```
::::::::::::::
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 *~ *#
        \rm -f *.print *.tar *.a
::::::::::::::
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;


  //..................................................
  // Allocate and compute A, B, Ap arrays
  //..................................................
  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()
{


}


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



//--------------------------------------------------------------------------------
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;

}



/*****
  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;



//============================
```

```
//  defined classes
//
//  [ class XRange ]
//  [ class uStat  ]
//  [ class Angles ]
//============================


//-------------------------------------------------------------------------------
//   Purpose:
//
//     Class Angles Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.05.28
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-------------------------------------------------------------------------------

extern string GnuTerm;
extern string ofExt;


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


double compute_threshold(int nIters);


// 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();

  // .......................................................
  // compute_angle_arrays() allocates and computes A, B, Ap arrays
  // which is called at the constructor function Angles()
  // .......................................................
  double        *A;      // A[nAngles]  : angle array
  double        *B;      // B[nAngles]  : sorted angle array
  char          **Ap;    // Ap[nAngles] : angle path array
  // .......................................................


  // .......................................................
  // used by tscale and uscale statistics
  // .......................................................
  uStat S;


  // .......................................................
  // used by figures of latex files
  // .......................................................
  list<string> epsList;


  // .......................................................
  // nIters     : no of iterations (levels)
  // nAngles    : no of nodes (leaf nodes / all nodes)
  // nPoints    : no of angle points (uniform scale)
  // Leaf       : leaf nodes / all nodes
  // .......................................................
  // threshold  :
  // useTh      : using thresholding
  // useThDisp  : verbose thresholding
  // useATAN    : use arctan()
  // .......................................................
  void    setnIters     (int    val) { nIters     = val;};
  void    setnAngles    (int    val) { nAngles    = val;};
  void    setnPoints    (int    val) { nPoints    = val;};
  void    setLeaf       (double val) { threshold  = 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;    };
int     getLeaf       () { return Leaf;       };

double  getThreshold  () { return threshold;  };
int     getUseTh      () { return useTh;      };
int     getUseThDisp  () { return useThDisp;  };
int     getUseATAN    () { return useATAN;    };

int     checkNIters(string str);  // check for max iterations




//---------------------------------------------------------------------------
// a.   compute_angle_arrays
//...........................................................................
// 1.b1 plot_angle_tree             : plot binary angle trees
// 1.b2 plot_circle_angle           : plot angle vectors on a unit circle
// 1.b3 plot_line_angle             : plot angle vectors on a linear scale
// 1.b4 plot_quantization           : plot non-uniform quantization effects
//...........................................................................
// 2.t1 calc_tscale_statistics      : find statistics on the tree angle scale
// 2.t2 plot_tscale_statistics      : plot delta and angle-delta distribution
//*2.t3 plot_tscale_residual_angles : plot residuals-angle and residuals-index
//...........................................................................
//*3.u1 calc_uscale_statistics      : find statistics on the uniform angle scale
// 3.u2 plot_uscale_statistics      : plot delta and angle-delta distribution
//*3.u3 plot_uscale_residual_angles : plot residuals-angle and residuals-index
// 3.u4 plot_uscale_histogram       : plot histograms of the uniform scale
//---------------------------------------------------------------------------
//*:    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);
  //..........................................................................



  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; };
  void    set_mse (double val)       { mse       = val; };
  void    set_rms (double val)       { rms       = val; };
  void    set_max_err (double val)   { max_err   = val; };



  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;       };
  double  get_mse ()       { return mse;       };
  double  get_rms ()       { return rms;       };
  double  get_max_err ()   { return max_err;   };



  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; };



private:

  // ........................................................
  // nIters     : no of iterations (levels)
  // nAngles    : no of nodes (leaf nodes / all nodes)
  // nPoints    : no of angle points (uniform scale)
  // Leaf       : leaf nodes / all nodes
  // ........................................................
```

```
  // threshold  :
  // useTh      : using thresholding
  // useThDisp  : verbose thresholding
  // useATAN    : use arctan()
  // ........................................................
  int     nIters;
  int     nAngles;
  int     nPoints;
  int     Leaf;

  double  threshold;
  int     useTh;
  int     useThDisp;
  int     useATAN;

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

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


  int tscale_stat_done;
  int uscale_stat_done;

};


  //---------------------------------------------------------------------------
  // fname  = preStr + prefix + inStr + suffix . fext
  // fname  : "egbX.____.circle_ang.____.eps"
  //---------------------------------------------------------------------------
  // preStr  : "egb[1,2,3,4]", "egt[2,3]", "egu[2,3,4]"
  // prefix  : "Leaf_10" / "All_10"
  // inStr   :
  //         ...ang_tree[1,2,3]
  //            circle_ang
  //            line_ang
```

```
//            quantization
//           ...delta_dist_bin
//              delta_dist_val
//              delta_vs_angle
//              res[0-7]_vs_angle
//              res[0-7]_vs_index
//           ...angle_vs_dff
//              angle_vs_res
//              dff_hist
//              res_hist
//              angle_vs_dff
//              angle_vs_res
//              res[0-7]_vs_angle_rnd
//              res[0-7]_vs_index_rnd
//              corr_dff_vs_angle
//              corr_res_vs_angle
//              dff_vs_angle
//              res_vs_angle
// suffix  : "n4096x1th0.55"
// fext    : "eps"
//-----------------------------------------------------------------------------



//-----------------------------------------------------------------------------
// Core::cordic() is called by
//-----------------------------------------------------------------------------
// Angles.2.t3.plot_tscale_residual_angles.cpp:       P5_make_plot_data()
//
// Angles.3.u1.calc_uscale_statistics.cpp:            find_residual_angles()
// Angles.3.u3.plot_uscale_residual_angles.cpp:       P9_make_plot_data()
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// Angles::calc_tscale_statics() is called by
//-----------------------------------------------------------------------------
// Angles.2.t2.plot_tscale_statistics.cpp:            plot_tscale_statistics()
//
// Angles.3.u1.calc_uscale_statistics.cpp:            calc_uscale_statistics()
// Angles.3.u2.plot_uscale_statistics.cpp:            plot_uscale_statistics()
// Angles.3.u4.plot_uscale_histogram.cpp:             plot_uscale_histogram()
//
// Angles.cpp:                                        compute_threshold()
//
// Angles_tb.cpp:                                     main()
//-----------------------------------------------------------------------------


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

```
// Angles::calc_uscale_statistics() is called by
//----------------------------------------------------------------------------
// Angles_tb.cpp:                                     main()
//----------------------------------------------------------------------------




//--------------------------------------------------------------------------
//  [Angles.1.b1.plot_angle_tree.cpp]
//  ................................
//   Angles::plot_angle_tree()
//   - plot binary angle tree
//--------------------------------------------------------------------------
//      egb1.All_11.ang_tree1.n4095.eps   (levels vs angles)
//      egb1.All_11.ang_tree2.n4095.eps   (levels vs angles)
//      egb1.All_11.ang_tree3.n4095.eps   (levels vs angles)
//      egb1.Leaf_11.ang_tree1.n2048.eps  (levels vs angles)
//      egb1.Leaf_11.ang_tree2.n2048.eps  (levels vs angles)
//--------------------------------------------------------------------------
//  [Angles.1.b2.plot_circle_angle.cpp]
//  ..................................
//   Angles::plot_circle_angle ()
//   - plot angle vectors on a circle
//--------------------------------------------------------------------------
//      egb2.All_11.circle_ang.n4095.eps  (x vs y)
//      egb2.Leaf_11.circle_ang.n2048.eps (x vs y)
//--------------------------------------------------------------------------
//  [Angles.1.b3.plot_line_tree.cpp]
//  ................................
//   Angles::plot_line_angle ()
//   - plot angle vectors on a line
//--------------------------------------------------------------------------
//      egb3.All_11.line_ang.i0.n4095.eps  (angles vs jitter)
//      egb3.Leaf_11.line_ang.i0.n2048.eps (angles vs jitter)
//--------------------------------------------------------------------------
//  [Angles.1.b4.plot_quantization.cpp]
//  ..................................
//   Angles::plot_quantization ()
//   - plot quantization effects
//--------------------------------------------------------------------------
//      egb4.All_11.quantization.n4095.eps   (quantized angles vs angles)
//      egb4.Leaf_11.quantization.n2048.eps  (quantized angles vs angles)
//--------------------------------------------------------------------------


//--------------------------------------------------------------------------
//  [Angles.2.t1.calc_tscale_statistics.cpp]
//  ........................................
//   Angles::calc_tscale_statistics ()
```

```
//   - find Angles Statistics  --> member data
//----------------------------------------------------------------------------
//  [Angles.2.t2.plot_tscale_statistics.cpp]
//  ........................................
//   Angles::plot_tscale_statistics ()
//   - plot delta distribution and angle-delta
//----------------------------------------------------------------------------
//      egt2.____.delta_dist_bin.n4095.eps (frequency vs delta bins)
//      egt2.____.delta_dist_val.n4095.eps (frequency vs delta values)
//      egt2.____.delta_vs_angle.n4095.eps (delta angles vs angles)
//----------------------------------------------------------------------------
//  [Angles.2.t3.plot_tscale_residual_angles.cpp]
//  .............................................
//   Angles::plot_tscale_residual_angles()
//   - plot residuals-angle and residuals-index
//----------------------------------------------------------------------------
//      egt3.____.res0_vs_angle.n4095.eps  (residual angles vs index)
//                res1                     (resolved angles vs index)
//                res2                     (full cos error vs index)
//                res3                     (full sin error vs index)
//                res4                     (resolved cos error vs index)
//                res5                     (resolved sin error vs index)
//                res6                     (norm resolved cos error vs index)
//                res7                     (norm resolved sin error vs index)
//      egt3.____.res0_vs_index.n4095.eps  (residual angles vs index)
//                res1                     (resolved angles vs index)
//                res2                     (full cos error vs index)
//                res3                     (full sin error vs index)
//                res4                     (resolved cos error vs index)
//                res5                     (resolved sin error vs index)
//                res6                     (norm resolved cos error vs index)
//                res7                     (norm resolved sin error vs index)
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//  [Angles.3.u1.calc_uscale_statistics.cpp]
//  ........................................
//   Angles::calc_uscale_statistics ()
//   - computing uniform scale statistics
//----------------------------------------------------------------------------
//  [Angles.3.u2.plot_uscale_statistics]
//  ....................................
//   Angles::plot_uscale_statistics ()
//   - plotting uniform scale statistics
//----------------------------------------------------------------------------
//      egu2.____.angle_vs_dff.n4095.eps  (frequency vs difference residue)
//      egu2.____.angle_vs_res.n4095.eps  (frequency vs ressidue)
//      egu2.____.dff_hist.n4095.eps      (residue vs angles)
//      egu2.____.res_hist.n4095.eps      (
```

```
  //-----------------------------------------------------------------------
  //  [Angles.3.u3.plot_uscale_residual_angles.cpp]
  //  .............................................
  //   Angles::plot_uscale_residual_angles()
  //   - plotting residual angles in the reg z after cordic iterations
  //-----------------------------------------------------------------------
  //      egu3.____.res0_vs_angle_rnd.n4095.eps  (rnd residual angles vs index)
  //                res1                         (rnd resolved angles vs index)
  //                res2                         (rnd full cos error vs index)
  //                res3                         (rnd full sin error vs index)
  //                res4                         (rnd resolved cos error vs index)
  //                res5                         (rnd resolved sin error vs index)
  //                res6                         (rnd norm resolved cos error vs index)
  //                res7                         (rnd norm resolved sin error vs index)
  //      egu3.____.res0_vs_index_rnd.n4095.eps  (rnd residual angles vs index)
  //                res1                         (rnd resolved angles vs index)
  //                res2                         (rnd full cos error vs index)
  //                res3                         (rnd full sin error vs index)
  //                res4                         (rnd resolved cos error vs index)
  //                res5                         (rnd resolved sin error vs index)
  //                res6                         (rnd norm resolved cos error vs index)
  //                res7                         (rnd norm resolved sin error vs index)
  //-----------------------------------------------------------------------
  //  [Angles.3.u4.plot_uscale_histogram.cpp]
  //  .......................................
  //   Angles::plot_uscale_histogram()
  //   - plotting uniform scale histograms
  //-----------------------------------------------------------------------
  //      egu4.____.corr_dff_vs_angle.n4095.eps
  //      egu4.____.corr_res_vs_angle.n4095.eps
  //      egu4.____.dff_vs_angle.n4095.eps
  //      egu4.____.res_vs_angle.n4095.eps
  //-----------------------------------------------------------------------
```

```
::::::::::::::
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.All_11.ang_tree1.n4095.eps
//       egb1.All_11.ang_tree2.n4095.eps
//       egb1.All_11.ang_tree3.n4095.eps
//       egb1.Leaf_11.sub_tree1.n2048.eps
//       egb1.Leaf_11.sub_tree2.n2048.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, int m);

//--------------------------------------------------------------------------------
//  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, mode;


  if (checkNIters("plot_angle_tree")) return;

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


  ofstream myout;
  char fname[256];

  //----------------------------------------
  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);

      GPData G(GnuTerm, nAngles);

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


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

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

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

      GPData G(GnuTerm, nAngles);
```

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

      //----------------------------------------
      B1_run_gnuplot(this, &G, i+2, m); // 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, int m)
{
  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, "sub_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;
  }

  //------------------------------------------------------------
  // Leaf: flag=1 : block index view
  // Leaf: flag=2 : offset index view
  //------------------------------------------------------------
  char tstr[256];

  if (Ang->getLeaf()) {
    if (flag == 1) {
      sprintf(tstr, "Subtree plot of a level %d nodes (block index view)", m);
      G->set_title(Ang, tstr);

      myout << "set title '" << G->title << "' " << endl;
      myout << "set xlabel \"Angles in degree\" " << endl;
      myout << "set ylabel \"block index \" " << endl;
      myout << "set format x \"%.0f\" " << endl;
      myout << "set format y \"%.0f\" " << endl;

      myout << "set xrange [-100:100]" << endl;
```

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

      if (strcmp(GnuTerm.c_str(), "wxt") == 0)
        myout << "pause mouse keypress" << endl;
    } else if (flag == 2) {
      sprintf(tstr, "Subtree plot of a level %d nodes (offset index view)", m);
      G->set_title(Ang, tstr);

      myout << "set title '" << G->title << "' " << endl;
      myout << "set xlabel \"Angles in degree\" " << endl;
      myout << "set ylabel \"offset index \" " << endl;
      myout << "set format x \"%.0f\" " << endl;
      myout << "set format y \"%.0f\" " << endl;

      myout << "set xrange [-100:100]" << endl;

      myout << "plot 'angle2.dat' using 1:2:3:4  notitle ";
      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  notitle ";
      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  notitle  ";
      myout << "with vectors head filled lt 3 ,  " ;
      myout << "     'angle2.dat' using 1:2:3:4  notitle ";
      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  notitle ";
      myout << "with vectors head filled lt 3 ,  " ;
      myout << "     'angle2.dat' using 1:2:3:4  notitle ";
      myout << "with vectors head filled lt 4  , " ;
      myout << "     'angle3.dat' using 1:2:3:4  notitle ";
      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.All_11.circle_ang.n4095.eps
//      egb2.Leaf_11.circle_ang.n2048.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 notitle ";
  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.All_11.line_ang.i0.n4095.eps
//      egb3.Leaf_11.line_ang.i0.n2048.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 notitle ";
  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.All_11.quantization.n4095.eps
//      egb4.Leaf_11.quantization.n2048.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 notitle, ";
  myout << "     'angle.dat' using 1:3 with lines notitle" << 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.cpp]
//
//      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.n4095.eps
//      egt2.____.delta_dist_val.n4095.eps
//      egt2.____.delta_vs_angle.n4095.eps
//
//--------------------------------------------------------------------------
// 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 notitle " << 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 notitle" << 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.____.res0_vs_angle.n4095.eps
//                res2
//                res3
//                res4
//                res5
//                res6
//                res7
//      egt3.____.res0_vs_index.n4095.eps
//                res2
//                res3
//                res4
//                res5
//                res6
//                res7
//
//----------------------------------------------------------------------------
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, Esin1;
      double Ecos2, 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 notitle" << 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 ()
//
//       - computing 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.cpp]
//
//       Angles::plot_uscale_statistics ()
//
//       - computing uniform scale statistics
//
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//      egu2.____.angle_vs_dff.n4095.eps
//      egu2.____.angle_vs_res.n4095.eps
//      egu2.____.dff_hist.n4095.eps
//      egu2.____.res_hist.n4095.eps
//
//----------------------------------------------------------------------------
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 notitle";
```

```
  myout << ",    'angle.dat' using " << "1:3" << " with lines notitle" << 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", "difference residue");
     sprintf(ylabel, "%s", "angles in the increasing order");
  } else {
     sprintf(title, "%s", "UScale: Angles vs. Residue");
     sprintf(xlabel, "%s", "residue (sqrt(z*z))");
     sprintf(ylabel, "%s", "angles in the increasing order");
  }
  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 notitle" << 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()
//
//       - plotting 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.____.res0_vs_angle_rnd.n4095.eps
//                res1
//                res2
//                res3
//                res4
//                res5
//                res6
//                res7
//      egu3.____.res0_vs_index_rnd.n4095.eps
//                res1
//                res2
//                res3
//                res4
//                res5
```

```
//                res6
//                res7
//
//-----------------------------------------------------------------------------
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, Esin1;
    double Ecos2, 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 notitle" << 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
//
//      [Angles.3.u4.plot_uscale_histogram.cpp]
//
//      Angles::plot_uscale_histogram()
//
//       - plotting uniform scale histograms
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2013.07.27
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//      egu4.____.corr_dff_vs_angle.n4095.eps
//      egu4.____.corr_res_vs_angle.n4095.eps
//      egu4.____.dff_vs_angle.n4095.eps
//      egu4.____.res_vs_angle.n4095.eps
//
//----------------------------------------------------------------------------
```

```
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 AT[2*nPoints], BT[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;
        AT[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);
        AT[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;
        AT[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;
        AT[n++] = sqrt(tmp1);
      }
    }

  }
```

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

  myout.open("angle.dat");

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

  for (int k=0; k<nPoints; ++k) {
    sprintf(str, "%d %g %g ", k, k*S.step_ang, BT[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, "Difference Residue vs. Angles");
      G->set_xlabel("Angles in the increasing order");
      G->set_ylabel("Difference Residue");
    }
    else {      // res
      G->set_title(Ang, "Residue Angles vs. 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 notitle" << 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 ()
{

  //......................................................
  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));
  }


  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) {
      // angle value of the j-th child at the nIters level (leaf level)
```

```
      // compute the angle and path string
      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) {

        // angle value of the j-th child at the "levle" level
        // compute angle and path string
        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;
}




::::::::::::::
Core.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


BurkDir = /home/young/MyWork/2.cordic_cpp/Burkadt
GHDLDir = /home/young/MyWork/7.cordic_accuracy/IF.GHDL

VPATH = ${BurkDir}:${GHDLDir}        \

INCS =  -I${BurkDir} -I${GHDLDir}    \
```

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

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

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


#--------------------------------------------------------------
# Classes
#--------------------------------------------------------------
SRC = Core.hpp                        \
      Core.cpp                        \
      Core.1.fptr1.cordic_org.cpp     \
      Core.1.fptr2.cordic_burk.cpp    \
      Core.1.fptr3.cordic_vhdl.cpp    \
      Core.2.wrap1.cordic_stat.cpp    \
      Core.2.wrap2.cordic_break.cpp   \

OBJ = Core.o                          \
      Core.1.fptr1.cordic_org.o       \
      Core.1.fptr2.cordic_burk.o      \
      Core.1.fptr3.cordic_vhdl.o      \
      Core.2.wrap1.cordic_stat.o      \
      Core.2.wrap2.cordic_break.o     \


INC = Core.hpp                        \

LIB = libcordic-core.a                \

EXE = Core_tb                         \



#--------------------------------------------------------------
Core.o : cordic-burk cordic-ghdl ${SRC}
        g++ -c -Wall -g ${INCS} Core.cpp

cordic-burk :
        cd ${BurkDir}; make all;
        # cordic-burk library to ${LIBD}

cordic-ghdl :
        cd ${GHDLDir}; make all;
        # cordic-ghdl library to ${LIBD}
        # cordic_vtb executable to ${EXED}
```

```
#---------------------------------------------------------------
all : ${OBJ} cordic-burk cordic-ghdl
#       ar -rcs libcordic-core.a cordic_core.o
        ar -cvq libcordic-core.a ${OBJ}
        \cp -f ${LIB} ${LIBD}
        \cp -f ${INC} ${INCD}
        \rm -f ${OBJ}

print : Core.make Core_tb.cpp ${SRC}
        /bin/more $? > Core.print

tar : Core.make Core_tb.cpp ${SRC}
        tar cvf Core.tar $?

clean :
        \rm -f *.o *~ *#
        \rm -f *.print *.tar *.a
```

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

using namespace std;

#include "Core.hpp"
#include "Core_tb.hpp"


//--------------------------------------------------------------------------
//   Purpose:
//
//     Test various cordic implementations
//
//  Discussion:
//
//
//  Licensing:
//
```

```
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.03.27
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
//
// Core_tb.cpp
// Core_tb.wrap1.cpp
// Core_tb.wrap2.cpp
//
//-----------------------------------------------------------------------------



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

  int nIters = 10;
  double x, y, z;


  //................................................
  Core C;
  //................................................

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

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

  C.setUseTh(0);
  C.setUseThDisp(0);
  C.setUseATAN(0);

  C.setMode(1);

  C.dispVars();
```

```
//.................................................

double pi = C.getPi();
double K  = C.getK();


//--------------------------------------------------------------------------
// printf ("\nGrinding on [K, 0, 0]\n");
// Circular (X0C, 0L, 0L);
//--------------------------------------------------------------------------
x = 1 / K;
y = 0.0;
z = 0.0;

printf ("\nGrinding on [K, 0, 0]\n");
cout << "----------------------------------------------\n";
printf("xi= %f yi= %f zi= %f \n", x, y, z);

C.cordic(&x, &y, &z);

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


//--------------------------------------------------------------------------
// printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
// Circular (X0C, 0L, HalfPi / 3L);
//--------------------------------------------------------------------------
x = 1 / K ;
y = 0.0;
z = pi / 6.0;

printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
cout << "----------------------------------------------\n";
printf("xi= %f yi= %f zi= %f \n", x, y, z);

C.cordic(&x, &y, &z);

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




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

```
  printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
  cout << "-----------------------------------------------\n";
  printf("xi= %f yi= %f zi= %f \n", x, y, z);

  C.cordic(&x, &y, &z);

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



  //----------------------------------------------------------------------
  // printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
  // Circular (X0C, 0L, 2L * (HalfPi / 3L));
  //----------------------------------------------------------------------
  x = 1 / K;
  y = 0.0;
  z = pi / 3.0;

  printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
  cout << "-----------------------------------------------\n";
  printf("xi= %f yi= %f zi= %f \n", x, y, z);

  C.cordic(&x, &y, &z);

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


  return 0;

}
::::::::::::::
Core.hpp
::::::::::::::
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <ctime>
#include <string.h>


using namespace std;


const int ANGLES_LENGTH =60;
const int KPROD_LENGTH =33;

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

```
//      Class Core Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.14
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//----------------------------------------------------------------------------
//   Mode                       setMode(),              getMode()
//   UseTh              setUseTh(),             getUseTh()
//   UseThDisp          setUseThDisp()          getUseThDisp()
//   UseATAN            setUseATAN()            getUseATAN()
//
//   Level              setLevel(),             getLevel()
//   Path                       setPath(),              getPath()
//   Threshold          setThreshold(),         getThreshold()
//   nBreak;            setNBreak(),            getNBreak()
//   nBreakInit;                setNBreakInit(),        getNBreakInit()
//
//   Pi                 setPi(),                getPi()
//   K                  setK(),                 getK()
//   Angles             setAngles()
//   KProd              setKProd()
//
//                      dispVars()
//                      initAcc()
//
//                      cordic_fptr()
//                      setFuncPtr()
//                      initScale()
//
//                      cordic()
//                      cordic_stat()
//                      cordic_break()
//
//   zz;
//   xx, sum_xx, sum_xx2;   //    SCE,        SSE,        SRE;
//   yy, sum_yy, sum_yy2;   //   sSCE,       sSSE,       sSRE;
```

```
//   sum_xx_n, sum_xx2_n;   //   mSCE,       mSSE,       mSRE;
//   sum_yy_n, sum_yy2_n;   //  rmSCE,      rmSSE,      rmSRE;
//   max_err, max_errn;     // minSCE,     minSSE,     minSRE;
//   cnt_xx, cnt_yy;        // maxSCE,     maxSSE,     maxSRE;
//
//------------------------------------------------------------------------------
//
//   Core instantiated in
//     Angles.2.t3.plot_tscale_residual_angles.cpp
//     Angles.3.u1.calc_uscale_statistics.cpp
//     Angles.3.u3.plot_uscale_residual_angles.cpp
//     Angles.3.u4.plot_uscale_histogram.cpp
//
//   calc_tscale_statistics used in
//     Angles.2.t2.plot_tscale_statistics.cpp:    calc_tscale_statistics();
//     Angles.3.u1.calc_uscale_statistics.cpp:    calc_tscale_statistics();
//     Angles.3.u2.plot_uscale_statistics.cpp:    calc_tscale_statistics();
//     Angles.3.u4.plot_uscale_histogram.cpp:    calc_tscale_statistics();
//
//------------------------------------------------------------------------------


class Core;
//--------------------------------------------------------------
// used via a pointer to a function  (friend functions)
//--------------------------------------------------------------
void    cordic_org  ( double *x, double *y, double *z, Core *C );
void    cordic_burk ( double *x, double *y, double *z, Core *C  );
void    cordic_vhdl ( double *x, double *y, double *z, Core *C  );



class Core
{

public:

  Core();
  ~Core();


  void    setMode(int m);
  void    setUseTh(int flag);
  void    setUseThDisp(int flag);
  void    setUseATAN(int flag);

  int     getMode();
  int     getUseTh();
  int     getUseThDisp();
  int     getUseATAN();
```

```
// ---------------------------------------------------------------
// level      : Number of Iteration = Height of binary angle tree
// path       : path string in the binary angle tree
// threshold  : threshold for breaking the cordic algorithm's loop
// nBreak     : number of such breaking events
// nBreakInit : initialize the nBreak counter
// ---------------------------------------------------------------
void    setLevel(int l);
void    setPath(char *p);
void    setThreshold(double th);
void    setNBreak(int nB);
void    setNBreakInit(int nBInit);

int     getLevel();
void    getPath(char *p);
double  getThreshold();
int     getNBreak();
int     getNBreakInit();


//----------------------------------------------
void    setPi();
void    setK();
void    setAngles();
void    setKprod();

double  getPi();
double  getK();


//----------------------------------------------
double *getAngles();
double *getKprod();

//----------------------------------------------
void    dispVars();

void    initAcc () ;



//--------------------------------------------------------------
// used via a pointer to a function
//--------------------------------------------------------------
friend void    cordic_org  ( double *x, double *y, double *z, Core *C );
friend void    cordic_burk ( double *x, double *y, double *z, Core *C );
friend void    cordic_vhdl ( double *x, double *y, double *z, Core *C );
```

```
  //------------------------------------------------------------
  // mode = 1: cordic_fptr = & cordic_org;
  // mode = 2: cordic_fptr = & cordic_burk;
  // mode = 3: cordic_fptr = & cordic_vhdl;
  //------------------------------------------------------------
  void    (* cordic_fptr) (double *x, double *y, double *z, Core *C );

  void    setFuncPtr();

  void    initScale(double *x, double *y);


  //------------------------------------------------------------
  // Wrapper Function
  //------------------------------------------------------------
  void    cordic (double *x, double *y, double *z);
  void    cordic_stat (double *x, double *y, double *z, int& cnt, int& xx, int& yy, int& zz);
  void    cordic_break ( double *x, double *y, double *z, int& init);




public:
    double zz;

    // xx = (*x - cosz); sum_xx += xx; sum_xx2 += (xx*xx);
    // yy = (*y - sinz); sum_yy += yy; sum_yy2 += (yy*yy);

    double xx, sum_xx, sum_xx2;
    double yy, sum_yy, sum_yy2;

    double sum_xx_n, sum_xx2_n;
    double sum_yy_n, sum_yy2_n;

    double max_err, max_errn;
    int    cnt_xx, cnt_yy;


    double    SCE,        SSE,        SRE;
    double   sSCE,       sSSE,       sSRE;
    double   mSCE,       mSSE,       mSRE;
    double  rmSCE,      rmSSE,      rmSRE;
    double minSCE,     minSSE,     minSRE;
    double maxSCE,     maxSSE,     maxSRE;


private:
  int           mode;

  int           useTh;
```

```
  int           useThDisp;
  int           useATAN;

  int           level;
  char          path[256];

  double        threshold;
  int           nBreak;
  int           nBreakInit;

  double        pi;
  double        K;
  double        angles[ANGLES_LENGTH];
  double        kprod[KPROD_LENGTH];

};





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


#include "Core.hpp"


using namespace std;

//-----------------------------------------------------------------------------
//   Purpose:
//
//     Class Core Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.14
```

```
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------------
// Mode                 setMode(),              getMode()
// UseTh                setUseTh(),             getUseTh()
// UseThDisp            setUseThDisp()          getUseThDisp()
// UseATAN              setUseATAN()            getUseATAN()
//
// Level                setLevel(),             getLevel()
// Path                 setPath(),              getPath()
// Threshold            setThreshold(),         getThreshold()
// nBreak;              setNBreak(),            getNBreak()
// nBreakInit;          setNBreakInit(),        getNBreakInit()
//
// Pi                   setPi(),                getPi()
// K                    setK(),                 getK()
// Angles               setAngles()
// KProd                setKProd()
//
//                      dispVars()
//                      initAcc()
//
//                      cordic_fptr()
//                      setFuncPtr()
//                      initScale()
//
//                      cordic()
//                      cordic_stat()
//                      cordic_break()
//
// zz;
// xx, sum_xx, sum_xx2;   //    SCE,        SSE,        SRE;
// yy, sum_yy, sum_yy2;   //   sSCE,       sSSE,       sSRE;
// sum_xx_n, sum_xx2_n;   //   mSCE,       mSSE,       mSRE;
// sum_yy_n, sum_yy2_n;   //  rmSCE,      rmSSE,      rmSRE;
// max_err, max_errn;     // minSCE,     minSSE,     minSRE;
// cnt_xx, cnt_yy;        // maxSCE,     maxSSE,     maxSRE;
//-----------------------------------------------------------------------------


Core::Core()
{
  setPi();
  setK();
  setAngles();
```

```
  setKprod();

  mode          = 1;

  useTh         = 1;
  useThDisp     = 1;
  useATAN       = 0;

  level         = 10;

  nBreak        = 0;
  nBreakInit    = 0;
  threshold     = 0.0001;

  strcpy(path, "");



}


Core::~Core()
{

}


//--------------------------------------------------------------------------------
//  Accessor & Changer
//--------------------------------------------------------------------------------
void    Core::setMode      (int m)    { mode      = m;    }
void    Core::setUseTh     (int flag) { useTh     = flag; }
void    Core::setUseThDisp (int flag) { useThDisp = flag; }
void    Core::setUseATAN   (int flag) { useATAN   = flag; }


int     Core::getMode()      { return(mode);      }
int     Core::getUseTh()     { return(useTh);     }
int     Core::getUseThDisp() { return(useThDisp); }
int     Core::getUseATAN()   { return(useATAN);   }

//--------------------------------------------------------------------------------
void    Core::setLevel      (int l)      { level      = l;      }
void    Core::setNBreak     (int nB)     { nBreak     = nB;     }
void    Core::setNBreakInit (int nBInit) { nBreakInit = nBInit; }
void    Core::setThreshold  (double th)  { threshold  = th;     }
void    Core::setPath       (char *p)    { strcpy(path, p);     }


int     Core::getLevel()       { return(level);      }
```

```
int     Core::getNBreak()      { return(nBreak);      }
int     Core::getNBreakInit()  { return(nBreakInit);  }
double  Core::getThreshold()   { return(threshold);   }
void    Core::getPath(char *p) { strcpy(p, path);     }

//----------------------------------------------------------------------------
double *Core::getAngles()      { return angles;       }
double *Core::getKprod()       { return kprod;        }


void    Core::dispVars() {
  printf("...............................................................\n");
  printf(". CORDIC Parameter Settings \n");
  printf("...............................................................\n");
  printf(". mode = %d \n", mode);
  printf(". (1: cordic_org, 2: cordic_burk, 3: cordic_vhdl)\n\n");

  printf(". useTh = %d \n", useTh);
  printf(". useThDisp = %d \n", useThDisp);
  printf(". useATAN = %d \n\n", useATAN);

  printf(". level = %d \n", level);
  printf(". path = %s \n\n", path);

  printf(". threshold = %f \n", threshold);
  printf(". nBreak = %d \n", nBreak);
  printf(". nBreakInit = %d \n", nBreakInit);
  printf("...............................................................\n");

if (0) {
  for (int i=0; i < ANGLES_LENGTH; ++i) {
    printf("angles[%d]=%f \n", i, angles[i]);
  }

  for (int i=0; i < KPROD_LENGTH; ++i) {
    printf("kprod[%d]=%f \n", i, kprod[i]);
  }
}


}


  double        pi;
  double        K;
  double        angles[ANGLES_LENGTH];
  double        kprod[KPROD_LENGTH];
```

```
//----------------------------------------------------------------------------
//  Initialize variables for statistics
//----------------------------------------------------------------------------
void Core::initAcc ()
{
    max_err =0.0,  max_errn =0.0;

    sum_xx =0.0,   sum_xx2 =0.0;
    sum_yy =0.0,   sum_yy2 =0.0;

    sum_xx_n =0.0, sum_xx2_n =0.0;
    sum_yy_n =0.0, sum_yy2_n =0.0;

    cnt_xx =0.0,   cnt_yy =0.0;
}



//----------------------------------------------------------------------------
//  Initialize the constants: pi, K
//----------------------------------------------------------------------------
void Core::setPi()
{
  pi = 3.141592653589793;
}


void Core::setK()
{
  K = 1.646760258121;
}

double Core::getPi() { return pi; }
double Core::getK()  { return K;  }


//----------------------------------------------------------------------------
//  Initialize the array Angles[ANGLES_LENGTH]
//----------------------------------------------------------------------------
void Core::setAngles()
{
  double angles_in[ANGLES_LENGTH] = {
    7.8539816339744830962E-01,
    4.6364760900080611621E-01,
    2.4497866312686415417E-01,
    1.2435499454676143503E-01,
    6.2418809995957348474E-02,
    3.1239833430268276254E-02,
    1.5623728620476830803E-02,
```

```
    7.8123410601011112965E-03,
    3.9062301319669718276E-03,
    1.9531225164788186851E-03,
    9.7656218955931943040E-04,
    4.8828121119489827547E-04,
    2.4414062014936176402E-04,
    1.2207031189367020424E-04,
    6.1035156174208775022E-05,
    3.0517578115526096862E-05,
    1.5258789061315762107E-05,
    7.6293945311019702634E-06,
    3.8146972656064962829E-06,
    1.9073486328101870354E-06,
    9.5367431640596087942E-07,
    4.7683715820308885993E-07,
    2.3841857910155798249E-07,
    1.1920928955078068531E-07,
    5.9604644775390554414E-08,
    2.9802322387695303677E-08,
    1.4901161193847655147E-08,
    7.4505805969238279871E-09,
    3.7252902984619140453E-09,
    1.8626451492309570291E-09,
    9.3132257461547851536E-10,
    4.6566128730773925778E-10,
    2.3283064365386962890E-10,
    1.1641532182693481445E-10,
    5.8207660913467407226E-11,
    2.9103830456733703613E-11,
    1.4551915228366851807E-11,
    7.2759576141834259033E-12,
    3.6379788070917129517E-12,
    1.8189894035458564758E-12,
    9.0949470177292823792E-13,
    4.5474735088646411896E-13,
    2.2737367544323205948E-13,
    1.1368683772161602974E-13,
    5.6843418860808014870E-14,
    2.8421709430404007435E-14,
    1.4210854715202003717E-14,
    7.1054273576010018587E-15,
    3.5527136788005009294E-15,
    1.7763568394002504647E-15,
    8.8817841970012523234E-16,
    4.4408920985006261617E-16,
    2.2204460492503130808E-16,
    1.1102230246251565404E-16,
    5.5511151231257827021E-17,
    2.7755575615628913511E-17,
    1.3877787807814456755E-17,
```

```
    6.9388939039072283776E-18,
    3.4694469519536141888E-18,
    1.7347234759768070944E-18 };

    for (int i=0; i<ANGLES_LENGTH; ++i) {
      angles[i] = angles_in[i];
    }

}


//-----------------------------------------------------------------------------
//  Initialize the array kprod[ANGLES_LENGTH]
//-----------------------------------------------------------------------------
void Core::setKprod()
{

  double kprod_in[KPROD_LENGTH] = {
    0.70710678118654752440,
    0.63245553203367586640,
    0.61357199107789634961,
    0.60883391251775242102,
    0.60764825625616820093,
    0.60735177014129595905,
    0.60727764409352599905,
    0.60725911229889273006,
    0.60725447933256232972,
    0.60725332108987516334,
    0.60725303152913433540,
    0.60725295913894481363,
    0.60725294104139716351,
    0.60725293651701023413,
    0.60725293538591350073,
    0.60725293510313931731,
    0.60725293503244577146,
    0.60725293501477238499,
    0.60725293501035403837,
    0.60725293500924945172,
    0.60725293500897330506,
    0.60725293500890426839,
    0.60725293500888700922,
    0.60725293500888269443,
    0.60725293500888161574,
    0.60725293500888134606,
    0.60725293500888127864,
    0.60725293500888126179,
    0.60725293500888125757,
    0.60725293500888125652,
    0.60725293500888125626,
    0.60725293500888125619,
```

```
    0.60725293500888125617 };

    for (int i=0; i<KPROD_LENGTH; ++i) {
      kprod[i] = kprod_in[i];
    }

}

//------------------------------------------------------------------------------
//  cordic ftpr function
//------------------------------------------------------------------------------
void Core::setFuncPtr() {

   switch (mode) {
        case 1 : cordic_fptr = cordic_org;   break;
        case 2 : cordic_fptr = cordic_burk;  break;
        case 3 : cordic_fptr = cordic_vhdl;  break;
        default: cordic_fptr = cordic_org;   break;
   }

}


//------------------------------------------------------------------------------
//  Adjust Initial Scaling Factor (starting with (1,0) or (K, 0))
//------------------------------------------------------------------------------
void Core::initScale(double *x, double *y) {

   switch (mode) {
        case 1 : (*x) = (*x) * K;  break;
        case 2 :   break;
        case 3 :   break;
        default: (*x) = (*x) * K;  break;
   }

}


//------------------------------------------------------------------------------
//  cordic wrapper function
//------------------------------------------------------------------------------
void Core::cordic(double *x, double *y, double *z )
{

   setFuncPtr();

   initScale(x, y);

   (* cordic_fptr)(x, y, z, this);
```

```
   return;
}








::::::::::::::
Core.1.fptr1.cordic_org.cpp
::::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>

#include "Core.hpp"


using namespace std;

//------------------------------------------------------------------------------
//  Purpose:
//
//     stand alone cordic_org() implementation file
//     friend function of class Core
//
//     [Core.1.fptr1.cordic_org.cpp]
//
//-----------------------------------------------------------------------------
//  CORDIC returns the sine and cosine using the CORDIC method.
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.12
//
//  Author:
//
//    Based on MATLAB code in a Wikipedia article.
//
//    Modifications by John Burkardt
//
//    Further modified by Young W. Lim
//
```

```
//  Parameters:
//
//    Input:
//      *x: x coord of an init vector
//      *y: y coord of an init vector
//      *z: angle (-90 <= angle <= +90)
//
//      level : number of iteration
//              A value of 10 is low.  Good accuracy is achieved
//              with 20 or more iterations.
//
//    Output:
//      *xo: x coord of a final vector
//      *yo: y coord of a final vector
//      *zo: angle residue
//
//  Local Parameters:
//
//    Local, real ANGLES(60)
//                ANGLES(j) = arctan ( (1/2)^(0:59) );
//                ANGLES_LENGTH
//
//    Local, real KPROD(33)
//                KPROD(j) = product ( 0 <= i <= j ) K(i)
//                    K(i) = 1 / sqrt ( 1 + 2^{-2i} )
//                KPROD_LENGTH
//
//----------------------------------------------------------------------------
//
//   C->useATAN  : using arctang function
//   C->useTh    : using thresholding
//      C->nBreakInit
//      C->nBreak
//      C->useThDisp
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
void cordic_org ( double *x, double *y, double *z, Core *C )
{
  double angle;
  double factor;

  double sigma;
  double poweroftwo;
  double theta;

  double xn, yn;
```

```
int j;



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

poweroftwo = 1.0;

if (C->useATAN)               // if useATAN, then use arctangent
  angle = atan( 1. );
else                          // otherwise, use angles array values
  angle = (C->angles)[0];



//----------------------------------------------------------------------
for ( j = 1; j <= C->level; j++ )
//----------------------------------------------------------------------
{


    if ( theta < 0.0 ) sigma = -1.0;  // if theta is pos, subtract
    else               sigma = +1.0;  // otherwise, add

    //..................................................................
    // path[i] : the path to the leaf angle in the binary angle tree
    //..................................................................
    if ( theta < 0.0 ) (C->path)[j-1] = '0'; // left child  : '0' (subtracting)
    else               (C->path)[j-1] = '1'; // right child : '1' (adding)
    (C->path)[j] = '\0';                     // null terminated string


    //..................................................................
    //   x' = cos(a)*x -sin(a)*y  ==> x' = cos(a) {x        -y*tan(a)}
    //   y' = sin(a)*x +cos(a)*y  ==> y' = cos(a) {x*tan(a)  y       }
    //..................................................................
    //   Generally, cos(t) = a/r = a/sqrt(a^2+b^2) = 1/sqrt(1+(b/a)^2)
    //   cos(t) = 1/sqrt(1+tan^2(t))
    //..................................................................
    //   x' = 1/sqrt{1+tan^2(a)} {x        -y*tan(a)}
    //   y' = 1/sqrt{1+tan^2(a)} {x*tan(a)  y       }
    //..................................................................
    //   tan(t) = 1/2^i
    //   Ki = 1 / sqrt(1 + 2^{-2i}) ==> K = Prod {K_i}
    //..................................................................
```

```
//   x' = Ki {x         -y*2^{-2i}}  ==> x" = {x         -y*2^{-2i}}
//   y' = Ki {x*2^{-2i}  y         }  ==> y" = {x*2^{-2i}  y         }
//....................................................................
//   x" = {x         -y*2^{-2i}}
//   y" = {x*2^{-2i}  y         }
//   a" = a - atan{1/2^i}
//....................................................................
factor = sigma * poweroftwo;             // +2^(j-1) / -2^(j-1)

*x =          xn - factor * yn;          // x" = {x         -y*2^{-2i}}
*y = factor * xn +          yn;          // y" = {x*2^{-2i}  y         }

xn = *x;
yn = *y;

//....................................................................
//  Update the remaining angle.
//....................................................................
theta = theta - sigma * angle;           // a" = a - atan{1/2^i}

*z = theta;

//....................................................................
// Thresholding:
// If residual angle is less than a given threshold, then break
//....................................................................
if (C->useTh) {
  static int cntBreak = 0;
  if (C->nBreakInit == 0)         // nBreakInit==0 : init Break counter
    cntBreak = 0;
  if (fabs(*z) < C->threshold) {  // residue less than Th
    C->nBreak = ++cntBreak;       // inc Break counter

    if (C->useThDisp) {           // if useThDisp, display its statistics
      cout << "cntBreak= " << cntBreak;
      cout << "  z= " << right << setw(15) << *z;
      cout << "  < "  << right << setw(7)  << C->threshold;
      cout << "  j= " << right << setw(4)  << j << endl;
    }
    break;
  }
}


//....................................................................
//  Update the angle from table, or eventually by just dividing by two.
//....................................................................
poweroftwo = poweroftwo / 2.0;   // 1/2^j

if (C->useATAN)                  // if useATAN, then use arctangent
```

```
        angle = atan( 1. / (1 << j));  // atan(1/2^i)
      else                             // otherwise, use angles array values
        if ( ANGLES_LENGTH < j+1 )  angle = angle / 2.0;
        else                        angle = (C->angles)[j];


  //-----------------------------------------------------------------------
  }  /* end of j */
  //-----------------------------------------------------------------------



  //-----------------------------------------------------------------------
  //  Adjust length of output vector to be [cos(beta), sin(beta)]
  //
  //  KPROD is essentially constant after a certain point, so if N is
  //  large, just take the last available value.
  //-----------------------------------------------------------------------
  if ( j > KPROD_LENGTH ) {
    *x = *x * (C->kprod) [ KPROD_LENGTH - 1 ];     // K = 1.647 limit val
    *y = *y * (C->kprod) [ KPROD_LENGTH - 1 ];     // K = 1.647 limit val
  }
  else {
    *x = *x * (C->kprod) [ j - 1 ];                // K = Prod(Ki)
    *y = *y * (C->kprod) [ j - 1 ];                // K = Prod(Ki)
  }

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

  return;

}
```

```
::::::::::::::
Core.1.fptr2.cordic_burk.cpp
::::::::::::::
```

```
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>


#include "Core.hpp"

using namespace std;

#include "cordic_burkardt.hpp"


//----------------------------------------------------------------------------
//   Purpose:
//
//     stand alone cordic_burk() implementation file
//     friend function of Core class
//
//     [Core.1.fptr2.cordic_burk.cpp]
//
//  Discussion:
//
//
//  CORDIC returns the sine and cosine using the CORDIC method.
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.15
//
//  Author:
//
//    Based on MATLAB code in a Wikipedia article.
//    Modifications by John Burkardt
//    Further modified by Young W. Lim
//
//  Parameters:
//
//    Input:
//      *x: x coord of an init vector
//      *y: y coord of an init vector
//      *z: angle (-90 <= angle <= +90)
//
//      level : number of iteration
//              A value of 10 is low.  Good accuracy is achieved
```

```
//              with 20 or more iterations.
//
//    Output:
//      *xo: x coord of a final vector
//      *yo: y coord of a final vector
//      *zo: angle residue
//
//  Local Parameters:
//
//    Local, real ANGLES(60) = arctan ( (1/2)^(0:59) );
//
//    Local, real KPROD(33), KPROD(j) = product ( 0 <= i <= j ) K(i),
//    K(i) = 1 / sqrt ( 1 + (1/2)^(2i) ).
//
//--------------------------------------------------------------------------
void cordic_burk( double *x, double *y, double *z, Core *C )
{

  using namespace burkardt;

  // using cossin_cordic routine in the file "cordic_burkardt.cpp"
  // void cossin_cordic ( double beta, int n, double *c, double *s );
  //
  // setLevel() is requried
  //
  // See http://people.sc.fsu.edu/~jburkardt/cpp_src/cordic/cordic.html
  //

  cossin_cordic(*z, C->level, x, y);

  return;

}
```

```
::::::::::::::
Core.1.fptr3.cordic_vhdl.cpp
::::::::::::::
#include "Core.hpp"


using namespace std;
```

```
extern "C" {
  void  cordic_ghdl( double *x, double *y, double *z) ;
}


//----------------------------------------------------------------------------
//   Purpose:
//
//     stand alone cordic_vhdl() implementation file
//     friend function of Core class
//
//     [Core.1.fptr3.cordic_vhdl.cpp]
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.03.15
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//----------------------------------------------------------------------------
void cordic_vhdl ( double *x, double *y, double *z, Core *C) {


   cordic_ghdl ( x, y, z );


}
```

```
::::::::::::::
Core.2.wrap1.cordic_stat.cpp
```

```
::::::::::::::
#include "Core.hpp"

//------------------------------------------------------------------------------
//   Purpose:
//
//     Class Core Implementation Files
//     Core::cordic_stat()
//
//     [Core.2.wrap1.cordic_stat.cpp]
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.15
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//    ref var    cnt=0 initialize statistics accumulators
//    ref var    xx = (*x - cosz)
//    ref var    yy = (*y - sinz)
//    ref var    zz = (*z)

//------------------------------------------------------------------------------
void Core::cordic_stat (double *x, double *y, double *z, int& cnt, int& xx, int& yy, int& zz)
{

    double cosz, sinz;


    if (cnt == 0) {
        setNBreak(nBreak=0);
        setNBreakInit(nBreakInit=0);
        initAcc();
        cnt++;
        sSCE   =   sSSE =   sSRE = 0.0;
        minSCE = minSSE = minSRE = +1.0e+10;
        maxSCE = maxSSE = maxSRE = -1.0e+10;
    }
```

```
    cosz = cos(*z);
    sinz = sin(*z);

    setNBreakInit(nBreakInit++);
    //.........................................................
    cordic(x, y, z);
    //.........................................................

    xx = (*x - cosz);
    yy = (*y - sinz);
    zz = (*z);

      SCE  = xx * xx;       SSE  = yy * yy;       SRE  = zz * zz;
     sSCE += SCE;          sSSE += SSE;          sSRE += SRE;
     mSCE  = sSCE/cnt;     mSSE  = sSSE/cnt;     mSRE  = sSRE/cnt;
    rmSCE  = sqrt(mSCE);  rmSSE  = sqrt(mSSE);  rmSRE  = sqrt(mSRE);

    minSCE = (minSCE > SCE) ? SCE : minSCE;
    minSSE = (minSSE > SSE) ? SSE : minSSE;
    minSRE = (minSRE > SRE) ? SRE : minSRE;

    maxSCE = (maxSCE < SCE) ? SCE : maxSCE;
    maxSSE = (maxSSE < SSE) ? SSE : maxSSE;
    maxSRE = (maxSRE < SRE) ? SRE : maxSRE;

}
```

```
::::::::::::::
Core.2.wrap2.cordic_break.cpp
::::::::::::::
#include "Core.hpp"


using namespace std;


//----------------------------------------------------------------------------
//   Purpose:
//
//     Class Core Implementation Files
//     Core::cordic_break()
//
//     [Core.2.wrap2.cordic_break.cpp]
```

```
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.15
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//    ref var init=0 initializes statistics accumulators
//
//----------------------------------------------------------------------------
void Core::cordic_break ( double *x, double *y, double *z, int& init)
{

    double cosz, sinz;


    if (init == 0) {
        setNBreak(nBreak=0);
        setNBreakInit(nBreakInit=0);
        initAcc();
        init++;
    }

    cosz = cos(*z);
    sinz = sin(*z);

    setNBreakInit(nBreakInit++);
    //...........................................................
    cordic(x, y, z);
    //...........................................................

    xx = (*x - cosz);
    yy = (*y - sinz);

    sum_xx += xx; sum_xx2 += (xx*xx);
    sum_yy += yy; sum_yy2 += (yy*yy);

    if (max_err < fabs(xx)) max_err = fabs(xx);
    if (max_err < fabs(yy)) max_err = fabs(yy);
```

```
    if (fabs(cosz) > 1.0e-10) {
        if (max_errn < fabs(xx/cosz))
            max_errn = fabs(xx/cosz);
        sum_xx_n += xx/cosz;
        sum_xx2_n += (xx*xx)/(cosz*cosz);
        cnt_xx++;
    }
    if (fabs(sinz) > 1.0e-10) {
        if (max_errn < fabs(yy/sinz))
            max_errn = fabs(yy/sinz);
        sum_yy_n += yy/sinz;
        sum_yy2_n += (yy*yy)/(sinz*sinz);
        cnt_yy++;
    }

}


::::::::::::::
Figures.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 =                    \

INCS =                     \

.SUFFIXES : .o .cpp .c

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

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


#---------------------------------------------------------------
# Classes
#---------------------------------------------------------------
SRC = Figures.cpp                        \
      Figures.hpp                        \
```

```
OBJ = Figures.o                          \

INC = Figures.hpp                        \

LIB = libcordic-figures.a                \

EXE =                                    \


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


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

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

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

clean :
	\rm -f *.o *~ *#
	\rm -f *.print *.tar *.a


::::::::::::::
Figures.cpp
::::::::::::::
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <cmath>
# include <fstream>
# include <vector>
# include <algorithm>
# include <cstring>
# include <string>

# include "Figures.hpp"


using namespace std;
```

```
//------------------------------------------------------------------------------
//   Purpose:
//
//     Figures Class Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.05.06
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------------
//
// Figures::Figures()
// Figures::include_fig_file(ofstream& myout)
// Figures::write_latex_file(int mode)
// Figures::merge_figures(list<string>& epsL1, list<string>& epsL2)
// Figures::make_figures(int mode, list<string>& epsL1, list<string>& epsL2)
//------------------------------------------------------------------------------



//------------------------------------------------------------------------------
//  Class Figures' Member Functions
//------------------------------------------------------------------------------
Figures::Figures()
{
   epsList.clear();

}



//----------------------------------------------------------------------------
// include basic figure eps files
//----------------------------------------------------------------------------
void Figures::include_fig_file(ofstream& myout) {
```

```
  list<string>::iterator I;
  int count=0;


  myout << "\\begin{lstlisting}" << endl;
  for (I = epsList.begin(); I != epsList.end(); ++I) {
    myout << *I << endl;
  }
  myout << "\\end{lstlisting}" << endl;


  myout << "\\newpage" << endl;

  myout << "\\begin{figure}[h!]" << endl;
  myout << "\\begin{center}" << endl;
  for (I = epsList.begin(); I != epsList.end(); ++I) {
    // myout << "\\includegraphics[scale=0.5]{./";
    myout << "\\includegraphics[width=0.48\\textwidth]{./";
    myout << *I << "}" << endl;

    count++;
    // if (count == epsList.size()/2) {
    if (count %8  == 0) {
      myout << "\\end{center}" << endl;
      myout << "\\end{figure}" << endl;
      myout << "\\newpage" << endl;
      myout << "\\begin{figure}[h!]" << endl;
      myout << "\\begin{center}" << endl;
    }

  }

  myout << "\\end{center}" << endl;
  myout << "\\end{figure}" << endl;

}


//-----------------------------------------------------------------------------
// Making tex output files
//-----------------------------------------------------------------------------
void Figures::write_latex_file() {
  ofstream myout;



  // writing gnuplot commands
  myout.open(fname);
```

```
myout << "\\documentclass[12pt]{article}" << endl;
myout << "" << endl;
myout << "\\usepackage{amsmath}    % need for subequations" << endl;
myout << "\\usepackage{graphicx}   % need for figures" << endl;
myout << "\\usepackage{verbatim}   % useful for program listings" << endl;
myout << "\\usepackage{color}      % use if color is used in text" << endl;
myout << "%\\usepackage{subfloat}  % use for side-by-side figures" << endl;
myout << "%\\usepackage{hyperref}   % use for hypertext links" << endl;

myout << "\\usepackage{listings}"  << endl;

myout << "" << endl;
myout << "" << endl;
myout << "\\setlength{\\baselineskip}{16.0pt}    % 16 pt usual spacing between lines" << endl;
myout << "" << endl;
myout << "%\\setlength{\\parskip}{3pt plus 2pt}" << endl;
myout << "%\\setlength{\\parindent}{20pt}" << endl;
myout << "%\\setlength{\\oddsidemargin}{0.5cm}" << endl;
myout << "%\\setlength{\\evensidemargin}{0.5cm}" << endl;
myout << "%\\setlength{\\marginparsep}{0.75cm}" << endl;
myout << "%\\setlength{\\marginparwidth}{2.5cm}" << endl;
myout << "%\\setlength{\\marginparpush}{1.0cm}" << endl;
myout << "%\\setlength{\\textwidth}{150mm}" << endl;
myout << "\\addtolength{\\oddsidemargin}{-2.5cm}" << endl;
myout << "\\addtolength{\\evensidemargin}{-2.5cm}" << endl;
myout << "\\addtolength{\\marginparwidth}{-3.0cm}" << endl;
myout << "\\addtolength{\\textwidth}{+4.0cm}" << endl;
myout << "" << endl;
myout << "\\begin{document}" << endl;
myout << "" << endl;
myout << "\\begin{center}" << endl;
myout << "{\\large " << title << "} \\\\" << endl;
myout << "\\today" << endl;
myout << "\\end{center}" << endl;
myout << "" << endl;
myout << "" << endl;
myout << "" << endl;

//..................................
include_fig_file(myout);
//..................................

myout << "" << endl;
myout << "" << endl;
myout << "" << endl;
myout << "\\end{document}" << endl;

myout.close();
```

```
}


//-----------------------------------------------------------------------------
// Merge figure lists (Leafnodes and Allnodes list)
//-----------------------------------------------------------------------------
void Figures::merge_figures(list<string>& epsL1, list<string>& epsL2) {

  list<string>::iterator I;

  epsList.clear();

  for (I = epsL1.begin(); I != epsL1.end(); ++I) {
    epsList.push_back(* I);
    cout << * I << endl;
  }

  for (I = epsL2.begin(); I != epsL2.end(); ++I) {
    epsList.push_back(* I);
    cout << * I << endl;
  }

  for (I = epsList.begin(); I != epsList.end(); ++I) {
    cout << * I << endl;
  }

}


//-----------------------------------------------------------------------------
// Determine the kinds of tex output files to be written
//-----------------------------------------------------------------------------
void Figures::make_figures(int mode, list<string>& epsL1, list<string>& epsL2) {

   //..................................
   merge_figures(epsL1, epsL2);
   //..................................


   cout << "mode= " << mode << endl;

   //-------------------------------------------------------
   if (mode & 1) {

     strcpy(fname, "fig_basic.tex");
     strcpy(title, "Basic Figures");

     //................................
     write_latex_file();
     //................................
```

```
    //................................
    system("latex fig_basic.tex");
    //................................
    cout << "end of latex \n";

    //................................
    system("dvipdf fig_basic.dvi");
    //................................
    cout << "end of dvipdf \n";
  }


  //----------------------------------------------------------
  if (mode & 2) {
    strcpy(fname, "fig_tscale.tex");
    strcpy(title, "TScale Figures");

    //................................
    write_latex_file();
    //................................

    //................................
    system("latex fig_tscale.tex");
    //................................
    cout << "end of latex \n";

    //................................
    system("dvipdf fig_tscale.dvi");
    //................................
    cout << "end of dvipdf \n";
  }


  //----------------------------------------------------------
  if (mode & 4) {
    strcpy(fname, "fig_uscale.tex");
    strcpy(title, "UScale Figures");

    //................................
    write_latex_file();
    //................................

    //................................
    system("latex fig_uscale.tex");
    //................................
    cout << "end of latex \n";

    //................................
    system("dvipdf fig_uscale.dvi");
```

```
    //.................................
    cout << "end of dvipdf \n";
  }

}

::::::::::::::
Figures.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 Figures Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.05.06
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//    instantiated in
//      main() in Angles_tb.cpp
//
//------------------------------------------------------------------------------
```

```
class Figures
{

  public:

  Figures();

  char fname[200];
  char title[200];

  void include_fig_file(ofstream& myout);
  void write_latex_file();

  void merge_figures(list<string>& epsL1, list<string>& epsL2);
  void make_figures(int mode, list<string>& epsL1, list<string>& epsL2);

  list<string> epsList;

};



::::::::::::::
GPData.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.Angles

INCS = -I../Class.Angles    \

.SUFFIXES : .o .cpp .c

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

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


#----------------------------------------------------------------
# Classes
```

```
#---------------------------------------------------------------
SRC = GPData.cpp                        \
      GPData.hpp                        \

OBJ = GPData.o                          \

INC = GPData.hpp                        \

LIB = libcordic-gpdata.a                \

EXE =                                   \


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


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

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

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

clean :
        \rm -f *.o *~ *#
        \rm -f *.print *.tar *.a

::::::::::::::
GPData.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:
//
//    GnuPlot Data Class Implementation
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.05.12
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//      ..................................................
//      prefix = LorAStr + lStr
//      suffix = nStr + xStr + thStr
//      fname  = preStr + prefix + inStr + suffix . fext
//      title  = inStr + prefix + suffix
//      ..................................................
//      prefix : "Leaf_10" / "All_10"
//      suffix : "n4096x1th0.55"
//      fname  : "egbX.____.circle_ang.____.eps"
//      title  : "Angle Tree (Leaf_10 n4096x1th0.55)"
//      ..................................................
//
//------------------------------------------------------------------------------
//
//  void GPData::set_prefix(Angles *Ang);
//  void GPData::set_suffix(Angles *Ang);
//  void GPData::set_fname(Angles *Ang, const char *preStr, const char *inStr);
//  void GPData::set_title(Angles *Ang, const char *inStr);
//------------------------------------------------------------------------------




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

```
//  Class GPData Member Functions
//----------------------------------------------------------------------------
GPData::GPData()
{

  strcpy        (GnuTerm,       "");

  strcpy        (fname,         "");
  strcpy        (fext,          "");

  strcpy        (title,         "");
  strcpy        (xlabel,        "");
  strcpy        (ylabel,        "");

  strcpy        (LorAStr,       "");
  strcpy        (lStr,          "");
  strcpy        (nStr,          "");
  strcpy        (xStr,          "");
  strcpy        (thStr,         "");

  strcpy        (udata,         "");

  strcpy        (prefix,        "");
  strcpy        (suffix,        "");

  nPoints       = 0;

  useSubRange   = 0;
  valSubRange   = 0;
}


GPData::GPData(string GTerm, int nPt)
{
  GPData();

  strcpy (GnuTerm, GTerm.c_str());
  strcpy (fext,    GTerm.c_str());
  check_fext();

  nPoints = nPt;
}


GPData::GPData(string GTerm)
{

  GPData();

  strcpy (GnuTerm, GTerm.c_str());
```

```
  strcpy (fext,    GTerm.c_str());
  check_fext();
}


void GPData::check_fext()
{
  if (strcmp(fext, "postscript") == 0)
    strcpy(fext, "eps");
}

//-------------------------------------------------------------------------------
// prefix = LorAStr + lStr
// prefix : "Leaf_10" / "All_10"
//-------------------------------------------------------------------------------
// set LorAStr : string "Leaf" or "All"
// set lStr    : string for the level
//-------------------------------------------------------------------------------
void GPData::set_prefix(Angles *Ang)
{
  (Ang->getLeaf()) ? strcpy(LorAStr, "Leaf_") : strcpy(LorAStr, "All_");
  sprintf(lStr,"%d", Ang->getnIters());

  strcpy(prefix, LorAStr);
  strcat(prefix, lStr);
}


//-------------------------------------------------------------------------------
// suffix = nStr + xStr + thStr : "n4096x1th0.55"
//-------------------------------------------------------------------------------
// set nStr    : string "Leaf" or "All"
// set xStr    : string for the sampling factor
// set thStr   : string for the threshold used
//-------------------------------------------------------------------------------
void GPData::set_suffix(Angles *Ang)
{
  sprintf(nStr, "n%d", nPoints);
  // sprintf(xStr, " ");
  // sprintf(thStr, " ");
  *xStr = 0;
  *thStr = 0;

  if (useSubRange)  sprintf(xStr, "x%d", valSubRange) ;
  if (Ang->getUseTh()) sprintf(thStr, "th%g", Ang->getThreshold()) ;

  sprintf(suffix, nStr, xStr, thStr);
}
```

```
//-----------------------------------------------------------------------------
// fname  = preStr + prefix + inStr + suffix . fext
// fname  : "egbX.____.circle_ang.____.eps"
//-----------------------------------------------------------------------------
// preStr  : "egb[1,2,3,4]", "egt[2,3]", "egu[2,3,4]"
// prefix  : "Leaf_10" / "All_10"
// inStr   :
//            ...ang_tree[1,2,3]
//               circle_ang
//               line_ang
//               quantization
//            ...delta_dist_bin
//               delta_dist_val
//               delta_vs_angle
//               res[0-7]_vs_angle
//               res[0-7]_vs_index
//            ...angle_vs_dff
//               angle_vs_res
//               dff_hist
//               res_hist
//               angle_vs_dff
//               angle_vs_res
//               res[0-7]_vs_angle_rnd
//               res[0-7]_vs_index_rnd
//               corr_dff_vs_angle
//               corr_res_vs_angle
//               dff_vs_angle
//               res_vs_angle
// suffix  : "n4096x1th0.55"
// fext    : "eps"
//-----------------------------------------------------------------------------
void GPData::set_fname(Angles *Ang, const char *preStr, const char *inStr)
{
  if (strcmp(GnuTerm, "wxt") == 0) {
    strcpy(fname, "");
    strcpy(fext, "");
    return;
  }

  // GnuTerm is set by the constructors : GTerm.c_str()
  strcpy(fext, GnuTerm);
  check_fext();

  sprintf(fname, "%s.%s.%s.%s.%s", preStr, prefix, inStr, suffix, fext);
}


//-----------------------------------------------------------------------------
// title = inStr + prefix + suffix
// title : "Angle Tree (Leaf_10 n4096x1th0.55)"
```

```
//------------------------------------------------------------------------------
// inStr   :
//            ...ang_tree[1,2,3]
//               circle_ang
//               line_ang
//               quantization
//            ...delta_dist_bin
//               delta_dist_val
//               delta_vs_angle
//               res[0-7]_vs_angle
//               res[0-7]_vs_index
//            ...angle_vs_dff
//               angle_vs_res
//               dff_hist
//               res_hist
//               angle_vs_dff
//               angle_vs_res
//               res[0-7]_vs_angle_rnd
//               res[0-7]_vs_index_rnd
//               corr_dff_vs_angle
//               corr_res_vs_angle
//               dff_vs_angle
//               res_vs_angle
// prefix  : "Leaf_10" / "All_10"
// suffix  : "n4096x1th0.55"
//------------------------------------------------------------------------------
void GPData::set_title(Angles *Ang, const char *inStr)
{
  sprintf(title, "%s (%s %s)", inStr, prefix, suffix);
}


//------------------------------------------------------------------------------
// xlabel :
//------------------------------------------------------------------------------
void GPData::set_xlabel(const char *inStr)
{
  sprintf(xlabel, inStr);
}


//------------------------------------------------------------------------------
// ylabel :
//------------------------------------------------------------------------------
void GPData::set_ylabel(const char *inStr)
{
  sprintf(ylabel, inStr);
}
```

```
::::::::::::::
GPData.hpp
::::::::::::::
using namespace std;


#define useXSampling    10;
#define useXPartition   20;
#define useXSubtree     30;

//----------------------------------------------------------------------------
//   Purpose:
//
//     Class GPData Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.05.12
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//
//     instantiated in
//       (o) Angles.1.b1.plot_angle_tree.cpp
//       (o) Angles.1.b2.plot_circle_angle.cpp
//       (o) Angles.1.b3.plot_line_angle.cpp
//       (o) Angles.1.b4.plot_quantization.cpp
//       (X)    Angles.2.t1.calc_tscale_statistics.cpp
//       (o) Angles.2.t2.plot_tscale_statistics.cpp
//       (o) Angles.2.t3.plot_tscale_residual_angles.cpp
//       (X)    Angles.3.u1.calc_uscale_statistics.cpp
//       (o) Angles.3.u2.plot_uscale_statistics.cpp
//       (o) Angles.3.u3.plot_uscale_residual_angles.cpp
//       (o) Angles.3.u4.plot_uscale_histogram.cpp
//       (X)    Angles.a.compute_angle_arrays.cpp
//
//       /* 1b1 */ void   plot_angle_tree              (int, int);
```

```
//       /* 1b2 */ void   plot_circle_angle            ();
//       /* 1b3 */ void   plot_line_angle              ();
//       /* 1b4 */ void   plot_quantization            ();
//       /* 2t2 */ void   plot_tscale_statistics       (int);
//       /* 2t3 */ void   plot_tscale_residual_angles  ();
//       /* 3u2 */ void   plot_uscale_statistics       (int);
//       /* 2u3 */ void   plot_uscale_residual_angles  (int);
//       /* 3u4 */ void   plot_uscale_histogram        (int);
//
//----------------------------------------------------------------------------

#define GPSLEN 255

//----------------------------------------------------------------------------
// Data structure for gnuplot call
//----------------------------------------------------------------------------

class GPData {
  public:
  GPData();
  GPData(string GTerm);
  GPData(string GTerm, int nPt);



  void set_prefix(Angles *Ang);
  void set_suffix(Angles *Ang);
  void set_fname(Angles *Ang, const char *preStr, const char *inStr);
  void set_title(Angles *Ang, const char *inStr);
  void set_xlabel(const char *inStr);
  void set_ylabel(const char *inStr);
  void check_fext();

  //--------------------------------------------------------------------
  char GnuTerm  [GPSLEN];  // wxt, eps, emf ...

  char prefix   [GPSLEN];  // eg. Leaf_10, All_10
  char suffix   [GPSLEN];  // eg. n4192x1th1.5 ...

  char fname    [GPSLEN];  // output file name
  char fext     [GPSLEN];  // output file extension ie eps, emf, ...

  char title    [GPSLEN];  // GNU plot title
  char xlabel   [GPSLEN];  // GNU plot x label
  char ylabel   [GPSLEN];  // GNU plot y label

  char udata    [GPSLEN];  // used columns

  char LorAStr  [GPSLEN];  // string "Leaf" or "All"
  char lStr     [GPSLEN];  // string for the level
```

```
  char nStr     [GPSLEN];  // string for the number of nodes
  char xStr     [GPSLEN];  // string for the sampling factor
  char thStr    [GPSLEN];  // string for the threshold used
  //----------------------------------------------------------------------

  int  nPoints;            // no of GNU plot data points
  int  nSamples;           // no of sample points for a GNU plot
  int  nParts;             // no of partitions (groups of sample points)

  //--------------------------------
  // useSubRnage :      useXSampling    10;
  // useSubRnage :      useXPartition   20;
  // useSubRnage :      useXSubtree     30;
  //--------------------------------
  int  useSubRange;
  int  valSubRange;

};
```