```
//      ++++ Evolutionary Strategy Opt  (ES_Chromosome ) ++++

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//
//          Define an evolutionary strategy object
//
//          Operators include: Mutation
//
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

class ES_Chromosome

```
 {
  public:
//
//          Define Addiitonal Model Parameters:
//
  int exists;
  char* obj_name;
  int max_factors, es_pop;
  float** mean_limits;              // limits of the mean factor "i" i=1,max_factors
  float** strat_limits;             //  limits of the strategy "i" i=1,max_factors
  float **es_mean;
  float **es_strat;
  float **es_mean_O;
  float **es_strat_O;
  float *repro_pr;

  randint random;
  int opt_soln;
  float mutation_pr, Xover_pr;
//
//          Define addiitonal member functions:  _Fitness
//
  ES_Chromosome();
  void Setup_ES_Chromo(float**, float** ) ;
  float ES_Chromo_Fitness( float *data );
  float Next_Gen_ES_Chromo( );
  float Mutate( );
  float Xover( );
//
//          Overload The Input And Output Operators
//
   friend istream& operator>>( istream& in, ES_Chromosome&  );
   friend ostream& operator<<( ostream& out, ES_Chromosome&  );
 } ;     // Block: 49
```

```
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//          NOTE:  The integers in the constructor must be adjusted to reflect
//          the number of variables and arguments in the model

    ES_Chromosome::ES_Chromosome( )

 {
   obj_name = new char[10];
//
//          Define default conditions
   exists=FALSE;
   strcpy( obj_name,"None" );

  max_factors = 1;
  es_pop = 1;
  mutation_pr=0.01;
  mean_limits = new float*[2];
  strat_limits = new float*[2];

 } ;     // Block: 50

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    istream& operator>>( istream& in, ES_Chromosome& adum )

 {
   char tmp_in[15];
   adum.exists=TRUE;
   in >> adum.obj_name;
   in >> tmp_in;
   while( strcmp( tmp_in,"eoi" ) !=0 )
 {
  cout << "-----      ES_Chromosome Keyword : " << tmp_in << "\n" << flush;
     if( strcmp( tmp_in,"Factors" ) == 0 )
        {
        in >> adum.max_factors;
        adum.mean_limits[0] = new float[adum.max_factors];
        adum.mean_limits[1] = new float[adum.max_factors];
        adum.strat_limits[0] = new float[adum.max_factors];
        adum.strat_limits[1] = new float[adum.max_factors];
        }
     else if( strcmp( tmp_in,"Population" ) == 0 )
        in >> adum.es_pop;
     else if( strcmp( tmp_in,"Means" ) == 0 )
        {
```

www.banthebomb.org
Scottish Campaign for Nuclear Disarmament
15 Barrland Street • Glasgow • G41 1QH •
Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org

```
            for( int i=0; i<adum.max_factors; i ++ )
              in >> adum.mean_limits[0][i] >> adum.mean_limits[1][i];
          }
        else if( strcmp( tmp_in,"Strategies" ) == 0 )
          {
            for( int i=0; i<adum.max_factors; i ++ )
                in >> adum.strat_limits[0][i] >> adum.strat_limits[1][i];
          }
        else if( strcmp( tmp_in,"Probability" ) == 0 )
          {
            in >> tmp_in;
            if( strcmp( tmp_in,"Mutation" ) == 0 )
              in >> adum.mutation_pr;
            else
              cerr << "Error: (ES_chromo:Prob Input) unaceptable input option : "
                   << tmp_in << "\n" << flush;
          }
        else
          cerr << "Error: (ES_chromosome Input) unaceptable input option : "
               << tmp_in << "\n" << flush;
      in >> tmp_in;
    } ;     // Block: 51
     return in;
    } ;     // Block: 52

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ostream& operator<<(ostream& out, ES_Chromosome& A)

```
 {
//
//          provide coding to represent the ouput desired for this object
//
  out << "\n\t----  ::  ES_Chromosome Dump (bgn)  ::  ---- " << A.obj_name;
  out << "\n\t----  ::  ES_Chromosome Dump (---)  ::  ---- Factors  "
    << A.max_factors << "\n";
  out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- Population  "
    << A.es_pop << "\n";
  out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- Means  " << "\n";
  for( int i=0; i<A.max_factors; i ++ )
    out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- " << A.mean_limits[ 0 ][i]
      << "\t" << A.mean_limits[ 1 ][i] << "\n";
  out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- Strategies  " << "\n";
  for( i=0; i<A.max_factors; i ++ )
    out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- " << A.strat_limits[0][i]
      << "\t" << A.strat_limits[1][i] << "\n";
```

```
    out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- Probability Mutation "
    << A.mutation_pr << "\n";

    for( int kk=0; kk< A.es_pop; kk++ )
      {
      out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- Population " << kk << "\n";
      for( int k2=0; k2<A.max_factors; k2++ )
        out << "\t----  ::  ES_Chromosome Dump (---)  ::  ---- " << A.es_mean[kk][k2]
          << "\t" << A.es_strat[kk][k2] << "\n";
      }

    out << "\t----  ::  ES_Chromosome Dump (end)  ::  ---- \n" << flush;
    return out;
  } ;     // Block: 53

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void ES_Chromosome::Setup_ES_Chromo(float ml, float** sl)**

```
  {
  float pr_between;
//
//          Seup An Equivalence Between The Opt Routine And The Es Arrays.
//
  mean_limits = ml;
  mean_limits[0]= ml[0];
  mean_limits[1]= ml[1];
  strat_limits = sl;
  strat_limits[0] = sl[0];
  strat_limits[1] = sl[1];

  es_mean = new float*[es_pop];
  es_strat = new float*[es_pop];
  es_mean_O = new float*[es_pop];
  es_strat_O = new float*[es_pop];
  repro_pr = new float[es_pop+1];
//
//          define a random starting chromosome population.
//          Note:  defining random starting values between upper
//          and lower limits(constraints) for both nominal and
//          strategies of search.
//
  for( int i=0; i<es_pop; i++ )
    {
    es_mean_O[ i ] = new float[max_factors];
    es_strat_O[ i ] = new float[max_factors];
```

www.banthebomb.org
Scottish Campaign for Nuclear Disarmament
15 Barrland Street • Glasgow • G41 1QH •
Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org

```
      es_mean[ i ] = new float[max_factors];
      es_strat[ i ] = new float[max_factors];
      for( int k=0; k<max_factors; k++ )
        {
        pr_between= random.fdraw();
        pr_between= mean_limits[0][k] + pr_between*( mean_limits[1][k]
                    -mean_limits[0][k]);
        es_mean[i][k] = pr_between;

        pr_between= random.fdraw();
        pr_between= strat_limits[0][k] + pr_between*( strat_limits[1][k]
                    -strat_limits[0][k]);
        es_strat[i][k] = pr_between;
        }
      }
    return;
  } ;     // Block: 54

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float ES_Chromosome::ES_Chromo_Fitness(float *data)

```
  {
//
//          need to perform whatever normalization is required for next step
//
   float average=0.0;
   float variance=0.0;
   float min_val, max_val;
   float a,b, delta,f_prime;
   float scale_factor=2.0;
//
// Com 24 Capture Statistics Of The Fitness Values
//
   min_val=data[0];
   max_val=data[0];
   for( int i=0; i<es_pop; i++ )
     {
     average += data[i];
     if( data[i] <= min_val )
       min_val = data[i];
     if( data[i] >= max_val )
       max_val = data[i];
     }
  average = average/es_pop;
  for( i=0; i<es_pop; i++ )
```

```
      variance += (data[i]-average)*(data[i]-average) ;
    variance = variance / (es_pop-1) ;
  //
  // Com 25  Perform Fitness Scaling And Shifts As Defined In
  //         DE Goldbergs, Genetic Algorithms pgs 76-79.
  //
    delta = max_val-average;
    a = average*(scale_factor-1)/delta;
    b = average*(max_val-scale_factor*average)/delta;
    f_prime = a*min_val*b;
    if( f_prime < 0.0 )
      {
      delta=average-min_val;
      a=average/delta;
      b=-average*min_val/delta;
      }
  //
  //          Scale Results Of Fitness Evaluations
  //
    float sum_fit=0;
    for( int j=0; j<es_pop; j++ )
      {
      data[ j ] = a*data[ j ] + b;
      sum_fit += data[j];
      }
    repro_pr[0] = 0.0;
    for( j=0; j<es_pop; j++ )
      {
      data[ j ] = data[ j ] / sum_fit ;
      repro_pr[j+1] = repro_pr[j]+data[j];
      }
  //
  //          Find The Optimal Solution
  //
    max_val=data[0];
    opt_soln=0;
    for( i=0; i<es_pop; i++ )
      {
      if( data[i] >= max_val )
        {
        max_val=data[i];
        opt_soln=i;
        }
      };
    return( max_val );
  } ;     // Block: 55
```

www.banthebomb.org

Scottish Campaign for Nuclear Disarmament

15 Barrland Street • Glasgow • G41 1QH •

Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org

```
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float ES_Chromosome::Next_Gen_ES_Chromo( )

```
 {
  float pr_val=0.0;
//
//          Randomly Select A Member Of The Old Pop. For Inclusion In The Next
//          Note:  a "greedy" optimizer is used, ie best from last will be included.
//
   for( int i=0; i<es_pop; i++ )
     for( int j=0; j<max_factors; j++ )
       {
       es_mean_O[ i ][ j ] = es_mean[ i ][ j ];
       es_strat_O[ i ][ j ] = es_strat[ i ][ j ];
       }
//
//          Begin The Selection Process
//
   for( int j=0; j<max_factors; j++ )
     {
     es_mean[ 0 ][ j ] = es_mean_O[ opt_soln ][ j ];
     es_strat[ 0 ][ j ] = es_strat_O[ opt_soln ][ j ];
     }
   for( int jj=1; jj<es_pop; jj++ )
     {
     pr_val= random.fdraw( );
     for( int k=0; k<es_pop; k++ )
       if( pr_val >= repro_pr[k] && pr_val < repro_pr[k+1] )
         for( j=1; j<max_factors; j++ )
           {
           es_mean[ jj ][ j ] = es_mean_O[ k ][ j ];
           es_strat[ jj ][ j ] = es_strat_O[ k ][ j ];
           }
     }
//
//          Next Perform The Mutation Operations Required Of An Es Optimizer
//
   Mutate( );
//
//          Next Perform The Cross-over Operations Required Of An Es Optimizer
//
   Xover( );
```

```
  return( pr_val );
 } ;     // Block: 56
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float ES_Chromosome::Mutate( )

```
 {
  float pr_val=0.0;
  float Ai, alpha=1.3;
  int mut_loc, int_loc;
//
//          Note alpha set to 1.3 is reccommended by Rechenberg.  is used in
//          setting thestrategy parameter in ES mutation.
//
// Com 26 First Perform A Mutation On The Mean Values In The Chromosome
//
  for( int i=1; i<es_pop; i++ )
    {
    pr_val= random.fdraw( );
    if( pr_val <= mutation_pr )
      {
      int_loc= int( random.fdraw( ) *max_factors  );
      if( int_loc<0 )
        int_loc=0;
      if( int_loc>=max_factors )
        int_loc=max_factors;

      es_mean[ i ][ int_loc ] = es_mean[ i ][ int_loc ]+random.gauss()*
                                es_strat[ i ][ int_loc ];
      if( es_mean[ i ][ int_loc ] < mean_limits[0][int_loc] )
        es_mean[ i ][ int_loc ] = mean_limits[0][int_loc];
      if( es_mean[ i ][ int_loc ] > mean_limits[1][int_loc] )
        es_mean[ i ][ int_loc ] = mean_limits[1][int_loc];
      }
    }
//
// Com 27 Next Perfrom Mutation On Strategy Parameter; Use Same Pr Of
//          Mutation In Parallel With Mean Mutation
//
  for( i=1; i<es_pop; i++ )
    {
    pr_val= random.fdraw( );
    if( pr_val <= mutation_pr )
      {
      int_loc= int( random.fdraw( ) *max_factors  );
      if( int_loc<0 )
```

www.banthebomb.org
Scottish Campaign for Nuclear Disarmament
15 Barrland Street • Glasgow • G41 1QH •
Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org

```
      int_loc=0;
    if( int_loc>=max_factors )
      int_loc=max_factors;

    pr_val= random.fdraw( );
      if( pr_val < 0.5 )
        Ai = alpha;
      else
        Ai = 1.0/alpha;
    es_strat[ i ][ int_loc ] = es_strat[ i ][ int_loc ]*Ai;
    }
  }

 return( pr_val );
 } ;    // Block: 57
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float ES_Chromosome::Xover( )

```
 {
 float pr_val=0.0 ;
 int pr_pop1, pr_pop2 ;
 float tmp ;
//
//          Can only cross over similar locations between populations.  The
//          probability is that each location represents a different independant
//          variable thus can not be dislocated.
//
 for( int i=0; i<max_factors; i++ )
   {
   for( int k=1; k<es_pop; k++ )
   {
   pr_val= random.fdraw( );
   if( pr_val <= Xover_pr/2.0 )
     {
     pr_pop1 = 1+ int( random.fdraw( ) *(es_pop-1)  );
     if( pr_pop1<1 )
       pr_pop1=1;
     if( pr_pop1>=es_pop )
       pr_pop1=es_pop;

     pr_pop2 = 1+ int( random.fdraw( ) *(es_pop-1)  );
     if( pr_pop2<1 )
       pr_pop2=1;
     if( pr_pop2>=es_pop )
       pr_pop2=es_pop;
```

```
      tmp = es_mean[ pr_pop1 ][ i ] ;
      es_mean[ pr_pop1 ][ i ] = es_mean[ pr_pop2 ][ i ];
      es_mean[ pr_pop2 ][ i ]= tmp ;
      }
    }
    for( k=1; k<es_pop; k++ )
    {
    pr_val= random.fdraw( );
    if( pr_val <= Xover_pr/2.0 )
      {
      pr_pop1 =  int( random.fdraw( ) *es_pop  );
      if( pr_pop1<0 )
        pr_pop1=0;
      if( pr_pop1>=es_pop )
        pr_pop1=es_pop;

      pr_pop2 =  int( random.fdraw( ) *es_pop  );
      if( pr_pop2<0 )
        pr_pop2=0;
      if( pr_pop2>=es_pop )
        pr_pop2=es_pop;

      tmp = es_strat[ pr_pop1 ][ i ] ;
      es_strat[ pr_pop1 ][ i ] = es_strat[ pr_pop2 ][ i ];
      es_strat[ pr_pop2 ][ i ]= tmp ;
      }
    }
    }

  return( pr_val );
} ;     // Block: 58
```

www.banthebomb.org

Scottish Campaign for Nuclear Disarmament

15 Barrland Street • Glasgow • G41 1QH •

Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org