  sort(BV.begin(), BV.end());

}
```

```
//----------------------------------------------------------------------------
void makeDV(Angles * Ang, vector <double> & BV, vector <double> & DV)
{
  // DV : difference angle vector --> delta distribution
  for (int i=0; i < Ang->getnAngles()-1; ++i)
    DV.push_back(BV[i+1]- BV[i]);
  sort(DV.begin(), DV.end());
}


//----------------------------------------------------------------------------
// void P4A_run_gnuplot(int binNum, Stat & S, Angles *Ang)
// void P4B_run_gnuplot (Stat & S, Angles *Ang)
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
// Plot the histogram of delta angles
//----------------------------------------------------------------------------
// binNum = 0: using actual values (delta_dist_val)
// binNum > 0: using bins          (delta_dist_bin)
//----------------------------------------------------------------------------
void P4A_run_gnuplot(int binNum, Stat & S, Angles *Ang, GPData *G)
{

  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");


  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    char fname[80];
    if (binNum)  sprintf(fname, "delta_dist_bin");
    else         sprintf(fname, "delta_dist_val");

    G->set_fname(Ang, "egt2", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }


  if (binNum) {
     G->set_title(Ang, "TScale: Delta Angle Distribution with bins");
```

```
      G->set_xlabel("delta bins");
      G->set_ylabel("delta bins' frequency");
   } else {
      G->set_title(Ang, "TScale: Delta Angle Distribution with values");
      G->set_xlabel("actual distinct delta values");
      G->set_ylabel("delta values' frequency");
   }


   myout << "set title '" << G->title << "' " << endl;
   myout << "set xlabel \" " << G->xlabel << "\" " << endl;
   myout << "set ylabel \" " << G->ylabel << "\" " << endl;
   myout << "set yrange [0:+1]" << endl;


   //...............................
   // Some arrows
   //...............................
   char str1[80], str2[80];

   sprintf(str1, "set arrow from %g, %g to %g, %g\n", S.avg, 0.0, S.avg, 0.5);
   sprintf(str2, "set label \"avg delta \" at %g, %g right\n", S.avg, 0.5);
   myout << str1 << str2;

   sprintf(str1, "set arrow from %g, %g to %g, %g\n", S.median, 0.0, S.median, 0.7);
   sprintf(str2, "set label \"median delta \" at %g, %g right\n", S.median, 0.7);
   myout << str1 << str2;

   sprintf(str1, "set arrow from %g, %g to %g, %g\n", S.udelta, 0.0, S.udelta, 0.8);
   sprintf(str2, "set label \"uniform delta \" at %g, %g right\n", S.udelta, 0.8);
   myout << str1 << str2;
   //...............................


   myout << "plot 'angle.dat' with linespoints" << endl;

   cout << "..............................................................." << endl;
   cout << G->title << endl;
   cout << "..............................................................." << endl;

   if (strcmp(GnuTerm.c_str(), "wxt") == 0)
     myout << "pause mouse keypress" << endl;

   myout.close();


   system("gnuplot command.gp");

   return;
}
```

```
//------------------------------------------------------------------------------
//   Plot angles vs. delta angles (to find dense area)
//------------------------------------------------------------------------------
void P4B_run_gnuplot (Stat & S, Angles *Ang, GPData *G)
{

  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");



  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0)  {
    char fname[80];
    sprintf(fname, "delta_vs_angle");

    G->set_fname(Ang, "egt2", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }


  G->set_title(Ang, "TScale:Delta Angle vs. Angle ");
  G->set_xlabel("increasing angle order ");
  G->set_ylabel("delta angles(adjacent angle difference) ");

  myout << "set title '" << G->title << "' " << endl;
  myout << "set xlabel \" " << G->xlabel << "\" " << endl;
  myout << "set ylabel \" " << G->ylabel << "\" " << endl;


  //..............................
  // Some arrows
  //..............................
  char str1[80], str2[80];

  sprintf(str1, "set arrow from %g, %g to %g, %g\n", -1.0, S.avg,  +1.0, S.avg);
  sprintf(str2, "set label \"avg delta \" at %g, %g left\n", -1.5, S.avg*1.02);
  myout << str1 << str2;
```

```
  sprintf(str1, "set arrow from %g, %g to %g, %g\n", -1.0, S.udelta,  +1.0, S.udelta);
  sprintf(str2, "set label \"uniform delta \" at %g, %g right\n", +1.5, S.udelta*1.02);
  myout << str1 << str2;

  sprintf(str1, "set arrow from %g, %g to %g, %g\n", -1.0, S.median,  +1.0, S.median);
  sprintf(str2, "set label \"median delta \" at %g, %g right\n", +0.0, S.median*1.02);
  myout << str1 << str2;

  sprintf(str1, "set arrow from %g, %g to %g, %g\n", -0.7853, S.mind,  -0.7853, S.maxd);
  sprintf(str2, "set label \"-pi/4 \" at %g, %g right\n", -0.7853, S.mind);
  myout << str1 << str2;

  sprintf(str1, "set arrow from %g, %g to %g, %g\n", +0.7853, S.mind,  +0.7853, S.maxd);
  sprintf(str2, "set label \"+pi/4 \" at %g, %g right\n", +0.7853, S.mind);
  myout << str1 << str2;
  //..............................


  myout << "plot 'angle.dat' with linespoints" << endl;

  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();


  system("gnuplot command.gp");


  return;

}





::::::::::::::
Angles.2.t3.plot_tscale_residual_angles.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
```

```
#include <algorithm>
#include <cstring>

#include "Core.hpp"
#include "Angles.hpp"
#include "GPData.hpp"

using namespace std;

//------------------------------------------------------------------------------
//   Purpose: Class Angles Implementation Files
//
//      [Angles.2.t2.plot_tscale_statistics.cpp]
//
//       Angles::plot_tscale_residual_angles()
//
//       - residual angles in the reg z after cordic iterations
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//  Outputs:
//      egt3.____.res_vs_index.x1.[th0.001].eps    (residual angles vs index)
//      egt3.____.res_vs_angle.x1.[th0.001].eps    (residual angles vs angle)
//
//------------------------------------------------------------------------------
void P5_make_plot_data(double *Arr, int mode, Angles *Ang, Core *C);
void P5_run_gnuplot(double *Arr, int mode, Angles *Ang, Core *C, GPData *G);


//------------------------------------------------------------------------------
//   plot residual errors
//   Residual Angles-Angle Plot and Residual Angles-Index Plot
//------------------------------------------------------------------------------
void Angles::plot_tscale_residual_angles ()
{
```

```
  int mode;
  int num_mode = 8;


  if (checkNIters("plot_tscale_residual_angles")) return;

  // B : sorted angles array
  vector <double> BV;

  for (int i=0; i < nAngles; ++i)  BV.push_back(A[i]);
  sort(BV.begin(), BV.end());
  for (int i=0; i < nAngles; ++i) B[i] = BV[i];



  Core C;

  char path[32];
  int nBreak =0;

  C.setPath(path);
  C.setLevel(nIters);
  C.setThreshold(threshold);
  C.setNBreak(nBreak);

  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);

  GPData G(GnuTerm, getnAngles());


  cout << "  + Residual angle vs. index plot \n" ;
  //..........................................
  // Use A[i] for the residual angle vs. index plot
  //..........................................
  for (int mode=0; mode<num_mode; mode++) {
    P5_make_plot_data(A, mode, this, &C);
    P5_run_gnuplot(A, mode, this, &C, &G);
  }

  cout << "  + Residual angle vs. angle plot \n" ;
  //..........................................
  // Use B[i] for the residual angle vs. angle plot
  //..........................................
  for (int mode=0; mode<num_mode; mode++) {
    P5_make_plot_data(B, mode, this, &C);
    P5_run_gnuplot(B, mode, this, &C, &G);
  }
```

```
  return;

}



//------------------------------------------------------------------------------
// Arr == Ang->A : Use A[i] for the residual angle vs. index plot
// Arr == Ang->B : Use B[i] for the residual angle vs. angle plot
//------------------------------------------------------------------------------
void P5_make_plot_data(double *Arr, int mode, Angles *Ang, Core *C)
{
  ofstream myout;

  double x, y, z;
  double nBreak;


  // not member but local variables
  double se, ssr, mse, rms, min_err, max_err;
  se = ssr = mse = rms = 0.0;
  min_err = +1.0e+10;
  max_err = -1.0e+10;


  if (Arr == Ang->A) {
    // with increasing index values
    cout << "  + TScale: a residual angle vs. index plot" << endl;
  }
  else if (Arr == Ang->B) {
    // with increasing angle values
    cout << "  + TScale: a residual angle vs. angle plot" << endl;
  }

  int nPoints =Ang->getnAngles();
  double ang = Ang->get_min_angle();
  double step = (Ang->get_max_angle() - Ang->get_min_angle()) / nPoints;

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

  int cnt;
  int i=0;
  for (int i=0; i<Ang->getnAngles(); i++) {
    x = 1.0;
    y = 0.0;

    z = Arr[i];
```

```
    C->setNBreakInit(i);
    //............................................................
    // C->cordic(&x, &y, &z);
    C->cordic_break(&x, &y, &z, cnt);
    //............................................................
    nBreak = C->getNBreak();


    // se = z * z;
    // se = C->yy * C->yy;
    se = z * z;
    ssr += se;
    if (se > max_err) max_err = se;
    if (se < min_err) min_err = se;




      myout << fixed <<  i << " ";
      myout << scientific << Arr[i] << " " ;


      double Ecos1, Ecos2, Esin1, Esin2;  int cnt;
      Ecos2 = x - cos(Arr[i] - z);  Esin2 = y - sin(Arr[i] - z);
      Ecos1 = C->xx - Ecos2;        Esin1 = C->yy - Esin2;

      switch (mode) {
        case 0: myout << scientific << z << endl;                        break;
        case 1: myout << scientific << Arr[i] - z << endl;               break;
        case 2: myout << scientific << C->xx << endl;                    break;
        case 3: myout << scientific << C->yy << endl;                    break;
        case 4: myout << scientific << x - cos(Arr[i] - z) << endl;      break;
        case 5: myout << scientific << y - sin(Arr[i] - z) << endl;      break;
        case 6: myout << scientific << Ecos2 / C->xx *100  << endl;      break;
        case 7: myout << scientific << Esin2 / C->yy *100  << endl;      break;
        default: myout << scientific << z << endl;                       break;
     }


  }

  myout.close();


  mse = ssr / Ang->getnAngles();
  rms = sqrt(mse);

  // max_err = sqrt(max_err);
```

```
  cout << "  No of points = " << Ang->getnAngles() ;
  cout << " (nBreak = " << nBreak << " : " ;
  cout <<  100. * nBreak / Ang->getnAngles() << " % )" << endl;

  printf("  SSR: Sum of Squared Residual Angles    = ") ;
  printf("%12.7f (= %g) \n", ssr, ssr);
  printf("  MSR: Mean Squared Residual Angles      = ") ;
  printf("%12.7f (= %g) \n", mse, mse);
  printf("  RMS: Root Mean Squared Residual Angles = ") ;
  printf("%12.7f (= %g) \n", rms, rms);
  printf("  Min Squared Residual Angle Error       = ") ;
  printf("%12.7f (= %g) \n", min_err, min_err);
  printf("  Max Squared Residual Angle Error       = ") ;
  printf("%12.7f (= %g) \n", max_err, max_err);

  // cout << fixed << right << setw(12) << setprecision(7) << ssr << endl;
  // cout << fixed << right << setw(12) << setprecision(7) << mse << endl;
  // cout << fixed << right << setw(12) << setprecision(7) << rms << endl;
  // cout << fixed << right << setw(12) << setprecision(7) << max_err << endl;


}


//-----------------------------------------------------------------------------
// Arr == Ang->A : Use A[i] for Index vs Residual Angles angles Plot
// Arr == Ang->B : Use B[i] for Angle vs Residual Angles angles Plot
//-----------------------------------------------------------------------------
void P5_run_gnuplot(double *Arr, int mode, Angles *Ang, Core *C, GPData *G)
{
  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");


  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    char fname[80];
    if (Arr == Ang->A)  sprintf(fname, "res%d_vs_index", mode);
    else                sprintf(fname, "res%d_vs_angle", mode);

    G->set_fname(Ang, "egt3", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
```

```
}


 char tstr[80];
 char istr[80];

 if (Arr == Ang->A)  sprintf(istr, "Index (mode%d)", mode);
 else                sprintf(istr, "Angle (mode%d)", mode);

 switch (mode) {
   case 0: sprintf(tstr, "TScale: A Residual Angle vs. %s", istr);   break;
   case 1: sprintf(tstr, "TScale: A Resolved Angle vs. %s", istr);   break;
   case 2: sprintf(tstr, "TScale: Full Cos Error vs. %s", istr);     break;
   case 3: sprintf(tstr, "TScale: Full Sin Error vs. %s", istr);     break;
   case 4: sprintf(tstr, "TScale: Resolved Cos Error vs. %s", istr); break;
   case 5: sprintf(tstr, "TScale: Resolved Sin Error vs. %s", istr); break;
   case 6: sprintf(tstr, "TScale: Norm. Resolved Cos Error vs. %s", istr); break;
   case 7: sprintf(tstr, "TScale: Norm. Resolved Sin Error vs. %s", istr); break;
   default: sprintf(tstr, "TScale: A Residual Angle vs. %s", istr);  break;
}

char ustring[80];
if (Arr == Ang->A) {
   G->set_title(Ang, tstr);
   G->set_xlabel("increasing index values");
   sprintf(ustring, "%s", "1:3");
} else {
   G->set_title(Ang, tstr);
   G->set_xlabel( "increasing angle values");
   sprintf(ustring, "%s", "2:3");
}



myout << "set title '" << G->title << "' " << endl;
myout << "set xlabel \" " << G->xlabel << "\" " << endl;
myout << "set ylabel \"residual angles in the z reg\" " << endl;


myout << "plot 'angle.dat' using " << ustring << " with linespoints " << endl;

cout << "..............................................................." << endl;
cout << G->title << endl;
cout << "..............................................................." << endl;

if (strcmp(GnuTerm.c_str(), "wxt") == 0)
  myout << "pause mouse keypress" << endl;

myout.close();
```

```
  system("gnuplot command.gp");

}




::::::::::::::
Angles.3.u1.calc_uscale_statistics.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>
# include <map>

# include "Angles.hpp"
# include "Core.hpp"

using namespace std;

//----------------------------------------------------------------------------
//   Purpose: Class Angles Implementation Files
//
//      [Angles.3.u1.calc_uscale_statistics.cpp]
//
//       Angles::calc_uscale_statistics ()
//
//       - uniform scale statistics
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
```

```
//----------------------------------------------------------------------------
void find_residual_angles(int nPoints, Angles *Ang, Core *C, uStat & S);
void calc_statistics(int nPoints, uStat & S);
void make_histogram(int nPoints, MMap & A, Map & C, const char * inStr);
double find_min(Map & H);
double find_max(Map & H);
double find_avg(Map & H);
double find_std(Map & H, double avg);
void print_map(Map & H);

//----------------------------------------------------------------------------
// calculate uniform scale statistics
//----------------------------------------------------------------------------
//  Map  ARm;   // Map       : angle - residual
//  Map  ADm;   // Map       : angle - difference (of adjacent residuals)
//  MMap RAmm;  // multiMap  : residual   - angle
//  MMap DAmm;  // multiMap  : difference - angle
//  Map  HRCm;  // Map       : residual   - angle for a histogram
//  Map  HDCm;  // Map       : difference  -angle for a histogram
//----------------------------------------------------------------------------
void Angles::calc_uscale_statistics (int nPoints =10000)
{

  // int sampling;

  if (checkNIters("calc_uscale_statistics")) return;

  if (nPoints < 0) {
    cout << "Overflow in nPoints=" << nPoints << endl;
    return;
  }


  if (~is_tscale_stat_done()) {
    cout << "........................................." << endl;
    calc_tscale_statistics();
    cout << "........................................." << endl;
  }


  Core C;

  char path[32] = "";
  int nBreak =0;

  C.setPath(path);
  C.setLevel(nIters);
  C.setThreshold(threshold);
  C.setNBreak(nBreak);
```

```
  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);


  setnPoints(nPoints);


  //...............................................................
  find_residual_angles(nPoints, this, &C, S);
  //...............................................................


  //...............................................................
  calc_statistics(nPoints, S);
  //...............................................................


  //...............................................................
  make_histogram(nPoints, S.RAmm, S.HRCm, "residual");
  make_histogram(nPoints, S.DAmm, S.HDCm, "difference of residual");
  //...............................................................


  return;

}

//----------------------------------------------------------------------------
// Find residual angles on a 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
//   min_err : minimum of squared errors
//----------------------------------------------------------------------------
void find_residual_angles(int nPoints, Angles *Ang, Core *C, uStat & S)
{
  double x, y, z;
  double nBreak;

  // not member but local variables
  double se, ssr, mse, rms, min_err, max_err;
  se = ssr = mse = rms = 0.0;
  min_err = +1.0e+100;
  max_err = -1.0e-100;


  double ang = Ang->get_min_angle();
```

```
double step = (Ang->get_max_angle() - Ang->get_min_angle()) / nPoints;
int    n = 0;
double old_z = 0., diff = 0.;

S.ARm.clear();
S.ADm.clear();
S.RAmm.clear();
S.DAmm.clear();
S.HRCm.clear();
S.HDCm.clear();

printf("  nPoints = %d init ang = %g step = %g \n", nPoints, ang, step);


int cnt=0;
while (ang < Ang->get_max_angle()) {
  x = 1.0;
  y = 0.0;
  z = ang;


  C->setNBreakInit(n);
  //............................................................
  // C->cordic(&x, &y, &z);
  C->cordic_break(&x, &y, &z, cnt);
  //............................................................
  nBreak = C->getNBreak();

  // se = z * z;
  se = C->xx * C->xx;
  ssr += se;
  if (se > max_err) max_err = se;
  if (se < min_err) min_err = se;


  diff = z - old_z;

  S.R.push_back(z);   // raw residue value

  S.ARm.insert   ( make_pair (ang,  se)   );
  S.RAmm.insert  ( make_pair (se,   ang)  );
  S.HRCm.insert  ( make_pair (se,   ang)  );  // overwrite

  S.ADm.insert   ( make_pair (ang,  diff) );
  S.DAmm.insert  ( make_pair (diff, ang)  );
  S.HDCm.insert  ( make_pair (diff, ang)  );  // overwrite

  // HRCm, HDCm stores the latest item --> to find unique res & diff
  // in make_histogram(), frequency count is stored in second field
```

```
    old_z = z;
    ang += step;
    n++;
  }


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

  printf("  No of points = %d \n", n);
  printf("  (nBreak = %d : %g %% )\n", (int) nBreak, (100.*nBreak)/n);

  printf("  SSR: Sum of    Squared Residual Angles (Sum z*z)    \n") ;
  printf("  MSR: Mean      Squared Residual Angles (SSR/nPoints) \n") ;
  printf("  RMS: Root Mean Squared Residual Angles (sqrt(MSR))  \n") ;
  printf("  Min            Squared Residual Angles (Min z*z)    \n") ;
  printf("  Max            Squared Residual Angles (Max z*z)    \n") ;

  printf("  SSR: (Sum z*z)     = %15.9f (= %g) \n", ssr, ssr) ;
  printf("  MSR: (SSR/nPoints) = %15.9f (= %g) \n", mse, mse) ;
  printf(" #RMS: (sqrt(MSR))  = %15.9f (= %g)#\n", rms, rms) ;
  printf("  Min  (Min z*z)     = %15.9f (= %g) \n", min_err, min_err) ;
  printf("  Max  (Max z*z)     = %15.9f (= %g) \n", max_err, max_err) ;

}


//----------------------------------------------------------------------------
//   Calculate statistics
//----------------------------------------------------------------------------
//   min, max angle
//   min, max, avg, std, rms residuals
//   min, max, avg, std, rms difference residuals
//----------------------------------------------------------------------------
void calc_statistics(int nPoints, uStat & S) {
  // double mean, std;
  // double diff, res, ang;
  // double step_ang, rms_res;
  // int count = 0;

  mI si, ei, i1;


  S.min_ang = find_min(S.ARm);
  S.max_ang = find_max(S.ARm);

  S.step_ang = (S.max_ang-S.min_ang)/nPoints;

  printf("--------------------------------\n");
  printf("  min angle        = %g \n", S.min_ang);
```

```
  printf("  max angle        = %g \n", S.max_ang);
  printf(" #step angle       = %g   ", S.step_ang);
  printf("= (max_angle-min_angle) / nPoints \n");


  //---------------------------------------------------------
  S.min_res = find_min(S.HRCm);
  S.max_res = find_max(S.HRCm);

  S.avg_res = find_avg(S.HRCm);
  S.std_res = find_std(S.HRCm, S.avg_res);

  S.rms_res = sqrt(S.avg_res);


  printf("---------------------------------\n");
  printf("  min     residual = %g (sqrt: %g) \n", S.min_res, sqrt(S.min_res));
  printf("  max     residual = %g (sqrt: %g) \n", S.max_res, sqrt(S.max_res));
  printf("  avg     residual = %g (sqrt: %g) \n", S.avg_res, sqrt(S.avg_res));
  printf("  std     residual = %g (sqrt: %g) \n", S.std_res, sqrt(S.std_res));

  // print_map(S.HRCm);



  //---------------------------------------------------------
  S.min_diff = find_min(S.HDCm);
  S.max_diff = find_max(S.HDCm);

  S.avg_diff = find_avg(S.HDCm);
  S.std_diff = find_std(S.HDCm, S.avg_diff);

  printf("---------------------------------\n");
  printf("  min     diff     = %g \n", S.min_diff);
  printf("  max     diff     = %g \n", S.max_diff);
  printf("  avg     diff     = %g \n", S.avg_diff);
  printf("  std     diff     = %g \n", S.std_diff);
  printf("---------------------------------\n");

}


//--------------------------------------------------------------------------------
// make_histogram(nPoints, S.RAmm, S.HRCm, "residual");
// make_histogram(nPoints, S.DAmm, S.HDCm, "difference of residual");
//--------------------------------------------------------------------------------
void make_histogram(int nPoints, MMap & A, Map & C, const char * inStr)
{
  double tmp;
  int sum, cnt;
```

```
  int index = 0;

  sum = 0.0;

  mI i1;

  for (i1=C.begin(); i1!=C.end(); i1++)
  {

    tmp = (*i1).first;
    cnt = A.count(tmp);
    (*i1).second = cnt;
    sum += cnt;
    index++;
// cout << "1st= " << (*i1).first << " ";
// cout << "2nd= " << (*i1).second << " ";
// cout << "     " << inStr << endl;

  }

  cout << "    the number of distinct " << inStr << " angles = " << index << endl;
  cout << "    total count: " << sum << " =  nPoints:" << nPoints << endl;

}





//-------------------------------------------------------------------------------
double find_min(Map & H)
{
  mI si = H.begin();
  return ((*si).first); // minimum of a range (res or diff)
}

//-------------------------------------------------------------------------------
double find_max(Map & H)
{

  mI ei = H.end();

  ei--;
  return ((*ei).first); // maximum of a range (res or diff)
}

//-------------------------------------------------------------------------------
double find_avg(Map & H)
{
  mI i1;
```

```
  double avg=0.0;
  int count = 0;
  for (i1=H.begin(); i1!=H.end(); i1++)
  {
    double tmp = (*i1).first;
    avg += tmp;
    count++;
  }
  avg /= count;  // average of a range (res or diff)
  return (avg);
}

//----------------------------------------------------------------------------
double find_std(Map & H, double avg)
{
  mI i1;

  double std=0.0;
  int count = 0;
  for (i1=H.begin(); i1!=H.end(); i1++)
  {
    double tmp = (*i1).first;
    std += ((tmp - avg) * (tmp - avg));
    count++;
  }
  std /= count;  // std dev of a range (res or diff)
  return (std);
}



//----------------------------------------------------------------------------
void print_map(Map & H)
{
  mI lb = H.begin();
  mI ub = H.end();
  mI i;
  int n=0;

  for (i=lb; i!=ub; i++) {
    printf("n=%d first=%g \n", n, (*i).first);
    n++;
  }

}
```

```
::::::::::::::
Angles.3.u2.plot_uscale_statistics.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>
# include <cstring>
# include <string>

# include "Angles.hpp"
# include "GPData.hpp"

using namespace std;

int prec = 2;

//--------------------------------------------------------------------------------
//   Purpose: Class Angles Implementation Files
//
//      [Angles.3.u2.plot_uscale_statistics]
//
//       Angles::plot_uscale_statistics ()
//
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//     egu2.____.dff_hist.n%d.x%d.--%f.ext
//     egu2.____.res_hist.n%d.x%d.--%f.ext
//     egu2.____.angle_vs_dff.n%d.x%d.--%f.ext
//     egu2.____.angle_vs_res.n%d.x%d.--%f.ext
//
```

```
//----------------------------------------------------------------------------
void P7A_make_plot_data_mI(uStat & S, int D_RB);
void P7B_make_plot_data_mmI(uStat & S, int D_RB);
void P7A_run_gnuplot(Angles *Ang, uStat& S, GPData *G, int D_RB);
void P7B_run_gnuplot(Angles *Ang, uStat& S, GPData *G, int D_RB);
void print_top5(Map & H, int sum, int D_RB);
void markArrows(char * str, int D_RB, uStat& S, int R_SB);


//----------------------------------------------------------------------------
//   Plot uniform scale statistics
//----------------------------------------------------------------------------
//   nPoints : the number of angle points on the uniform scale
//   uStat :
//      Map  ARm;   // map       : angle - residual
//      Map  ADm;   // map       : angle - difference (of adjacent residuals)
//      MMap RAmm;  // multimap  : residual   - angle
//      MMap DAmm;  // multimap  : difference - angle
//      Map  HRCm;  // map       : residual   - count for a histogram
//      Map  HDCm;  // map       : difference  -count for a histogram
//      double min_ang,  max_ang;
//      double min_res,  max_res,  avg_res,  std_res;
//      double min_diff, max_diff, avg_diff, std_diff;
//----------------------------------------------------------------------------
void Angles::plot_uscale_statistics (int nPoints)
{

  if (~is_tscale_stat_done()) {
    cout << "........................................." << endl;
    calc_tscale_statistics();
    cout << "........................................." << endl;
  }


  if (checkNIters("plot_uscale_statistics")) return;

  // D_RB=0 : RAmm, HRCm - residue
  // D_RB=1 : DAmm, HDCm - difference residue
  int D_RB;

  GPData G(GnuTerm, getnAngles());


  //.........................................
  cout << "  + Residue - Histogram Plot \n" ;
  P7A_make_plot_data_mI(S, D_RB=0);
  P7A_run_gnuplot(this, S, &G, D_RB=0);
  //.........................................
  cout << "  + Difference Residue - Histogram Plot \n" ;
  P7A_make_plot_data_mI(S, D_RB=1);
```

```
  P7A_run_gnuplot(this, S, &G, D_RB=1);
  //.........................................

   //.........................................
  cout << "  + Angles - Residue Plot \n" ;
  P7B_make_plot_data_mmI(S, D_RB=0);
  P7B_run_gnuplot(this, S, &G, D_RB=0);
  //.........................................
  cout << "  + Angles - Difference Residue Plot \n" ;
  P7B_make_plot_data_mmI(S, D_RB=1);
  P7B_run_gnuplot(this, S, &G, D_RB=1);
  //.........................................


  return;

}


//----------------------------------------------------------------------------
//   make plot data for residue or difference of residue (histogram)
//----------------------------------------------------------------------------
//   D_RB = 0 : HRCm (Residue - Count)
//   D_RB = 1 : HDCm (Difference - Count)
//----------------------------------------------------------------------------
void P7A_make_plot_data_mI(uStat & S, int D_RB)
{
  mI lbound, ubound;

  if (D_RB) {
    lbound = S.HDCm.begin();
    ubound = S.HDCm.end();
    cout << "     . [difference residue - frequency] plot using HDCm " << endl;
  } else {
    lbound = S.HRCm.begin();
    ubound = S.HRCm.end();
    cout << "     . [residue - frequency] plot using HRCm " << endl;
  }


  ofstream myout;

  // write histogram data from delta array
  myout.open("angle.dat");

  map<double, double> C;
  map<double, double>::iterator i;

  mI i1;
  char str[80];
```

```
  double tmp1, tmp2, tmp3, tmp4, sum, maxCount;

  sum = 0.0;
  maxCount = 0.0;
  for (i1=lbound; i1!=ubound; i1++) {
    tmp1  = (*i1).first;    // residue or difference residue
    tmp2  = (*i1).second;   // count
    sum += tmp2;

    // reducing effective numbers -- like a round off
    if (D_RB) {
        sprintf(str, "%20.10f", tmp1);   // rounded difference residue
    } else {
        int method = 1;
        if (method) {
          sprintf(str, "%20.9f", tmp1);   // rounded residue
          // printf(str, "%20.9f", tmp1);   // rounded residue
        } else {
          sprintf(str, "%20.2e", tmp1);   // rounded residue
          // printf(str, "%20.2e", tmp1);   // rounded residue
        }
    }

    if (C[atof(str)] ==  0.0) {
      C[atof(str)] = tmp2;              // new count
    } else {
      tmp3 = tmp2 + C[atof(str)];       // add the second comp
      C[atof(str)] = tmp3;              // to the existing count
    }

    if (maxCount < C[atof(str)]) maxCount = C[atof(str)];

  }

  print_top5(C, sum, D_RB);

cout << "total count sum = " << sum << endl;

  // for cumulative relative frequency plot
  double max_freq = 0;
  tmp3 = 0.0;
  for (i=C.begin(); i!=C.end(); i++) {

    if (D_RB) tmp1 = (*i).first;
    else tmp1 = sqrt((*i).first);   // residue or difference residue
    tmp2 = (*i).second / sum;       // relative frequency
    tmp3 = tmp3 + tmp2;
    tmp4 = tmp3 * maxCount/sum;     // normalized cumulative frequency

    sprintf(str, "%g %g %g", tmp1, tmp2, tmp4);
```

```
    myout << str << endl;

    if (max_freq < tmp2) max_freq = tmp2;

  }

  myout.close();

  if (D_RB) S.max_freq_diff = max_freq;
  else      S.max_freq_res  = max_freq;


}


//----------------------------------------------------------------------------
//   make plot data for residue or difference vs. angles
//----------------------------------------------------------------------------
//   D_RB = 0 : RAmm (Residue - Angle)
//   D_RB = 1 : DAmm (Difference - Angle)
//----------------------------------------------------------------------------
void P7B_make_plot_data_mmI(uStat & S, int D_RB)
{

  mmI lbound, ubound;

  if (D_RB) {
    lbound = S.DAmm.begin();
    ubound = S.DAmm.end();
    cout << "    . [angle - difference residue] plot using HDCm " << endl;
  } else {
    lbound = S.RAmm.begin();
    ubound = S.RAmm.end();
    cout << "    . [angle - residue] plot using HRCm " << endl;
  }



  ofstream myout;

  // write histogram data from delta array
  myout.open("angle.dat");

  mmI i1;

  multimap<double, double> C;
  multimap<double, double>::iterator i;

  char str1[80], str2[80];
  double tmp1, tmp2;
```

```
  int n;
  for (i1=lbound; i1!=ubound; i1++) {
    tmp1  = (*i1).first;     // residue or difference residue
    tmp2  = (*i1).second;    // angle
    n++;
    // printf("n=%d res = %g angle = %g \n", n, tmp1, tmp2);


#if 0
    n++;
    do {
      tmp3 = (*i1).first;
      i1++;
      if (i1 == ubound) break;
      n++;
      printf("n=%i, tmp3 - tmp1 =%g step_ang=%g\n", n, sqrt(tmp3)-sqrt(tmp1), S.step_ang);
    } while ((sqrt(tmp3) - sqrt(tmp1)) < S.step_ang);

    if (i1 == ubound) break;
#endif


    //----------------------------------------------------------------
    //  sprintf(str1, "%20.6f", tmp1);   -- reticle -- step angle ?
    //  sprintf(str2, "%20.2f", tmp2);   -- reticle -- period?
    //----------------------------------------------------------------
    // reducing effective numbers -- like a round off
    if (D_RB) {
        sprintf(str1, "%20.7f", tmp1);   // rounded difference residue
        sprintf(str2, "%20.3f", tmp2);   // rounded difference residue
    } else {
        sprintf(str1, "%20.7f", sqrt(tmp1));   // rounded residue
        sprintf(str2, "%20.3f", tmp2);   // rounded residue
    }

    C.insert( make_pair(atof(str1), atof(str2))  );

// cout << "first = " << str << " second = " << tmp2 << endl;
  }



  for (i=C.begin(); i!=C.end(); i++) {
    tmp1 = (*i).first;           // residue or difference residue
    tmp2 = (*i).second;          // angles
    sprintf(str1, "%g %g", tmp1, tmp2);
    myout << str1 << endl;
  }
```

```
  myout.close();

}



//-----------------------------------------------------------------------------
//   Plot the histogram of residue or differece
//-----------------------------------------------------------------------------
//   D_RB = 0 : Residue Histogram
//   D_RB = 1 : Difference Histogram
//-----------------------------------------------------------------------------
void P7A_run_gnuplot(Angles *Ang, uStat& S, GPData *G, int D_RB)
{

  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");

  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    char fname[80];
    if (D_RB) sprintf(fname, "dff_hist");
    else        sprintf(fname, "res_hist");

    G->set_fname(Ang, "egu2", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }


  if (D_RB) {
     G->set_title(Ang, "UScale: Frequency vs. Difference Residue");
     G->set_xlabel("difference residue in the increasing order");
     G->set_ylabel("difference residue frequency");
  } else {
     G->set_title(Ang, "UScale: Frequency vs. Residue");
     G->set_xlabel("residue (sqrt(z*z))");
     G->set_ylabel("residue frequency");
  }


  myout << "set title '" << G->title << "' " << endl;
```

```
  myout << "set xlabel \" " << G->xlabel << "\" " << endl;
  myout << "set ylabel \" " << G->ylabel << "\" " << endl;


  char str[256];
  int R_SB;

  markArrows(str, D_RB, S, R_SB=0);  // rms res label
  myout << str << endl;

  markArrows(str, D_RB, S, R_SB=1);  // step angle label
  myout << str << endl;


  myout << "plot 'angle.dat' using " << "1:2" << " with impulses ";
  myout << ",    'angle.dat' using " << "1:3" << " with lines " << endl;

  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();

  system("gnuplot command.gp");

}


//------------------------------------------------------------------------------
//   Plot angles vs residue or differece s
//------------------------------------------------------------------------------
//   D_RB = 0 : Angles vs. Residue
//   D_RB = 1 : Angles vs. Difference
//------------------------------------------------------------------------------
void P7B_run_gnuplot(Angles *Ang, uStat& S, GPData *G, int D_RB)
{

  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");

  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
```

```
    char fname[80];
    if (D_RB) sprintf(fname, "angle_vs_dff");
    else        sprintf(fname, "angle_vs_res");

    G->set_fname(Ang, "egu2", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }


  char title[80];
  char xlabel[80];
  char ylabel[80];
  if (D_RB) {
     sprintf(title, "%s", "UScale: Angles vs. Difference Residue ");
     sprintf(xlabel, "%s", "angles in the increasing order");
     sprintf(ylabel, "%s", "difference residue");
  } else {
     sprintf(title, "%s", "UScale: Angles vs. Residue");
     sprintf(xlabel, "%s", "angles in the increasing order");
     sprintf(ylabel, "%s", "residue (sqrt(z*z))");
  }
  G->set_title(Ang, title);

  myout << "set title '" << G->title << "' " << endl;
  myout << "set xlabel \" " << xlabel << "\" " << endl;
  myout << "set ylabel \" " << ylabel << "\" " << endl;


  myout << "plot ";
  // myout << " 'angle.dat' using " << G.ustring << " with impulses linetype 1, ";
  myout << " 'angle.dat' using " << "1:2" << " with points linetype 1" << endl;

  //myout << "set style data histograms" << endl;
  //myout << "set style histogram cluster" << endl;
  //myout << "set style fill solid 1.0 border lt -1" << endl;
  //myout << plot for [COL=2:4:2] 'file.dat' using COL

  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();

  system("gnuplot command.gp");
```

```
}



//----------------------------------------------------------------------------
//  Print the most frequent top 5 (used in  P7A_run_gnuplot)
//----------------------------------------------------------------------------
//   D_RB = 0 : Residue
//   D_RB = 1 : Difference
//----------------------------------------------------------------------------
void print_top5(Map & H, int sum, int D_RB)
{

  mI i1;
  map<double, double> C;

  double tmp1, tmp2;

  for (i1=H.begin(); i1!=H.end(); i1++) {
    C[(*i1).second] = (*i1).first;
  }

  printf("     top 5 list \n");

  i1 = C.end();
  for (int s=0; s<5; s++) {
    --i1;
    if (D_RB) {
      tmp1 = ((*i1).first)/sum;
      tmp2 = ((*i1).second);
    } else {
      tmp1 = ((*i1).first)/sum;
      tmp2 = sqrt((*i1).second);
    }

    printf("       rel freq: %g  residue: %g \n", tmp1, tmp2);
  }

}



//----------------------------------------------------------------------------
//  mark arrows with labels (used in  P7A_run_gnuplot)
//----------------------------------------------------------------------------
//  R_SB=0: step angle
//  R_SB=1: rms value of residual angles
//----------------------------------------------------------------------------
void markArrows(char * str, int D_RB, uStat& S, int R_SB)
```

```
{
  char label[256];
  char tmp[256];
  double x1, x2, y1, y2, dx, dy;

  if (D_RB) {
    dx = (S.max_diff - S.min_diff)*0.1;
    dy = S.max_freq_diff * 0.2;

  } else {
    dx = (sqrt(S.max_res) - sqrt(S.min_res))*0.1;
    dy = S.max_freq_res * 0.2;
  }


  if (R_SB) {
    sprintf(label, "rms res");
    x2 = S.rms_res;
    y2 = 0;
    x1 = x2 + dx;
    y1 = y2 + 2*dy;
  } else {
    sprintf(label, "step angle");
    x2 = S.step_ang;
    y2 = 0;
    x1 = x2 + dx;
    y1 = y2 + dy;
  }


  sprintf(str, "set arrow from %g, %g ", x1, y1);
  sprintf(tmp, "to %g, %g \n", x2, y2);
  strcat(str, tmp);

  sprintf(tmp, "set label '%s' at %g, %g left ", label, x1, y1);
  strcat(str, tmp);


}
```

```
::::::::::::::
Angles.3.u3.plot_uscale_residual_angles.cpp
::::::::::::::
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cstring>

#include "Core.hpp"
#include "Angles.hpp"
#include "GPData.hpp"

using namespace std;


// #define RND


//----------------------------------------------------------------------------
//   Purpose: Class Angles Implementation Files
//
//      [Angles.3.u3.plot_uscale_residual_angles.cpp]
//
//       Angles::plot_uscale_residual_angles()
//
//       - residual angles in the reg z after cordic iterations
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.02.07
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
```

```
//  Outputs:
//      egu3.____.res_vs_index.x1.[th0.001].eps    (residual angles vs index)
//      egu3.____.res_vs_angle.x1.[th0.001].eps    (residual angles vs angle)
//
//-----------------------------------------------------------------------------
void P9_make_plot_data(double *Arr, int mode, int rnd, Angles *Ang, Core *C);
void P9_run_gnuplot(double *Arr, int mode, int rnd, Angles *Ang, Core *C, GPData *G);


//-----------------------------------------------------------------------------
//   plot residual errors
//   Residual Angles-Angle Plot and Residual Angles-Index Plot
//-----------------------------------------------------------------------------
void Angles::plot_uscale_residual_angles (int rnd)
{

  int mode;
  int num_mode = 8;


  if (checkNIters("plot_uscale_residual_angles")) return;


  if (rnd)
    cout << "Random Mode : ON" << endl;
  else
    cout << "Random Mode : OFF" << endl;

/*
  if (rnd)
    setnAngles(getnAngles()*6);
*/


  Core C;

  char path[32];
  int nBreak =0;

  C.setPath(path);
  C.setLevel(nIters);
  C.setThreshold(threshold);
  C.setNBreak(nBreak);

  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);

  GPData G(GnuTerm, getnAngles());
```

```
if (1) {
  cout << "  + Residual angle vs. index plot [[random angles]] \n" ;
  //..........................................
  // Use A[i] for the residual angle vs. index plot
  //..........................................
  for (int mode=0; mode<num_mode; mode++) {
    P9_make_plot_data(A, mode, rnd, this, &C);
    P9_run_gnuplot(A, mode, rnd, this, &C, &G);
  }
}


if (1) {

  // B : sorted angles array
  vector <double> BV;

  for (int i=0; i < nAngles; ++i)  BV.push_back(A[nAngles-i-1]);
  sort(BV.begin(), BV.end());
  for (int i=0; i < nAngles; ++i) B[i] = BV[i];


  cout << "  + Residual angle vs. angle plot  \n" ;
  //..........................................
  // Use B[i] for the residual angle vs. angle plot
  //..........................................
  for (int mode=0; mode<num_mode; mode++) {
    P9_make_plot_data(B, mode, rnd, this, &C);
    P9_run_gnuplot(B, mode, rnd, this, &C, &G);
  }

  BV.clear();

}

  return;

}



//--------------------------------------------------------------------------------
// Arr == Ang->A : Use A[i] for the residual angle vs. index plot
// Arr == Ang->B : Use B[i] for the residual angle vs. angle plot
//--------------------------------------------------------------------------------
void P9_make_plot_data(double *Arr, int mode, int rnd, Angles *Ang, Core *C)
{
  ofstream myout;

  double x, y, z;
```

```
  double nBreak;


  // not member but local variables
  double se, ssr, mse, rms, min_err, max_err;
  se = ssr = mse = rms = 0.0;
  min_err = +1.0e+10;
  max_err = -1.0e+10;


  if (Arr == Ang->A) {
    // with increasing index values
    cout << "  + uscale: a residual angle vs. an index plot" << endl;
  }
  else if (Arr == Ang->B) {
    // with increasing angle values
    cout << "  + uscale: a residual angle vs. an angle plot" << endl;
  }

  int nPoints =Ang->getnAngles();
  double ang = Ang->get_min_angle();
  double rng = (Ang->get_max_angle() - Ang->get_min_angle());
  double step = (Ang->get_max_angle() - Ang->get_min_angle()) / nPoints;



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

  int cnt;
  int i=0;

/*
#ifdef RND
  while (ang < Ang->get_max_angle()) {
#else
  for (int i=0; i<Ang->getnAngles(); i++) {
#endif
*/

  for (int i=0; i<Ang->getnAngles(); i++) {

    x = 1.0;
    y = 0.0;

/*
      if (rnd) {
        Arr[i] = ((double) rand() / (RAND_MAX) - 0.5) * rng;
      } else {
        Arr[i] = ang;
```

```
        ang += step;
      }
*/

    if (Arr == Ang->A) {

      if (rnd) {
        Arr[i] = ((double) rand() / (RAND_MAX) - 0.5) * rng;
      } else {
        Arr[i] = ang;
        ang += step;
      }

    }
    else {
        // Arr[i]=ang;
    }


    z = Arr[i];

    C->setNBreakInit(i);
    //...........................................................
    // C->cordic(&x, &y, &z);
    C->cordic_break(&x, &y, &z, cnt);
    //...........................................................
    nBreak = C->getNBreak();



    // se = z * z;
    // se = C->yy * C->yy;
    se = z * z;
    ssr += se;
    if (se > max_err) max_err = se;
    if (se < min_err) min_err = se;




    myout << fixed <<  i << " ";
    myout << scientific << Arr[i] << " " ;

    double Ecos1, Ecos2, Esin1, Esin2;
    Ecos2 = x - cos(Arr[i] - z);  Esin2 = y - sin(Arr[i] - z);
    Ecos1 = C->xx - Ecos2;        Esin1 = C->yy - Esin2;

    switch (mode) {
      case 0: myout << scientific << z << endl;                 break;
      case 1: myout << scientific << Arr[i] - z << endl;                break;
```

```
      case 2: myout << scientific << x - cos(Arr[i]) << endl;           break;
      case 3: myout << scientific << y - sin(Arr[i]) << endl;           break;
      case 4: myout << scientific << x - cos(Arr[i] - z) << endl;       break;
      case 5: myout << scientific << y - sin(Arr[i] - z) << endl;       break;
      case 6: myout << scientific << Ecos2 / C->xx *100 << endl;        break;
      case 7: myout << scientific << Esin2 / C->yy *100 << endl;        break;
      default: myout << scientific << z << endl;                break;
   }


  }

  myout.close();


  mse = ssr / Ang->getnAngles();
  rms = sqrt(mse);

  // max_err = sqrt(max_err);


  cout << "  No of points = " << Ang->getnAngles() ;
  cout << " (nBreak = " << nBreak << " : " ;
  cout <<  100. * nBreak / Ang->getnAngles() << " % )" << endl;

  printf("  SSR: Sum of Squared Residual Angles    = ") ;
  printf("%12.7f (= %g) \n", ssr, ssr);
  printf("  MSR: Mean Squared Residual Angles      = ") ;
  printf("%12.7f (= %g) \n", mse, mse);
  printf("  RMS: Root Mean Squared Residual Angles = ") ;
  printf("%12.7f (= %g) \n", rms, rms);
  printf("  Min Squared Residual Angle Error       = ") ;
  printf("%12.7f (= %g) \n", min_err, min_err);
  printf("  Max Squared Residual Angle Error       = ") ;
  printf("%12.7f (= %g) \n", max_err, max_err);

  // cout << fixed << right << setw(12) << setprecision(7) << ssr << endl;
  // cout << fixed << right << setw(12) << setprecision(7) << mse << endl;
  // cout << fixed << right << setw(12) << setprecision(7) << rms << endl;
  // cout << fixed << right << setw(12) << setprecision(7) << max_err << endl;


}


//------------------------------------------------------------------------------
// Arr == Ang->A : Use A[i] for Index vs Residual Angles angles Plot
// Arr == Ang->B : Use B[i] for Angle vs Residual Angles angles Plot
//------------------------------------------------------------------------------
void P9_run_gnuplot(double *Arr, int mode, int rnd, Angles *Ang, Core *C, GPData *G)
```

```
{
  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");


  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {

    char fname[80], rnd_str[80];

    if (rnd) sprintf(rnd_str, "rnd");
    else     sprintf(rnd_str, "uni");

    if (Arr == Ang->A)  sprintf(fname, "res%d_vs_index_%s", mode, rnd_str);
    else                sprintf(fname, "res%d_vs_angle_%s", mode, rnd_str);

    G->set_fname(Ang, "egu3", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout  << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }


   char tstr[80];
   char istr[80];

   if (Arr == Ang->A)  sprintf(istr, "Index (mode%d)", mode);
   else                sprintf(istr, "Angle (mode%d)", mode);

   switch (mode) {
     case 0: sprintf(tstr, "UScale: A Residual Angle vs. %s", istr);   break;
     case 1: sprintf(tstr, "UScale: A Resolved Angle vs. %s", istr);   break;
     case 2: sprintf(tstr, "UScale: Full Cos Error vs. %s", istr);     break;
     case 3: sprintf(tstr, "UScale: Full Sin Error vs. %s", istr);     break;
     case 4: sprintf(tstr, "UScale: Resolved Cos Error vs. %s", istr); break;
     case 5: sprintf(tstr, "UScale: Resolved Sin Error vs. %s", istr); break;
     case 6: sprintf(tstr, "UScale: Norm. Resolved Cos Error vs. %s", istr); break;
     case 7: sprintf(tstr, "UScale: Norm. Resolved Sin Error vs. %s", istr); break;
     default: sprintf(tstr, "UScale: A Residual Angle vs. %s", istr);  break;
  }

  char ustring[80];
  if (Arr == Ang->A) {
     G->set_title(Ang, tstr);
```

```
     G->set_xlabel("increasing index values");
     if (mode == 0) {
        system("sort -k 3 angle.dat > t.dat; mv t.dat angle.dat");
     }
     sprintf(ustring, "%s", "3");
  } else {
     G->set_title(Ang, tstr);
     G->set_xlabel( "increasing angle values");
     sprintf(ustring, "%s", "2:3");
  }



  myout << "set title '" << G->title << "' " << endl;
  myout << "set xlabel \" " << G->xlabel << "\" " << endl;
  myout << "set ylabel \"residual angles in the z reg\" " << endl;


  myout << "plot 'angle.dat' using " << ustring << " with linespoints " << endl;

  cout << "................................................................" << endl;
  cout << G->title << endl;
  cout << "................................................................" << endl;

     switch (mode) {
        case 0: cout << "z "                  << endl;  break;
        case 1: cout << "Arr[i] - z"          << endl;  break;
        case 2: cout << "C->xx"               << endl;  break;
        case 3: cout << "C->yy"               << endl;  break;
        case 4: cout << "x - cos(Arr[i] - z)" << endl;  break;
        case 5: cout << "y - sin(Arr[i] - z)" << endl;  break;
        case 6: cout << "Ecos1 / C->xx *100"  << endl;  break;
        case 7: cout << "Esin1 / C->yy *100"  << endl;  break;
        default: cout << "z "                 << endl;  break;
     }

  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();


  system("gnuplot command.gp");

}




::::::::::::::
```

```
Angles.3.u4.plot_uscale_histogram.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>
# include <cstring>
# include <string>


# include "Core.hpp"
# include "Angles.hpp"
# include "GPData.hpp"

using namespace std;

//----------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Implementation Files
//     plot_uscale_histogram()
//     [residual angles in the reg z after cordic iterations]
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//     egu4.____.res_vs_angle.n%d.x%d.--%f.ext
//     egu4.____.dff_vs_angle.n%d.x%d.--%f.ext
//     egu4.____.res_corr_vs_angle.n%d.x%d.--%f.ext
//     egu4.____.dff_corr_vs_angle.n%d.x%d.--%f.ext
//
//----------------------------------------------------------------------------
void P8A_make_plot_data(uStat & S, int D_RB);
void P8B_make_plot_data(uStat & S, int nPoints, int D_RB, int R_SB);
```

```
void P8A_run_gnuplot(Angles *Ang, int nPoints, int C_RB, int D_RB, GPData *G);

//-----------------------------------------------------------------------------
//   Plot residual errors on the uniform scale
//-----------------------------------------------------------------------------
void Angles::plot_uscale_histogram (int nPoints =10000)
{


  if (checkNIters("plot_uscale_histogram")) return;


  if (~is_tscale_stat_done()) {
    cout << "..........................................." << endl;
    calc_tscale_statistics();
    cout << "..........................................." << endl;
  }





  Core C;

  char path[32] ="";
  int nBreak =0;

  C.setPath(path);
  C.setLevel(nIters);
  C.setThreshold(threshold);
  C.setNBreak(nBreak);

  C.setUseTh(useTh);
  C.setUseThDisp(useThDisp);
  C.setUseATAN(useATAN);


  int C_RB, D_RB, R_SB;
  //..............................................................
  // C_RB = 0 : Raw Data Plot
  // C_RB = 1 : C_RBelation Plot
  //..............................................................
  // R_SB = 0 : Use the signed values
  // R_SB = 1 : Use RMS values
  //..............................................................
  // D_RB = 0 : Residue vs. Angles Plot
  // D_RB = 1 : Difference Residue vs. Angles Plot
  //..............................................................

  GPData G(GnuTerm, getnAngles());
```

```
  //.............................................................
  // C_RB = 0:  (D_RB=0: res,  D_RB=1: dff)
  //.............................................................
  cout << "  + Residual Angles vs. Angles plot \n" ;
  //.............................................................
  P8A_make_plot_data(S, D_RB=0);
  P8A_run_gnuplot(this, nPoints, C_RB=0, D_RB=0, &G);
  //.............................................................
  cout << "  + Difference Residual Angles vs. Angles plot \n" ;
  //.............................................................
  P8A_make_plot_data(S, D_RB=1);
  P8A_run_gnuplot(this, nPoints, C_RB=0, D_RB=1, &G);
  //.............................................................

  //.............................................................
  // C_RB = 1:  (D_RB=0: res,  D_RB=1: dff)
  //.............................................................
  cout << "  + Correlation of Residual Angles vs. Angles plot \n" ;
  //.............................................................
  P8B_make_plot_data(S, nPoints, D_RB=0, R_SB=0);
  P8A_run_gnuplot(this, nPoints, C_RB=1, D_RB=0, &G);
  //.............................................................
  cout << "  + Correlation of Residual Angles vs. Angles plot \n" ;
  //.............................................................
  P8B_make_plot_data(S, nPoints, D_RB=1, R_SB=0);
  P8A_run_gnuplot(this, nPoints, C_RB=1, D_RB=1, &G);
  //.............................................................



}


//------------------------------------------------------------------------------
//   make plot data for residue or difference of residue
//------------------------------------------------------------------------------
//   D_RB = 0 : ARm (Angles - Residue)
//   D_RB = 1 : ADm (Angles - Difference Residue)
//------------------------------------------------------------------------------
void P8A_make_plot_data(uStat & S, int D_RB)
{
  mI lbound, ubound;

  if (D_RB) {
    lbound = S.ADm.begin();
    ubound = S.ADm.end();
    cout << "    . [Angles - difference residue] plot using ADm " << endl;
  } else {
    lbound = S.ARm.begin();
```

```
    ubound = S.ARm.end();
    cout << "     . [Angles - residue] plot using ARm " << endl;
  }


  ofstream myout;

  // write histogram data from delta array
  myout.open("angle.dat");

  mI i1;

  int n;
  char str[80];
  double tmp1, tmp2;

  n = 0;
  for (i1=lbound; i1!=ubound; i1++) {
    tmp1  = (*i1).first;
    tmp2  = (*i1).second;

/*
    if (n%sampling == 0) {
      sprintf(str, "%d %g %g ", n, tmp1, tmp2);
      myout << str << endl;
    }
*/

    sprintf(str, "%d %g %g ", n, tmp1, tmp2);
    myout << str << endl;

    n++;
  }

  myout.close();

}

//-------------------------------------------------------------------------------
//   make plot data for the CONVOLUTION of residue or difference of residue
//-------------------------------------------------------------------------------
//   R_SB = 0 : Use the signed values
//   R_SB = 1 : Use RMS values
//-------------------------------------------------------------------------------
//   D_RB = 0 : ARm (Angles - Residue)
//   D_RB = 1 : ADm (Angles - Difference Residue)
//-------------------------------------------------------------------------------
void P8B_make_plot_data(uStat & S, int nPoints, int D_RB, int R_SB)
{
  double A[2*nPoints], B[2*nPoints];
```

```
double tmp1;
int n;

mI i1;
vector<double>::iterator j1;


//--------------------------------------------------------------------------
// R_SB:1 - Consider signs of the residue and difference residue values
//--------------------------------------------------------------------------
if (R_SB) {

  if (D_RB) {
    cout << "    . Convolution for the signed [difference residue] using ADm" << endl;
    for (n=0, i1=S.ADm.begin(); i1!=S.ADm.end(); i1++) {
      tmp1 = (*i1).second;
      A[n++] = tmp1;
   }
  } else {
    cout << "    . Convolution for the signed [residue] using R " << endl;
    for (n=0, j1=S.R.begin(); j1!=S.R.end(); j1++) {
      tmp1 = (*j1);
      A[n++] = tmp1;
    }
  }

//--------------------------------------------------------------------------
// R_SB:0 - Consider signs of the residue and difference residue values
//--------------------------------------------------------------------------
} else {

  if (D_RB) {
    cout << "    . Convolution for the RMS [difference residue] using ADm" << endl;
    for (n=0, i1=S.ADm.begin(); i1!=S.ADm.end(); i1++) {
      tmp1 = (*i1).second;
      A[n++] = sqrt(tmp1*tmp1);
   }
  } else {
    cout << "    . Convolution for the RMS [residue] using ARm " << endl;
    for (n=0, i1=S.ARm.begin(); i1!=S.ARm.end(); i1++) {
      tmp1 = (*i1).second;
      A[n++] = sqrt(tmp1);
    }
  }

}

// write convolution data
char str[8];
```

```
  ofstream myout;

  myout.open("angle.dat");

  for (int k=0; k<nPoints; ++k) {
    B[k] = 0.;
    for (int i=0; i<nPoints; ++i) {
      B[k] += A[i] * A[(k+i) % nPoints];
    }
  }

  for (int k=0; k<nPoints; ++k) {
    sprintf(str, "%d %g %g ", k, k*S.step_ang, B[k]);
    myout << str << endl;
  }

  myout.close();

}



//------------------------------------------------------------------------------
//   Plot residue or differece
//------------------------------------------------------------------------------
//   D_RB = 0 : ARm (Angles - Residue)
//   D_RB = 1 : ADm (Angles - Difference Residue)
//------------------------------------------------------------------------------
void P8A_run_gnuplot(Angles *Ang, int nPoints, int C_RB, int D_RB, GPData *G)
{


  ofstream myout;

  // writing gnuplot commands
  myout.open("command.gp");


  G->set_prefix(Ang);
  G->set_suffix(Ang);

  myout << "set terminal " << GnuTerm << endl;
  if (strcmp(GnuTerm.c_str(), "wxt") != 0) {
    char fname[80];
    if (C_RB) {  // correlation plot
      if (D_RB) sprintf(fname, "corr_dff_vs_angle"); // diff
      else      sprintf(fname, "corr_res_vs_angle"); // res
    } else {   // raw data plot
      if (D_RB) sprintf(fname, "dff_vs_angle"); // diff
      else      sprintf(fname, "res_vs_angle"); // res
```

```
    }

    G->set_fname(Ang, "egu4", fname);
    Ang->epsList.push_back(G->fname);
    cout << "set output '" << G->fname << "'" << endl;
    cout << "pause" << endl;
    myout << "set output '" << G->fname << "'" << endl;
  }


  if (C_RB) {  // correlation plot
    if (D_RB) {  // diff
      G->set_title(Ang, "Corr(Difference Residue) vs. Angles");
      G->set_xlabel("Angles in the increasing order");
      G->set_ylabel("Corr(Difference Residue)");
    }
    else {       // res
      G->set_title(Ang, "Corr(Residue) vs. Angles");
      G->set_xlabel("Angles in the increasing order");
      G->set_ylabel("Corr(Residue)");
    }
  } else {   // raw data plot
    if (D_RB) {  // diff
      G->set_title(Ang, "Angles vs. Difference Residue");
      G->set_xlabel("Angles in the increasing order");
      G->set_ylabel("Difference Residue");
    }
    else {      // res
      G->set_title(Ang, "Angles vs. Residue Angles");
      G->set_xlabel("Angles in the increasing order");
      G->set_ylabel("Residue");
    }
  }


  myout << "set title \""    << G->title   <<  "\" " << endl;
  myout << "set xlabel \" "  << G->xlabel  << "\" " << endl;
  myout << "set ylabel \" "  << G->ylabel  << "\" " << endl;

  myout << "plot 'angle.dat' using " << "2:3" << " with points " << endl;

  cout << "..............................................................." << endl;
  cout << G->title << endl;
  cout << "..............................................................." << endl;

  if (strcmp(GnuTerm.c_str(), "wxt") == 0)
    myout << "pause mouse keypress" << endl;

  myout.close();
```

```
  system("gnuplot command.gp");


}






::::::::::::::
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:
//
//    2014.02.06
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
//
// double Angles::compute_angle (int idx, int level, char *s)
// void Angles::compute_angle_arrays ()
```

```
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//  Initialize and compute the arrays A[] and Ap[][]
//----------------------------------------------------------------------------
//  nIter = 3; Leaf nodes
//  level 3: A[0], A[1], A[2], A[3], A[4], A[5], A[6], A[7]       : 2^3 nodes
//----------------------------------------------------------------------------
//  nIter = 3; All nodes
//  level 0: A[0]                                                  : 2^0 nodes
//  level 1: A[1], A[2]                                            : 2^1 nodes
//  level 2: A[3], A[4], A[5], A[6]                                : 2^2 nodes
//  level 3: A[7], A[8], A[9], A[10], A[11], A[12], A[13], A[14]  : 2^3 nodes
//----------------------------------------------------------------------------
//  nIter = 3; Leaf nodes Ap[0~7]
//  level 3: {0,1,2,3,4,5,6,7}:"000","001","010","011","100","101","110","111"
//----------------------------------------------------------------------------
//  nIter = 3; All nodes Ap[0~15]
//  level 0: -
//  level 1: {0,1}:"0", "1"
//  level 2: {0,1,2,3}:"00","01","10","11"
//  level 3: {0,1,2,3,4,5,6,7}:"000","001","010","011","100","101","110","111"
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
void Angles::compute_angle_arrays ()
{

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

  //--------------------------------------------------------------------------
  // Store only the leaf angle values into the array A[]
  //--------------------------------------------------------------------------
  if (Leaf) {
    for (j=0; j<nAngles; ++j) {
      A[j] = compute_angle(j, nIters, Ap[j]);
      // cout << "A[" << j << "]=" << setw(12) << setprecision(8) << A[j] << endl;
    }
  }
  //--------------------------------------------------------------------------
  // Store all the angle values into the array A[]
  // can be considered as
  // all the leaf angle values at the level 0,                 2^0 values
  // all the leaf angle values at the level 1,                 2^1 values
  //   ...        ...           ...
  // all the leaf angle values at the final level nIters       2^nIters
```

```
  //-----------------------------------------------------------------------
  else {
    k=0;
    for (i=0; i<=nIters; ++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;
    }
  }

}




//-------------------------------------------------------------------------
//  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;
}
```