ภาคผนวก ก

โปรแกรมคำนวณการเรียงแถบไม้ที่ดีที่สุด

โปรแกรมที่ 1 และโปรแกรมที่ 4 เป็นโปรแกรมที่ใช้คำนวณค่าในบทที่ 2 ส่วนโปรแกรมที่ 2 และโปรมแกรมที่ 5 เป็นโปรแกรมที่ใช้คำนวณในบทที่ 4 ซึ่งสมบัติเชิงกลต่าง ๆจะเปลี่ยนไปตามความยาวแถบไม้และความถ่วงจำเพาะของชิ้นทดสอบโดยค่าต่าง ๆแสดงในภาคผนวก ฉ โปรแกรมที่ 3 เป็นโปรแกรมหาการกระจายความเค้นโดยโปรแกรมทุกโปรแกรมใช้ Borland C++ เป็น compiler

1. โปรแกรมหาการเรียงตัวที่ดีที่สุด ใช้โมดูลัสยืดหยุ่น 2 ค่าคือ E_1 และE_2

   1.1 เกณฑ์ความเสียหายของไซ-ฮิลล์

```
//*************************************************************************//
//                                                                         //
//                        Optimal Orientation of OSL                       //
//                     According to Tsai-Hill Criterion                    //
//                           (Univariate search)                           //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

#define C1 -35.89E6        //   compression strength in the fiber direction
#define C2 -11.87E6        //   compression strength in the tangential direction
#define T1 56.684E6        //   tensile strength in the fiber direction
#define T2 2.795E6         //   tensile strength in the tangential direction
#define S12 15.985E6       //   shear strength in plane LT

#define dy 20              //   the number of layers
#define MAXLAYERS 30       //   the maximum number of layers allowance
#define H 0.152            //   thickness of OSL in meter

#define E1 28.05E9         //   MOE in the longitudinal (fiber:L) direction
#define E2 2.079E9         //   MOE in the transverse (tangential:T) direction
#define v12 0.35           //   possion's ratio in plane 1-2(LT)
#define G12 2.244E9        //   modulus of rigidity in plane 1-2(LT)

#define f_name "THC_antisym_Ed2.txt"

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy,double deg[MAXLAYERS],double sigma12[][2][3]);
   double find_min(double a[dy],double a_min);
   void display(int DEG[MAXLAYERS],double min_mx);

  int i,j,l,m;
  int lastchanged, change;
  int bestangle, lastangle;
  double DEG[MAXLAYERS],DEG_MAX_Mx[MAXLAYERS],DEG_optimize_Mx[MAXLAYERS],
         DEG_TMP[MAXLAYERS];
  double mx[2],min_mx[MAXLAYERS],mfail,MAX_Mx,optimize_Mx,
         SIGMA[MAXLAYERS][2][3],
         X,Y;
```

```
   optimize=fopen(f_name,"w+");
   optimize_Mx = 0.0;

// looping for initial trial degree
   for(m=0; m<=1000; m++)
    {
    gotoxy(1,1);
    printf("Intitial trial angles No. : %d\n",m);
    for(i=1; i<=dy/2; i++)
     {
      DEG[i] = random(180);    //  set initial trail degree  //
      DEG[dy-(i-1)] = -DEG[i];
      DEG_MAX_Mx[i] = 0.0;
      min_mx[i] = 0.0;
      }
      l = 1;
      lastchanged = 1;
    MAX_Mx=0.0;

    do
     {
        change = 0;
      bestangle = DEG[l];
         lastangle = DEG[l];
      do {
       evaluate_stress(dy,DEG,SIGMA);
     for(i=1; i<=dy; i++)
       {
       if (SIGMA[i][0][0] > 0) X = T1; else X = C1;
              if (SIGMA[i][0][1] > 0) Y = T2; else Y = C2;

              mx[0] = sqrt(1 /( (SIGMA[i][0][0]*SIGMA[i][0][0])/(X*X) -
                       (SIGMA[i][0][0]*SIGMA[i][0][1])/(X*X) +
                                 (SIGMA[i][0][1]*SIGMA[i][0][1])/(Y*Y) +
                                 (SIGMA[i][0][2]*SIGMA[i][0][2])/(S12+S12)
                               )
                         );

           if (SIGMA[i][1][0] > 0) X = T1; else X = C1;
              if (SIGMA[i][1][1] > 0) Y = T2; else Y = C2;

              mx[1] = sqrt(1/( (SIGMA[i][1][0]*SIGMA[i][1][0])/(X*X) -
                           (SIGMA[i][1][0]*SIGMA[i][1][1])/(X*X) +
                               (SIGMA[i][1][1]*SIGMA[i][1][1])/(Y*Y) +
                               (SIGMA[i][1][2]*SIGMA[i][1][2])/(S12*S12)
                     )
                         );

      min_mx[i] = mx[0];
           if(min_mx[i] > mx[1]) min_mx[i] = mx[1];
           }
      mfail = min_mx[1];                                  //*************************//
      for(j=1; j<= dy;j++)                                //    find minimum Mx in   //
       if (mfail > min_mx[j])   mfail = min_mx[j];        //     each orientation    //
                                                          //*************************//
      if(MAX_Mx < mfail)
            {                                 //************************************//
            MAX_Mx = mfail;                   //  compare Mx in each orietation for  //
               bestangle = DEG[l];            //            Finding max Mx           //
               change = 1;                    //                                     //
       }                                     //*************************************//
            if (DEG [l] > 180)
             DEG[l] = 1;
      DEG[l]++;
          DEG[dy-(l-1)] = -DEG[l];       //         VARY 1° EACH TIME           //
     } while(DEG[l] != lastangle);
          DEG[l] = bestangle;
          DEG[dy-(l-1)] = DEG[l];
          if (change == 1)
            lastchanged = l;
          l++;
          if (l > dy)l = 1;
   } while (l != lastchanged);
```

```
   //****Find Optimize orientation from univariate search****//
    if(MAX_Mx > optimize_Mx)
     {
     optimize_Mx = MAX_Mx;
      for(j=1; j<=dy; j++)
     DEG_optimize_Mx[j] = DEG_MAX_Mx[j];
     }
   }
//****************Display***************//
   fprintf(optimize,"Univariate Search\n");
   fprintf(optimize,"\nOptimal Orientation : [% 2lg°,% 2lg°,% 2lg°,% 2lg°,% 2lg°,%
2lg°,% 2lg°,% 2lg°,% 2lg°,% 2lg°]T\n",
          DEG_optimize_Mx[1], DEG_optimize_Mx[2], DEG_optimize_Mx[3],
          DEG_optimize_Mx[4], DEG_optimize_Mx[5], DEG_optimize_Mx[6],
                     DEG_optimize_Mx[7], DEG_optimize_Mx[8], DEG_optimize_Mx[9],
                     DEG_optimize_Mx[10]);
   fprintf(optimize,"Maximum moment (Nm/m) = %.4le\n\n",optimize_Mx);

//****Find optimize orientation by exhaustive search****//
   for(l=1; l<=dy; l++)
    DEG_TMP[l] = DEG_optimize_Mx[l];

   for(DEG[5]=DEG_TMP[5]-3;DEG[5]<=DEG_TMP[5]+3;DEG[5]=DEG[5]+1)
    {
      DEG[6]=(-1)*DEG[5];
 for(DEG[4]=DEG_TMP[4]-3;DEG[4]<=DEG_TMP[4]+3;DEG[4]=DEG[4]+1)
      {
         DEG[7]=(-1)*DEG[4];
 for(DEG[3]=DEG_TMP[3]-3;DEG[3]<=DEG_TMP[3]+3;DEG[3]=DEG[3]+1)
         {
           DEG[8]=(-1)*DEG[3];
 for(DEG[2]=DEG_TMP[2]-3;DEG[2]<=DEG_TMP[2]+3;DEG[2]=DEG[2]+1)
            {
               DEG[9]=(-1)*DEG[2];
 for(DEG[1]=DEG_TMP[1]-3;DEG[1]<=DEG_TMP[1]+3;DEG[1]=DEG[1]+1)
               {
                 DEG[10]=(-1)*DEG[1];
       evaluate_stress(dy,DEG,SIGMA);
                   for(i=1;i<=dy;i++)
                      {
                      gotoxy(1,2);printf("-");

                      gotoxy(1,2);printf("/");

                      gotoxy(1,2);printf("|");

                      gotoxy(1,2);printf("\\");
                      if(SIGMA[i][0][0]>0) X = T1; else X = C1;
                      if(SIGMA[i][1][0]>0) Y = T2; else Y = C2;

                      mx[0] = sqrt(1.0/( (SIGMA[i][0][0]*SIGMA[i][0][0])/(X*X) -
                                  (SIGMA[i][0][0]*SIGMA[i][0][1])/(X*X) +
                                        (SIGMA[i][0][1]*SIGMA[i][0][1])/(Y*Y) +
                                        (SIGMA[i][0][2]*SIGMA[i][0][2])/(S12*S12)
                                       )
                             );
                      if(SIGMA[i][0][1]>0) X = T1; else X = C1;
                      if(SIGMA[i][1][1]>0) Y = T2; else Y = C2;

                      mx[1] = sqrt(1.0/( (SIGMA[i][1][0]*SIGMA[i][1][0])/(X*X) -
                                  (SIGMA[i][1][0]*SIGMA[i][1][1])/(X*X) +
                                        (SIGMA[i][1][1]*SIGMA[i][1][1])/(Y*Y) +
                                        (SIGMA[i][1][2]*SIGMA[i][1][2])/(S12*S12)
                                       )
                             );
                      min_mx[i] = mx[0];               //***find min. Mx***//
                if(min_mx[i] > mx[1]) min_mx[i] = mx[1]; //***in each layer**//

         }
                   mfail=min_mx[1];
                   for(i=1; i<=dy/2; i++)
                    if(mfail>min_mx[i])mfail=min_mx[i];
                   if(optimize_Mx<mfail)
```

```
                    {
                    optimize_Mx=mfail;
                    for(j=1;j<=dy;j++)
                    DEG_optimize_Mx[j]=DEG[j];
                    }
                 }
              }
           }
        }
     }
   fprintf(optimize,"Exhaustive Search\n");
   fprintf(optimize,"\nOptimal Orientation : [% 2lg°,% 2lg°,% 2lg°,% 2lg°,% 2lg°,%
2lg°,% 2lg°,% 2lg°,% 2lg°,% 2lg°]T\n",
          DEG_optimize_Mx[1], DEG_optimize_Mx[2], DEG_optimize_Mx[3],
          DEG_optimize_Mx[4], DEG_optimize_Mx[5], DEG_optimize_Mx[6],
                    DEG_optimize_Mx[7], DEG_optimize_Mx[8], DEG_optimize_Mx[9],
                    DEG_optimize_Mx[10]);
   fprintf(optimize,"Maximum moment (Nm/m) = %.4le\n\n",optimize_Mx);
   fclose(optimize);
   gotoxy(1,2);printf("Finish!!!!!!");getch();
}
/*******************************End of main program*********************************/

//**************************Stresses Evaluation Function**************************//
void evaluate_stress(int Dy,double deg[MAXLAYERS],double sigma12[][2][3])
{
   void MAT_Q(double e1,double e2,double u12, double g12,double q[3][3]);
   void MAT_Transform(double deg,double t[3][3]);
   void MAT3_copy(int k,double t[3][3],double T[][3][3]);
   void MAT3_Inverse(double u[3][3],double r[3][3]);
   void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3]);
   void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);
   void MAT3_T(double a[3][3],double r[3][3]);
   void vec3_plus(double u[3],double v[3],int k,double w[][3]);

   int i,j,k;
   double Q[3][3],t[3][3],T_kth[MAXLAYERS][3][3],Q_bar[MAXLAYERS][3][3],
          T_Transpose_Inverse[3][3],T_Inverse[3][3],
          A[3][3],B[3][3],D[3][3],B_prime[3][3],D_prime[3][3],
          a[3][3],b[3][3],d[3][3],
          M[3],N[3],
          epsilon_o[3],keppa_o[3],
          epsilon_xy[2][3],
          h,Mx,z[dy+1],
          temp[3][3],temp1[3][3];

   Mx = 1.0;
   MAT_Q(E1,E2,v12,G12,Q);
   for(k=1;k<=Dy;k++)
     {
       MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
       MAT3_T(t,temp);
       MAT3_Inverse(temp,T_Transpose_Inverse);
       MAT3_Inverse(t,T_Inverse);
       Dot_MAT3(T_Inverse,Q,temp);
       Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);
     }

   h=H/Dy;
   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      {  A[i][j]=B[i][j]=D[i][j]=0.0;
      for(k=1;k<=dy;k++)
         {
            A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
            B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
                *(powl((-H/2 + h*k),2) - powl((-H/2 + h*(k-1)),2)));
            D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
                *(powl((-H/2 + h*k),3) - powl((-H/2 + h*(k-1)),3)));
         }
      }
```

```
   MAT3_Inverse(A,a);
   MAT3_Inverse(B,b);
   MAT3_Inverse(D,d);

   M[0]= Mx  ; M[1] = 0.0 ; M[2] = 0.0 ;
   N[0]= 0.0 ; N[1] = 0.0 ; N[2] = 0.0 ;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       B_prime[i][j]=D_prime[i][j]=temp[i][j]=0.0;

   //**************D'=(D-BA^(-1)B)^(-1)**************//
   Dot_MAT3(B,a,temp);
   Dot_MAT3(temp,B,temp1);
   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
        temp1[i][j]=D[i][j]-temp1[i][j];
   MAT3_Inverse(temp1,D_prime);

   //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
   Dot_MAT3(a,B,temp);
   MAT3_multip(temp,-1,temp1);
   Dot_MAT3(temp1,D_prime,B_prime);

   Dot_MAT3_vec3(B_prime,M,epsilon_o);
   Dot_MAT3_vec3(D_prime,M,keppa_o);

   //******** Determine stresses in plane 1-2 *******//
   for(k=1;k<=Dy;k++)
   {
    Dot_MAT3(T_kth[k],Q_bar[k],temp1);
      z[k-1]=-H/2+h*(k-1);
      z[k]=-H/2+h*(k);

      for(i=0;i<3;i++)
      {
      epsilon_xy[0][i]=epsilon_o[i] + keppa_o[i]*z[k-1] ;
         epsilon_xy[1][i]=epsilon_o[i] + keppa_o[i]*z[k]   ;
      }

      Dot_MAT3_vec3(temp1,epsilon_xy[0],sigma12[k][0]);
      Dot_MAT3_vec3(temp1,epsilon_xy[1],sigma12[k][1]);
   }

   //****************** Display ******************//
/*   printf("\n");
   printf("s1 s2 s12 top : s1 s2 s12 bottom\n\n");
   for(k=1;k<=dy;k++)
      {printf("% 11.4Le  % 11.4Le  % 11.4Le : % 11.4Le  % 11.4Le  % 11.4Le\n",
             sigma12[k][0][0], sigma12[k][0][1], sigma12[k][0][2],
                sigma12[k][1][0], sigma12[k][1][1], sigma12[k][1][2]);
      }getch();*/
}
/***********************End of Stresses Evaluation Function ***********************/

/**************************Matrix Q Determination Function*************************/
void MAT_Q(double e1, double e2, double u12,
      double g12,double q[3][3])
{
   double u21;

 u21=e2/e1*u12;

   q[0][0]=e1/(1-u12*u21);
   q[0][1]=q[1][0]=u12*e2/(1-u12*u21);
   q[1][1]=e2/(1-u12*u21);
   q[2][2]=g12;
   q[0][2]=q[1][2]=q[2][0]=q[2][1]=0.0;

   return;
}
/***********************End of Matrix Q Determination Function***********************/
/***********************Transformation Matrix ,[T], Function***********************/
```

```
void MAT_Transform(double deg,double t[3][3])
{
 t[0][0]=pow(cos(M_PI/180*deg),2);
   t[0][1]=pow(sin(M_PI/180*deg),2);
   t[0][2]=2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0]=pow(sin(M_PI/180*deg),2);
   t[1][1]=pow(cos(M_PI/180*deg),2);
   t[1][2]=-2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0]=-sin(M_PI/180*deg)*cosl(M_PI/180*deg);
   t[2][1]=sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2]=pow(cos(M_PI/180*deg),2)-pow(sin(M_PI/180*deg),2);

   return;
}
/*******************End of Transformation Matrix [T] Function **********************/

/*****************************Matrix Replication Function*****************************/
void MAT3_copy(int k,double t[3][3],double T[][3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    T[k][i][j]=t[i][j];
   return;
}
/*************************End of Matrix Replication Function***********************/

/*********************************Dot Matrix Function*********************************/
void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      w[i][j]=u[i][0]*v[0][j]+u[i][1]*v[1][j]+u[i][2]*v[2][j];
   return;
}
/*****************************End of Dot Matrix Function ***************************/

/*********************************Dot Vector Function*********************************/
void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3])
{
   int i;

 for(i=0;i<3;i++)
    w[i]=u[i][0]*v[0]+u[i][1]*v[1]+u[i][2]*v[2];
   return;
}
/*****************************End of Dot Vector Function ***************************/

/*********************************Inverse Matrix Function*****************************/
void MAT3_Inverse(double u[3][3],double r[3][3])
{
 double DET(double u[3][3]);
   void adj(double u[3][3],double r[3][3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);

   int i,j;
   double temp[3][3];

   adj(u,temp);
   if(DET(u)==0){
    for(i=0;i<3;i++)
     for(j=0;j<3;j++)
          r[i][j]=0.0;
      }else
   MAT3_multip(temp,1/DET(u),r);
   return;
}
/*****************************End of Inverse Matrix Function ***********************/

/*********************************Determinant Function*********************************/
```

```
double DET(double u[3][3])
{
   double temp;

 temp=u[0][0]*u[1][1]*u[2][2]+u[0][1]*u[1][2]*u[2][0]+u[0][2]*u[1][0]*u[2][1]
     -(u[2][0]*u[1][1]*u[0][2]+u[2][1]*u[1][2]*u[0][0]+u[2][2]*u[1][0]*u[0][1]);
   return(temp);
}
/*****************************End of Determinant Function*****************************/

/*****************************Adjoint Matrix Function*****************************/
void adj(double u[3][3],double r[3][3])
{
   void MAT3_T(double a[3][3],double r[3][3]);
   void Cof(double a[3][3],double r[3][3]);

   double temp[3][3];

   Cof(u,temp);
   MAT3_T(temp,r);
   return;
}
/*****************************End of Determinant Function*****************************/

/*****************************Matrix Transpose Function*****************************/
void MAT3_T(double a[3][3],double r[3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=a[j][i];
   return;
}
/*************************End of Matrix Transpose Function*************************/

/*****************************Cofactor Matrix Function*****************************/
void Cof(double a[3][3],double r[3][3])
{
   int i,j;
   double M[3][3];

 M[0][0]=a[1][1]*a[2][2]-a[2][1]*a[1][2];
   M[0][1]=a[1][0]*a[2][2]-a[2][0]*a[1][2];
   M[0][2]=a[1][0]*a[2][1]-a[2][0]*a[1][1];
   M[1][0]=a[0][1]*a[2][2]-a[2][1]*a[0][2];
   M[1][1]=a[0][0]*a[2][2]-a[2][0]*a[0][2];
   M[1][2]=a[0][0]*a[2][1]-a[2][0]*a[0][1];
   M[2][0]=a[0][1]*a[1][2]-a[1][1]*a[0][2];
   M[2][1]=a[0][0]*a[1][2]-a[1][0]*a[0][2];
   M[2][2]=a[0][0]*a[1][1]-a[1][0]*a[0][1];

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=powl(-1,i+j)*M[i][j];
   return;
}
/*************************End of Cofactor Matrix Function*************************/

/*************************Scalar-Multiply-Matrix Function*************************/
void MAT3_multip(double u[3][3],double k,double r[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=k*u[i][j];
   return;
}
/*********************End of Scalar-Multiply-Matrix Function*********************/


/*****************************Combination Vector Function*****************************/
```

```
void vec3_plus(double u[3],double v[3],int k,double w[][3])
{
   int i;

 for(i=0;i<3;i++)
    w[k][i]=u[i]+v[i];
   return;
}
/***********************End of Combination Vector Function ************************/

/*******************************Minimum Value Function*******************************/
double find_min(double a[2],double a_min)
{
   int i;

   a_min=1.0e500;
 for(i=0;i<3;i++)
   {
      if(a[i]>0)
      if(a_min>a[i])a_min=a[i];
   }
   return a_min;
}
/*************************End of Minimum Value Function **************************/
```

1.2 เกณฑ์ความเค้นสูงสุด

```
//*************************************************************************//
//                                                                         //
//                        Optimal Orientation of OSL                       //
//                    According to Maximum Stress Criterion                //
//                             (Univariate search)                         //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

#define C1 -35.89E6        //   compression strength in the fiber direction
#define C2 -11.87E6        //   compression strength in the tangential direction
#define T1 56.684E6        //   tensile strength in the fiber direction
#define T2 2.795E6         //   tendile strength in the tangential direction
#define S12 15.985E6       //   shear strength in plane LT

#define dy 10              //   the number of layers
#define MAXLAYERS 20       //   the maximum number of layers allowance
#define H 0.152            //   thickness of OSL in meter

#define E1 28.05E9         //   MOE in the longitudinal (fiber:L) direction
#define E2 2.079E9         //   MOE in the transverse (tangential:T) direction
#define v12 0.35           //   possion's ratio in plane 1-2(LT)
#define G12 2.244E9        //   modulus of rigidity in plane 1-2(LT)

#define f_name "MSC_antisym.txt"

int i,j,k,l,m,ch;

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy,int deg[MAXLAYERS],long double sigma12[][2][3]);
   long double find_min(long double a[dy],long double a_min);
   void display(int DEG[MAXLAYERS],long double min_mx);
   int i,j,k,l,DEG[MAXLAYERS],DEG_MAX_Mx[MAXLAYERS],DEG_optimize_Mx[MAXLAYERS],
       DEG_TMP[MAXLAYERS];
   long double mx[dy],min_mx[MAXLAYERS],MIN_Mx,MAX_Mx,optimize_Mx,
               SIGMA[MAXLAYERS][2][3];
```

```
clrscr();
optimize=fopen(f_name,"w+");
optimize_Mx = 0.0;
//*** looping for initial trial degree***//
for(m = 0; m < 1000; m++) {
 clrscr();printf("%d",m);
 //printf("\nInitial trial degree = [");
 for(i=1;i<=dy/2;i++)
  {
  DEG[i] = random(181);                    //  set initial trail degree  //
   DEG[dy-(i-1)] = -DEG[i];
   DEG_MAX_Mx[i] = DEG_MAX_Mx[dy-(i-1)]= 0.0;
   min_mx[i] = min_mx[dy-(i-1)] = 0.0;
   }

 MAX_Mx=0.0;
 START:
 for(l=1;l<=dy;l++)
   DEG_TMP[l]=DEG_MAX_Mx[l];  /*CHECKER*/
 for(l=1;l<=dy/2;l++) {
   DEG[l]=DEG[dy-(l-1)]=0.0;
   do {
    evaluate_stress(dy,DEG,SIGMA);
   for(i=1;i<=dy;i++) {

    if(SIGMA[i][0][0]==0)mx[0]=1e300;       //*********************************//
    else mx[0]=(C1/SIGMA[i][0][0]);         //                                 //
        if(SIGMA[i][1][0]==0)mx[1]=1e300;   //                                 //
    else mx[1]=(C1/SIGMA[i][1][0]);         //                                 //
                                            //                                 //
    if(SIGMA[i][0][0]==0)mx[2]=1e300;       //       find Maximum Mx in        //
    else mx[2]=(T1/SIGMA[i][0][0]);         //        each failure mode        //
        if(SIGMA[i][1][0]==0)mx[3]=1e300;   //                                 //
    else mx[3]=(T1/SIGMA[i][1][0]);         //                                 //
                                            //                                 //
    if(SIGMA[i][0][1]==0)mx[4]=1e300;       //                                 //
    else mx[4]=(C2/SIGMA[i][0][1]);         //                                 //
        if(SIGMA[i][1][1]==0)mx[5]=1e300;   //                                 //
    else mx[5]=(C2/SIGMA[i][1][1]);         // ( if  a stress is zero then     //
                                            //                                 //
    if(SIGMA[i][0][1]==0)mx[6]=1e300;       //      Mx will be set to          //
    else mx[6]=(T2/SIGMA[i][0][1]);         //         enormous value      )   //
        if(SIGMA[i][1][1]==0)mx[7]=1e300;   //                                 //
    else mx[7]=(T2/SIGMA[i][1][1]);         //                                 //
                                            //                                 //
    if(SIGMA[i][0][2]==0)mx[8]=1e300;       //                                 //
    else mx[8]=(S12/SIGMA[i][0][2]);        //                                 //
        if(SIGMA[i][1][2]==0)mx[9]=1e300;   //                                 //
    else mx[9]=(S12/SIGMA[i][1][2]);        //*********************************//

   min_mx[i]=find_min(mx,min_mx[i]);        // find minimum Mx in each layer  //
    }
      MIN_Mx=min_mx[1];                     //*********************************//
   for(j=1;j<=dy;j++)                       //         find minimum Mx in      //
   if(MIN_Mx>min_mx[j])MIN_Mx=min_mx[j];    //         each orientation        //
                                            //*********************************//
   if(MAX_Mx<MIN_Mx){                     //************************************//
    MAX_Mx=MIN_Mx;                        //  compare Mx in each orietation for  //
    for(k=1;k<=dy;k++)                    //           finding maximum Mx         //
        DEG_MAX_Mx[k]=DEG[k];             //          (optimal orientation)       //
      }                                   //************************************//

   DEG[l]=DEG[l]+1;
      DEG[dy-(l-1)]=-DEG[l];              //         VARY 1° EACH TIME           //
} while (DEG[l]<=180);
   for(j=1;j<=dy;j++)
 DEG[j]=DEG_MAX_Mx[j];
}
for(l=1;l<=dy;l++){         //         EXAMINE WHEN SHOULD BE FINISHED :          //
                            //  IF STACK SEGUENCES ARE NOT CHANGED THEN FINISHED //
   if (DEG_TMP[l]==DEG_MAX_Mx[l]) {
       if(l==dy){
        goto END;
```

```
        }
     } else l=dy+1;
   }
  //*******Find Optimize orientation from univariate search*******//
   if(MAX_Mx > optimize_Mx){
   optimize_Mx = MAX_Mx;
      for(j=1;j<=dy;j++)
    DEG_optimize_Mx[j]=DEG_MAX_Mx[j];
   }
 goto START;
  END:
 }
 //*******Display Optimal Orientation & Maximum Moment*******//
 fprintf(optimize,"Univariate Search\n");
 fprintf(optimize,"\nOptimal Orientation : [%2d°,%2d°,%2d°,%2d°,%2d°]s\n",
          DEG_optimize_Mx[1], DEG_optimize_Mx[2], DEG_optimize_Mx[3],
                DEG_optimize_Mx[4], DEG_optimize_Mx[5]);
 fprintf(optimize,"Maximum moment (Nm/m) = % .4Le\n\n",optimize_Mx);

 //*******Find optimize orientation by exhaustive search*******//
 for(l=1;l<=dy;l++)
     DEG_TMP[l]=DEG_optimize_Mx[l];

 for(DEG[5]=DEG_TMP[5]-3;DEG[5]<=DEG_TMP[5]+3;DEG[5]=DEG[5]+1)
  {DEG[6]=(-1)*DEG[5];
for(DEG[4]=DEG_TMP[4]-3;DEG[4]<=DEG_TMP[4]+3;DEG[4]=DEG[4]+1)
     {DEG[7]=(-1)*DEG[4];
for(DEG[3]=DEG_TMP[3]-3;DEG[3]<=DEG_TMP[3]+3;DEG[3]=DEG[3]+1)
         {DEG[8]=(-1)*DEG[3];
for(DEG[2]=DEG_TMP[2]-3;DEG[2]<=DEG_TMP[2]+3;DEG[2]=DEG[2]+1)
           {DEG[9]=(-1)*DEG[2];
for(DEG[1]=DEG_TMP[1]-3;DEG[1]<=DEG_TMP[1]+3;DEG[1]=DEG[1]+1)
               {DEG[10]=(-1)*DEG[1];

          evaluate_stress(dy,DEG,SIGMA);
                    for(i=1;i<=dy;i++)
                    {
          if(SIGMA[i][0][0]==0)mx[0]=1e300;       //******************************//
          else mx[0]=(C1/SIGMA[i][0][0]);         //                              //
          if(SIGMA[i][1][0]==0)mx[1]=1e300;       //                              //
          else mx[1]=(C1/SIGMA[i][1][0]);         //                              //
                                                  //                              //
                      gotoxy(1,2);printf("-");    //                              //
                                                  //                              //
          if(SIGMA[i][0][0]==0)mx[2]=1e300;       //       Find Maximum Mx in     //
          else mx[2]=(T1/SIGMA[i][0][0]);         //        Each Failue mode      //
          if(SIGMA[i][1][0]==0)mx[3]=1e300;       //                              //
          else mx[3]=(T1/SIGMA[i][1][0]);         //                              //
                                                  //                              //
                      gotoxy(1,2);printf("/");    //                              //
                                                  //                              //
          if(SIGMA[i][0][1]==0)mx[4]=1e300;       //                              //
          else mx[4]=(C2/SIGMA[i][0][1]);         //                              //
          if(SIGMA[i][1][1]==0)mx[5]=1e300;       //                              //
          else mx[5]=(C2/SIGMA[i][1][1]);         // ( if  a stress is zero then  //
                                                  //                              //
                      gotoxy(1,2);printf("|");    //                              //
                                                  //                              //
          if(SIGMA[i][0][1]==0)mx[6]=1e300;       //      Mx will be set to       //
          else mx[6]=(T2/SIGMA[i][0][1]);         //         enormous value     ) //
          if(SIGMA[i][1][1]==0)mx[7]=1e300;       //                              //
          else mx[7]=(T2/SIGMA[i][1][1]);         //                              //
                                                  //                              //
                      gotoxy(1,2);printf("\\");   //                              //
                                                  //                              //
          if(SIGMA[i][0][2]==0)mx[8]=1e300;       //                              //
          else mx[8]=(S12/SIGMA[i][0][2]);        //                              //
          if(SIGMA[i][1][2]==0)mx[9]=1e300;       //                              //
          else mx[9]=(S12/SIGMA[i][1][2]);        //******************************//

         min_mx[i]=find_min(mx,min_mx[i]);        // find minimum Mx in each layer //
        }
                    MIN_Mx=min_mx[1];
```

```
                    for(i=1;i<=dy/2;i++)
                    if(MIN_Mx>min_mx[i])MIN_Mx=min_mx[i];
                    if(optimize_Mx<MIN_Mx)
                    {optimize_Mx=MIN_Mx;
                    for(j=1;j<=dy;j++)
                        DEG_optimize_Mx[j]=DEG[j];
                    }
                  }
               }
            }
         }
      fprintf(optimize,"Exhaustive Search\n");
      fprintf(optimize,"\nOptimal Orientation :
[%2d°,%2d°,%2d°,%2d°,%2d°,%2d°,%2d°,%2d°,%2d°,%2d°]s\n",
            DEG_optimize_Mx[1], DEG_optimize_Mx[2], DEG_optimize_Mx[3],
            DEG_optimize_Mx[4], DEG_optimize_Mx[5], DEG_optimize_Mx[6],
                    DEG_optimize_Mx[7], DEG_optimize_Mx[8], DEG_optimize_Mx[9],
                    DEG_optimize_Mx[10]);
      fprintf(optimize,"Maximum moment (Nm/m) = % .4Le\n\n",optimize_Mx);
      fclose(optimize);
      gotoxy(1,2);printf("Finish!!!!!!");getch();
}
/*****************************End of Main Program*********************************/

//**************************Stresses Evaluation Function*************************//
void evaluate_stress(int Dy,double deg[MAXLAYERS],double sigma12[][2][3])
{
   void MAT_Q(double e1,double e2,double u12, double g12,double q[3][3]);
   void MAT_Transform(double deg,double t[3][3]);
   void MAT3_copy(int k,double t[3][3],double T[][3][3]);
   void MAT3_Inverse(double u[3][3],double r[3][3]);
   void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3]);
   void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);
   void MAT3_T(double a[3][3],double r[3][3]);
   void vec3_plus(double u[3],double v[3],int k,double w[][3]);

   int i,j,k;
   double Q[3][3],t[3][3],T_kth[MAXLAYERS][3][3],Q_bar[MAXLAYERS][3][3],
          T_Transpose_Inverse[3][3],T_Inverse[3][3],
          A[3][3],B[3][3],D[3][3],B_prime[3][3],D_prime[3][3],
          a[3][3],b[3][3],d[3][3],
          M[3],N[3],
          epsilon_o[3],keppa_o[3],
          epsilon_xy[2][3],
          h,Mx,z[dy+1],
          temp[3][3],temp1[3][3];

   Mx = 1.0;
   MAT_Q(E1,E2,v12,G12,Q);
   for(k=1;k<=Dy;k++)
     {
       MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
       MAT3_T(t,temp);
       MAT3_Inverse(temp,T_Transpose_Inverse);
       MAT3_Inverse(t,T_Inverse);
       Dot_MAT3(T_Inverse,Q,temp);
       Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);
     }

   h=H/Dy;
   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      {  A[i][j]=B[i][j]=D[i][j]=0.0;
      for(k=1;k<=dy;k++)
         {
            A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
            B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
                *(powl((-H/2 + h*k),2) - powl((-H/2 + h*(k-1)),2)));
            D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
                *(powl((-H/2 + h*k),3) - powl((-H/2 + h*(k-1)),3)));
         }
```

```
      }

   MAT3_Inverse(A,a);
   MAT3_Inverse(B,b);
   MAT3_Inverse(D,d);

   M[0]= Mx  ; M[1] = 0.0 ; M[2] = 0.0 ;
   N[0]= 0.0 ; N[1] = 0.0 ; N[2] = 0.0 ;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       B_prime[i][j]=D_prime[i][j]=temp[i][j]=0.0;

   //**************D'=(D-BA^(-1)B)^(-1)**************//
   Dot_MAT3(B,a,temp);
   Dot_MAT3(temp,B,temp1);
   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
        temp1[i][j]=D[i][j]-temp1[i][j];
   MAT3_Inverse(temp1,D_prime);

   //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
   Dot_MAT3(a,B,temp);
   MAT3_multip(temp,-1,temp1);
   Dot_MAT3(temp1,D_prime,B_prime);

   Dot_MAT3_vec3(B_prime,M,epsilon_o);
   Dot_MAT3_vec3(D_prime,M,keppa_o);

   //******** Determine stresses in plane 1-2 *******//
   for(k=1;k<=Dy;k++)
   {
    Dot_MAT3(T_kth[k],Q_bar[k],temp1);
       z[k-1]=-H/2+h*(k-1);
       z[k]=-H/2+h*(k);

       for(i=0;i<3;i++)
       {
       epsilon_xy[0][i]=epsilon_o[i] + keppa_o[i]*z[k-1] ;
          epsilon_xy[1][i]=epsilon_o[i] + keppa_o[i]*z[k]   ;
       }

       Dot_MAT3_vec3(temp1,epsilon_xy[0],sigma12[k][0]);
       Dot_MAT3_vec3(temp1,epsilon_xy[1],sigma12[k][1]);
   }

   //****************** Display ******************//
   printf("s1 s2 s12 top : s1 s2 s12 bottom\n\n");
   for(k=1;k<=dy;k++)
      {printf("% 11.4Le  % 11.4Le  % 11.4Le : % 11.4Le  % 11.4Le  % 11.4Le\n",
             sigma12[k][0][0], sigma12[k][0][1], sigma12[k][0][2],
                 sigma12[k][1][0], sigma12[k][1][1], sigma12[k][1][2]);
      }getch();*/
}
/***********************End of Stresses Evaluation Function ***********************/

/***************************Matrix Q Determination Function*************************/
void MAT_Q(double e1, double e2, double u12,
      double g12,double q[3][3])
{
   double u21;

 u21=e2/e1*u12;

   q[0][0]=e1/(1-u12*u21);
   q[0][1]=q[1][0]=u12*e2/(1-u12*u21);
   q[1][1]=e2/(1-u12*u21);
   q[2][2]=g12;
   q[0][2]=q[1][2]=q[2][0]=q[2][1]=0.0;

   return;
}
/***********************End of Matrix Q Determination Function***********************/
```

```
/**********************Transformation Matrix ,[T], Function***********************/
void MAT_Transform(double deg,double t[3][3])
{
 t[0][0]=pow(cos(M_PI/180*deg),2);
   t[0][1]=pow(sin(M_PI/180*deg),2);
   t[0][2]=2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0]=pow(sin(M_PI/180*deg),2);
   t[1][1]=pow(cos(M_PI/180*deg),2);
   t[1][2]=-2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0]=-sin(M_PI/180*deg)*cosl(M_PI/180*deg);
   t[2][1]=sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2]=pow(cos(M_PI/180*deg),2)-pow(sin(M_PI/180*deg),2);

   return;
}
/*******************End of Transformation Matrix [T] Function **********************/

/*****************************Matrix Replication Function*****************************/
void MAT3_copy(int k,double t[3][3],double T[][3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    T[k][i][j]=t[i][j];
   return;
}
/*************************End of Matrix Replication Function***********************/

/*********************************Dot Matrix Function*********************************/
void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       w[i][j]=u[i][0]*v[0][j]+u[i][1]*v[1][j]+u[i][2]*v[2][j];
   return;
}
/*****************************End of Dot Matrix Function ***************************/

/*********************************Dot Vector Function*********************************/
void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3])
{
   int i;

 for(i=0;i<3;i++)
    w[i]=u[i][0]*v[0]+u[i][1]*v[1]+u[i][2]*v[2];
   return;
}
/*****************************End of Dot Vector Function ***************************/

/*********************************Inverse Matrix Function*****************************/
void MAT3_Inverse(double u[3][3],double r[3][3])
{
 double DET(double u[3][3]);
   void adj(double u[3][3],double r[3][3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);

   int i,j;
   double temp[3][3];

   adj(u,temp);
   if(DET(u)==0){
    for(i=0;i<3;i++)
     for(j=0;j<3;j++)
          r[i][j]=0.0;
      }else
   MAT3_multip(temp,1/DET(u),r);
   return;
}
/*****************************End of Inverse Matrix Function ***********************/
```

```
/*********************************Determinant Function*****************************/
double DET(double u[3][3])
{
   double temp;

 temp=u[0][0]*u[1][1]*u[2][2]+u[0][1]*u[1][2]*u[2][0]+u[0][2]*u[1][0]*u[2][1]
     -(u[2][0]*u[1][1]*u[0][2]+u[2][1]*u[1][2]*u[0][0]+u[2][2]*u[1][0]*u[0][1]);
   return(temp);
}
/***************************End of Determinant Function*****************************/

/*****************************Adjoint Matrix Function*******************************/
void adj(double u[3][3],double r[3][3])
{
   void MAT3_T(double a[3][3],double r[3][3]);
   void Cof(double a[3][3],double r[3][3]);

   double temp[3][3];

   Cof(u,temp);
   MAT3_T(temp,r);
   return;
}
/***************************End of Determinant Function*****************************/

/*****************************Matrix Transpose Function*****************************/
void MAT3_T(double a[3][3],double r[3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=a[j][i];
   return;
}
/************************End of Matrix Transpose Function***************************/

/*****************************Cofactor Matrix Function******************************/
void Cof(double a[3][3],double r[3][3])
{
   int i,j;
   double M[3][3];

   M[0][0]=a[1][1]*a[2][2]-a[2][1]*a[1][2];
   M[0][1]=a[1][0]*a[2][2]-a[2][0]*a[1][2];
   M[0][2]=a[1][0]*a[2][1]-a[2][0]*a[1][1];
   M[1][0]=a[0][1]*a[2][2]-a[2][1]*a[0][2];
   M[1][1]=a[0][0]*a[2][2]-a[2][0]*a[0][2];
   M[1][2]=a[0][0]*a[2][1]-a[2][0]*a[0][1];
   M[2][0]=a[0][1]*a[1][2]-a[1][1]*a[0][2];
   M[2][1]=a[0][0]*a[1][2]-a[1][0]*a[0][2];
   M[2][2]=a[0][0]*a[1][1]-a[1][0]*a[0][1];

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=powl(-1,i+j)*M[i][j];
   return;
}
/************************End of Cofactor Matrix Function****************************/

/************************Scalar-Multiply-Matrix Function****************************/
void MAT3_multip(double u[3][3],double k,double r[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=k*u[i][j];
   return;
}
/*********************End of Scalar-Multiply-Matrix Function************************/
```

```
/*************************Combination Vector Function*****************************/
void vec3_plus(double u[3],double v[3],int k,double w[][3])
{
   int i;

 for(i=0;i<3;i++)
    w[k][i]=u[i]+v[i];
   return;
}
/**********************End of Combination Vector Function ************************/

/*****************************Minimum Value Function*******************************/
double find_min(double a[2],double a_min)
{
   int i;

   a_min=1.0e500;
 for(i=0;i<3;i++)
   {
      if(a[i]>0)
      if(a_min>a[i])a_min=a[i];
   }
   return a_min;
}
/*************************End of Minimum Value Function ***************************/
```

2. โปรแกรมหาการเรียงตัวที่ดีที่สุด ใช้โมดูลัสยืดหยุ่น 4 ค่าคือ E_1^T E_1^C และ E_2^T E_2^C

```
//*************************************************************************//
//                                                                         //
//                        Optimal Orientation of OSL                       //
//                      According to Tsai-Hill Criterion                   //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

#define C1 -53.774E6       //   compression strength in the fiber direction
#define C2 -53.774E6       //   compression strength in the tangential direction
#define T1 53.774E6        //   tensile strength in the fiber direction
#define T2 1.39E6          //   tendile strength in the tangential direction
#define S12 15.985E6       //   shear strength in plane LT

#define dy 20              //  the number of layers
#define MAXLAYERS 30       //   the maximum number of layers allowance
#define H 0.020            //   thickness of OSL in meter

#define EC1 4335.26E6      //   MOE in the longitudinal direction
#define ET1 1638.15E6      //   MOE in the longitudinal direction
#define EC2 228.44E6       //   MOE in the transverse direction @compression region
#define ET2 235.38E6       //   MOE in the transverse direction @tension region

#define v12 0.6            //   possion's ratio in plane 1-2(LT)
#define G12 120E6          //   modulus of rigidity in plane 1-2(LT)

#define f_name "THC_NS_Ed2.txt"

double NA_in, NA_out;

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy, double deg[MAXLAYERS], double sigma12[][2][3]);
   double find_min(double a[dy], double a_min);
   void display(int DEG[MAXLAYERS], double min_mx);
   int i, j, k, l, m, n, changed;
   double  DEG[MAXLAYERS], DEG_MAX_Mx[MAXLAYERS], DEG_optimize_Mx[MAXLAYERS],
```

```
        DEG_optimize_min_Mx[MAXLAYERS], DEG_TMP[MAXLAYERS];
   double mx[2], min_mx[MAXLAYERS], MIN_Mx,MAX_Mx, optimize_Mx, optimize_min_Mx,
        SIGMA[MAXLAYERS][2][3],
        X, Y;

   clrscr();
   optimize = fopen(f_name, "w+");
   optimize_Mx = 0.0;

//*****looping for initial trial degree*****//
   for(m = 0; m <= 500; m++)
    {
    gotoxy(1,1);
    printf("No. : %d", m);
    for(i = 1; i <= dy; i++)
     {
      DEG[i] = random(181);    //  set initial trail degree  //
      DEG_MAX_Mx[i] = 0.0;
      min_mx[i] = 0.0;
      }
    MAX_Mx = 0.0;
    do
      {
         changed = 0;
     for(l = 1; l <= dy; l++)
        DEG_TMP[l] = DEG_MAX_Mx[l];  /*CHECKER*/
     for(l = 1; l <= dy; l++)
      {
       DEG[l] = 0.0;
       do
        {
              n = 0;
              do{
         n = n + 1;
        NA_in = n*H/dy;
       evaluate_stress(dy,DEG,SIGMA);
              }while((NA_out - NA_in) > 0.0005);

        for(i = 1; i <= dy; i++)
        {
        if(SIGMA[i][0][0] > 0) X = T1; else X = C1;
             if(SIGMA[i][0][1] > 0) Y = T2; else Y = C2;
             mx[0] = sqrt(1.0/( 1.0/(X*X)*(SIGMA[i][0][0]*SIGMA[i][0][0]) -
                      (1.0/(X*X)*SIGMA[i][0][0]*SIGMA[i][0][1]) +
                           (1.0/(Y*Y)*SIGMA[i][0][1]*SIGMA[i][0][1]) +
                           (1.0/(S12*S12)*SIGMA[i][0][2]*SIGMA[i][0][2]))
                    );

             if(SIGMA[i][1][0]>0) X = T1; else X = C1;
             if(SIGMA[i][1][1]>0) Y = T2; else Y = C2;
             mx[1] = sqrt(1.0/( (SIGMA[i][1][0]*SIGMA[i][1][0])/(X*X) -
                    (SIGMA[i][1][0]*SIGMA[i][1][1])/(X*X) +
                           (SIGMA[i][1][1]*SIGMA[i][1][1])/(Y*Y) +
                           (SIGMA[i][1][2]*SIGMA[i][1][2])/(S12*S12))
                    );
       min_mx[i] = mx[0];
            if(min_mx[i] > mx[1]) min_mx[i] = mx[1];
        }
       MIN_Mx = min_mx[1];                          //*************************//
       for(j = 1; j <= dy; j++)                     //    find minimum Mx in   //
        if( MIN_Mx > min_mx[j] ) MIN_Mx=min_mx[j];  //     each orientation    //
                                                    //*************************//
       if(MAX_Mx < MIN_Mx)              //************************************//
             {                          //                                    //
              MAX_Mx = MIN_Mx;          //  compare Mx in each orietation for  //
              for(k = 1; k <= dy; k++)  //          finding maximum Mx         //
              DEG_MAX_Mx[k] = DEG[k];   //         (optimal orientation)       //
            }                           //************************************//

       DEG[l] = DEG[l] + 1;
           DEG[dy - (l - 1)] = -DEG[l];       //         VARY 1 EACH TIME           //
      }while(DEG[l] <= 180.0);
```

```
       for(j = 1; j <= dy; j++)
      DEG[j] = DEG_MAX_Mx[j];
      }
     for(l = 1; l <= dy; l++)    //     CHECKED THAT WHEN SHOULD BE FINISHED :       //
      {                          // IF STACK SEGUENCES ARE NOT CHANGED THEN FINISHED //
     if(DEG_TMP[l]!=DEG_MAX_Mx[l])
         {
               changed = 1;
               l = dy+1;
               }
      }

   //*******Find Optimize orientation from univariate search*******//
     if(MAX_Mx > optimize_Mx)
      {
      optimize_Mx = MAX_Mx;
       for(j = 1; j <= dy; j++)
     DEG_optimize_Mx[j] = DEG_MAX_Mx[j];
      }
    }while(changed == 1);
   }
   fprintf(optimize, "Univariate Search\n");
   fprintf(optimize, "\nOptimal Orientation : [% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,%
2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,%
2lg]T\n",
          DEG_optimize_Mx[1], DEG_optimize_Mx[2], DEG_optimize_Mx[3],
          DEG_optimize_Mx[4], DEG_optimize_Mx[5], DEG_optimize_Mx[6],
          DEG_optimize_Mx[7], DEG_optimize_Mx[8], DEG_optimize_Mx[9],
          DEG_optimize_Mx[10],DEG_optimize_Mx[11],DEG_optimize_Mx[12],
          DEG_optimize_Mx[13],DEG_optimize_Mx[14],DEG_optimize_Mx[15],
          DEG_optimize_Mx[16],DEG_optimize_Mx[17],DEG_optimize_Mx[18],
          DEG_optimize_Mx[19],DEG_optimize_Mx[20]);
   fprintf(optimize, "Maximum moment (Nm/m) = %.4le\n",optimize_Mx);
   fprintf(optimize, "N.A. refered from top = %.4le\n\n",NA_out);

   //*******Find optimize orientation by exhaustive search (Optional)********//

   for(l = 1; l <= dy; l++)
    DEG_TMP[l] =  DEG_optimize_Mx[l];
   optimize_min_Mx = optimize_Mx;

   for(DEG[10] = DEG_TMP[10] - 3; DEG[10] <= DEG_TMP[10] + 3; DEG[10] = DEG[10] + 1)
    {
      DEG[11] = (-1)*DEG[10];
 for(DEG[9] = DEG_TMP[9] - 3; DEG[9] <= DEG_TMP[9] + 3; DEG[9] = DEG[9] + 1)
      {
         DEG[12] = (-1)*DEG[9];
 for(DEG[8] = DEG_TMP[8] - 3; DEG[8] <= DEG_TMP[8] + 3; DEG[8] = DEG[8] + 1)
          {
            DEG[13] = (-1)*DEG[8];
 for(DEG[7] = DEG_TMP[7] - 3; DEG[7] <= DEG_TMP[7] + 3; DEG[7] = DEG[7] + 1)
             {
                DEG[14] = (-1)*DEG[7];
 for(DEG[6] = DEG_TMP[6] - 3; DEG[6] <= DEG_TMP[6] + 3; DEG[6] = DEG[6] + 1)
               {
                 DEG[15] = (-1)*DEG[6];
 for(DEG[5] = DEG_TMP[5] - 3; DEG[5] <= DEG_TMP[5] + 3; DEG[5] = DEG[5] + 1)
   {
     DEG[16] = (-1)*DEG[5];
 for(DEG[4] = DEG_TMP[4] - 3; DEG[4] <= DEG_TMP[4] + 3; DEG[4] = DEG[4] + 1)
      {
         DEG[17] = (-1)*DEG[4];
 for(DEG[3] = DEG_TMP[3] - 3; DEG[3] <= DEG_TMP[3] + 3; DEG[3] = DEG[3] + 1)
          {
            DEG[18] = (-1)*DEG[3];
 for(DEG[2] = DEG_TMP[2] - 3; DEG[2] <= DEG_TMP[2] + 3; DEG[2] = DEG[2] + 1)
             {
                DEG[19] = (-1)*DEG[2];
 for(DEG[1] = DEG_TMP[1] - 3; DEG[1] <= DEG_TMP[1] + 3; DEG[1] = DEG[1] + 1)
               {
                 DEG[20] = (-1)*DEG[1];

          evaluate_stress(dy,DEG,SIGMA);
```

```
                    for(i = 1; i <= dy; i++)
                     {
                        gotoxy(1,2);printf("-");

                        gotoxy(1,2);printf("/");

                        gotoxy(1,2);printf("|");

                        gotoxy(1,2);printf("\\");
                        if(SIGMA[i][0][0] > 0) X = T1; else X = C1;
                        if(SIGMA[i][1][0] > 0) Y = T2; else Y = C2;

                        mx[0] = sqrt(1.0/( (SIGMA[i][0][0]*SIGMA[i][0][0])/(X*X) -
                                  (SIGMA[i][0][0]*SIGMA[i][0][1])/(X*X) +
                                      (SIGMA[i][0][1]*SIGMA[i][0][1])/(Y*Y) +
                                      (SIGMA[i][0][2]*SIGMA[i][0][2])/(S12*S12) )
                                );
                if(SIGMA[i][0][1] > 0) X = T1; else X = C1;
                        if(SIGMA[i][1][1] > 0) Y = T2; else Y = C2;

                        mx[1] = sqrt(1.0/( (SIGMA[i][1][0]*SIGMA[i][1][0])/(X*X) -
                                  (SIGMA[i][1][0]*SIGMA[i][1][1])/(X*X) +
                                      (SIGMA[i][1][1]*SIGMA[i][1][1])/(Y*Y) +
                                      (SIGMA[i][1][2]*SIGMA[i][1][2])/(S12*S12) )
                                    );
            min_mx[i] = mx[0];
                 if(min_mx[i] > mx[1]) min_mx[i] = mx[1];  //find min Mx in each layer //
            }
                     MIN_Mx = min_mx[1];
                     for(i = 1; i <= dy/2; i++)
                     if(MIN_Mx > min_mx[i])  MIN_Mx = min_mx[i];

                     if(optimize_Mx < MIN_Mx)
                        {
                        optimize_Mx = MIN_Mx;
                        for(j = 1; j <= dy; j++)
                          DEG_optimize_Mx[j] = DEG[j];
                        }
                     if(optimize_min_Mx > MIN_Mx)
                        {
                        optimize_min_Mx = MIN_Mx;
                        for(j = 1; j <= dy; j++)
                          DEG_optimize_min_Mx[j] = DEG[j];
                        }
                     }
                  }
               }
            }
         }
      }
                     }
                  }
               }
            }
         }
      }
   fprintf(optimize, "Exhaustive Search\n");
   fprintf(optimize, "\nOptimal Orientation : [% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,%
2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,% 2lg,%
2lg]T\n",
          DEG_optimize_Mx[1], DEG_optimize_Mx[2], DEG_optimize_Mx[3],
          DEG_optimize_Mx[4], DEG_optimize_Mx[5], DEG_optimize_Mx[6],
          DEG_optimize_Mx[7], DEG_optimize_Mx[8], DEG_optimize_Mx[9],
          DEG_optimize_Mx[10],DEG_optimize_Mx[11],DEG_optimize_Mx[12],
          DEG_optimize_Mx[13],DEG_optimize_Mx[14],DEG_optimize_Mx[15],
          DEG_optimize_Mx[16],DEG_optimize_Mx[17],DEG_optimize_Mx[18],
          DEG_optimize_Mx[19],DEG_optimize_Mx[20]);
   fprintf(optimize, "Maximum moment (Nm/m) = %.4le\n\n", optimize_Mx);

 fclose(optimize);
   gotoxy(1,3); printf("Finish!!!!!!\a\a\a"); getch();
}
/*****************************End of Main Program*********************************/
```

```
//**************************Stresses Evaluation Function**************************//
void evaluate_stress(int Dy, double deg[MAXLAYERS], double sigma12[][2][3])
{
   void MAT_Q(double e1,  double e2, double u12, double g12, double q[3][3]);
   void MAT_Transform(double deg, double t[3][3]);

   void MAT3_copy(int k, double t[3][3], double T[][3][3]);
   void MAT3_Inverse(double u[3][3], double r[3][3]);
   void Dot_MAT3(double u[3][3], double v[3][3], double w[3][3]);
   void Dot_MAT3_vec3(double u[3][3], double v[3], double w[3]);
   void MAT3_multip(double u[3][3], double k, double r[3][3]);
   void MAT3_T(double a[3][3], double r[3][3]);
   void vec3_plus(double u[3], double v[3], int k, double w[][3]);

   int i, j, k;
   double Q[MAXLAYERS][3][3], t[3][3], T_kth[MAXLAYERS][3][3], Q_bar[MAXLAYERS][3][3],
          T_Transpose_Inverse[3][3], T_Inverse[3][3],
          A[3][3], B[3][3], D[3][3], B_prime[3][3], D_prime[3][3],
          a[3][3], b[3][3], d[3][3],
          M[3], N[3],
          epsilon_o[3], keppa_o[3],
          epsilon_xy[2][3],
          h, Mx, z[dy+1],
          temp[3][3], temp1[3][3],
          e1,e2;

   Mx = 1.0;
   //**********  Find Matrix Q_bar at Layer kth  **********//
   for(k = 1; k <= Dy; k++)
 {
      if(NA_in >= k*H/Dy){
      e1 = EC1;
         e2 = EC2;
         }
      else{
      e1 = ET1;
         e2 = ET2;
         }
      MAT_Q(e1,e2,v12,G12,Q[k]);
    MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
      MAT3_T(t,temp);
    MAT3_Inverse(temp,T_Transpose_Inverse);
      MAT3_Inverse(t,T_Inverse);
    Dot_MAT3(T_Inverse,Q[k],temp);
      Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);
 }

   h = H/Dy;
   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       {
      A[i][j] = B[i][j] = D[i][j] = 0.0;
      for(k = 1; k <= dy; k++)
          {
            A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
            B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
                *(pow((-H/2 + h*k),2) - pow((-H/2 + h*(k-1)),2)));
            D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
                *(pow((-H/2 + h*k),3) - pow((-H/2 + h*(k-1)),3)));
          }
    }

   MAT3_Inverse(A, a);
   MAT3_Inverse(B, b);
   MAT3_Inverse(D, d);

   M[0] = Mx ;  N[0]= 0.0;
   M[1] = 0.0;  N[1] = 0.0;
   M[2] = 0.0;  N[2] = 0.0;

   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
      B_prime[i][j] = D_prime[i][j] = temp[i][j] = 0.0;
```

```
   //**************D'=(D-BA^(-1)B)^(-1)**************//
   Dot_MAT3(B, a, temp);
   Dot_MAT3(temp, B, temp1);
   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++){ temp1[i][j] = D[i][j] - temp1[i][j]};
   MAT3_Inverse(temp1, D_prime);

   //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
   Dot_MAT3(a, B, temp);
   MAT3_multip(temp, -1, temp1);
   Dot_MAT3(temp1, D_prime, B_prime);
   Dot_MAT3_vec3(B_prime, M, epsilon_o);
   Dot_MAT3_vec3(D_prime, M, keppa_o);

   //*****************Determine N.A.*****************//
   //             (refer to top surface)             //

   NA_out = H/2 - epsilon_o[0]/keppa_o[0];

 //******** Determine stresses in plane 1-2 *******//
   for(k = 1; k <= Dy; k++)
    {
      Dot_MAT3(T_kth[k], Q_bar[k], temp1);
      z[k-1] = -H/2 + h*(k-1);
      z[k]   = -H/2 + h*(k);

      for(i = 0; i < 3; i++)
      {
        epsilon_xy[0][i] = epsilon_o[i] + keppa_o[i]*z[k-1] ;
        epsilon_xy[1][i] = epsilon_o[i] + keppa_o[i]*z[k]   ;
      }

      Dot_MAT3_vec3(temp1, epsilon_xy[0], sigma12[k][0]);
      Dot_MAT3_vec3(temp1, epsilon_xy[1], sigma12[k][1]);
    }
}
/************************End of Stresses Evaluation Function ************************/

/**************************Matrix Q Determination Function**************************/
void MAT_Q(double e1,  double e2, double u12, double g12, double q[3][3])
{
   double u21;

   u21 = e2/e1*u12;

   q[0][0] = e1/(1 - u12*u21);
   q[0][1] = q[1][0] = u12*e2/(1 - u12*u21);
   q[1][1] = e2/(1 - u12*u21);
   q[2][2] = g12;
   q[0][2] = q[1][2] = q[2][0] = q[2][1] = 0.0;

   return;
}
/************************End of Matrix Q Determination Function************************/

/************************Transformation Matrix ,[T], Function************************/
void MAT_Transform(double deg, double t[3][3])
{
   t[0][0] = pow(cos(M_PI/180*deg),2);
   t[0][1] = pow(sin(M_PI/180*deg),2);
   t[0][2] = 2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0] = pow(sin(M_PI/180*deg),2);
   t[1][1] = pow(cos(M_PI/180*deg),2);
   t[1][2] = -2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0] = -sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][1] = sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2] = pow(cos(M_PI/180*deg),2)-pow(sin(M_PI/180*deg),2);

   return;
}
/********************End of Transformation Matrix [T] Function **********************/
```

```
/*****************************Matrix Replication Function*****************************/

void MAT3_copy(int k, double t[3][3], double T[][3][3])
{
   int i,j;

 for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
    T[k][i][j] = t[i][j];
 return;
}
/*************************End of Matrix Replication Function***********************/

/*********************************Dot Matrix Function*********************************/
void Dot_MAT3(double u[3][3], double v[3][3], double w[3][3])
{
   int i, j;

   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       w[i][j] = u[i][0]*v[0][j] + u[i][1]*v[1][j] + u[i][2]*v[2][j];
   return;
}
/*****************************End of Dot Matrix Function ***************************/

/*********************************Dot Vector Function*********************************/
void Dot_MAT3_vec3(double u[3][3], double v[3], double w[3])
{
   int i;

 for(i = 0; i < 3; i++)
    w[i] = u[i][0]*v[0] + u[i][1]*v[1] + u[i][2]*v[2];
   return;
}
/*****************************End of Dot Vector Function ***************************/

/*********************************Inverse Matrix Function*****************************/
void MAT3_Inverse(double u[3][3], double r[3][3])
{
 double DET(double u[3][3]);
   void adj(double u[3][3], double r[3][3]);
   void MAT3_multip(double u[3][3], double k, double r[3][3]);

   int i, j;
   double temp[3][3];

   adj(u, temp);
   if(DET(u) == 0)
    {
    for(i = 0; i < 3; i++)
     for(j = 0; j < 3; j++)
          r[i][j] = 0.0;
      }
      else
     MAT3_multip(temp, 1/DET(u), r);
   return;
}
/*****************************End of Inverse Matrix Function ***********************/

/*********************************Determinant Function********************************/
double DET(double u[3][3])
{
   double temp;

 temp =   u[0][0]*u[1][1]*u[2][2] + u[0][1]*u[1][2]*u[2][0]
      + u[0][2]*u[1][0]*u[2][1] - u[2][0]*u[1][1]*u[0][2]
          - u[2][1]*u[1][2]*u[0][0] - u[2][2]*u[1][0]*u[0][1];
   return(temp);
}
/*****************************End of Determinant Function*****************************/

/*********************************Adjoint Matrix Function*****************************/
void adj(double u[3][3], double r[3][3])
```

```
{
   void MAT3_T(double a[3][3], double r[3][3]);
   void Cof(double a[3][3], double r[3][3]);

   double temp[3][3];

   Cof(u, temp);
   MAT3_T(temp, r);
   return;
}
/***************************End of Determinant Function***************************/

/*****************************Matrix Transpose Function*****************************/
void MAT3_T(double a[3][3], double r[3][3])
{
   int i, j;

 for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       r[i][j] = a[j][i];
   return;
}
/************************End of Matrix Transpose Function*************************/

/*****************************Cofactor Matrix Function*****************************/
void Cof(double a[3][3], double r[3][3])
{
   int i, j;
   double M[3][3];

   M[0][0] = a[1][1]*a[2][2] - a[2][1]*a[1][2];
   M[0][1] = a[1][0]*a[2][2] - a[2][0]*a[1][2];
   M[0][2] = a[1][0]*a[2][1] - a[2][0]*a[1][1];
   M[1][0] = a[0][1]*a[2][2] - a[2][1]*a[0][2];
   M[1][1] = a[0][0]*a[2][2] - a[2][0]*a[0][2];
   M[1][2] = a[0][0]*a[2][1] - a[2][0]*a[0][1];
   M[2][0] = a[0][1]*a[1][2] - a[1][1]*a[0][2];
   M[2][1] = a[0][0]*a[1][2] - a[1][0]*a[0][2];
   M[2][2] = a[0][0]*a[1][1] - a[1][0]*a[0][1];

 for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       r[i][j] = powl(-1, i+j)*M[i][j];
   return;
}
/************************End of Cofactor Matrix Function*************************/

/************************Scalar-Multiply-Matrix Function*************************/
void MAT3_multip(double u[3][3], double k, double r[3][3])
{
   int i, j;

   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       r[i][j] = k*u[i][j];
   return;
}
/**********************End of Scalar-Multiply-Matrix Function**********************/

/****************************Combination Vector Function*****************************/
void vec3_plus(double u[3], double v[3], int k, double w[][3])
{
   int i;

 for(i = 0; i < 3; i++)
    w[k][i] = u[i] + v[i];
   return;
}
/**********************End of Combination Vector Function ************************/

/*****************************Minimum Value Function*****************************/
double find_min(double a[2],double a_min)
{
```

```
   int i;

   a_min = 1.0e500;
 for(i = 0; i < 3; i++)
    {
      if(a[i] > 0)
      if(a_min > a[i])a_min = a[i];
    }
   return a_min;
}
/*************************End of Minimum Value Function *************************/
```

3. โปรแกรมหาการกระจายความเค้น

```
//*************************************************************************//
//                                                                         //
//                                Stress Profile                           //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

#define C1 -35.89E6         //   compression strength in the fiber direction
#define C2 -11.87E6         //   compression strength in the tangential direction
#define T1 56.684E6         //   tensile strength in the fiber direction
#define T2 2.795E6          //   tendile strength in the tangential direction
#define S12 15.985E6        //   shear strength in plane LT

#define dy 20               //   the number of layers
#define MAXLAYERS 30        //   the maximum number of layers allowance
#define H 0.020             //   thickness of OSL in meter

#define E1 28.05E9          //   MOE in the longitudinal (fiber:L) direction
#define E2 2.079E9          //   MOE in the transverse (tangential:T) direction
#define v12 0.35            //   possion's ratio in plane 1-2(LT)
#define G12 2.244E9         //   modulus of rigidity in plane 1-2(LT)

#define f_name "StressProfile.txt"

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy,double Mx,int deg[MAXLAYERS],double
sigma12[][2][3],double sigmaxy[][2][3],double exy[][2][3]);
   double find_min(double a[dy],double a_min);
   void display(int DEG[MAXLAYERS],double min_mx);
   int i,j,k,l,m;
    int DEG[MAXLAYERS], DEG_MAX_Mx[MAXLAYERS], DEG_optimize_Mx[MAXLAYERS],
DEG_optimize_min_Mx[MAXLAYERS], DEG_TMP[MAXLAYERS];
   double mx[2],min_mx[MAXLAYERS],MIN_Mx,MAX_Mx,optimize_Mx,optimize_min_Mx,
SIGMA12[MAXLAYERS][2][3],SIGMAxy[MAXLAYERS][2][3],Exy[MAXLAYERS][2][3], X,Y;

   clrscr();
   optimize=fopen(f_name,"w+");

   for(i=1;i<=dy/2;i++)
    {
      printf("Enter Angle Layer %d = ",i);
      scanf("%d",&DEG_optimize_Mx[i]);
      DEG_optimize_Mx[dy-(i-1)] = -DEG_optimize_Mx[i];    /****** Anti-sym ******/
      min_mx[i] = 0.0;
      }
   for(i=1;i<=dy;i++) fprintf(optimize,"%d ",DEG_optimize_Mx[i]);
   printf("Optimize Moment = ");
   scanf("%lf",&optimize_Mx);
   fprintf(optimize,"\nOptimum Moment = % .2lf\n",optimize_Mx);
```

```
   evaluate_stress(dy,optimize_Mx,DEG_optimize_Mx,SIGMA12,SIGMAxy,Exy);

   printf("s1 s2 s12 top : s1 s2 s12 bottom\n\n");
   fprintf(optimize,"\n       s1          s2         s12     \n\n");
   for(k=1;k<=dy;k++)
      {
      printf("% 11.4lg  % 11.4lg  % 11.4lg : % 11.4lg  % 11.4lg  % 11.4lg\n",
            SIGMA12[k][0][0], SIGMA12[k][0][1], SIGMA12[k][0][2],
            SIGMA12[k][1][0], SIGMA12[k][1][1], SIGMA12[k][1][2]);
      fprintf(optimize,"% 11.4lg  % 11.4lg  % 11.4lg \n",
         SIGMA12[k][0][0], SIGMA12[k][0][1], SIGMA12[k][0][2]);
      fprintf(optimize,"% 11.4lg  % 11.4lg  % 11.4lg \n",
         SIGMA12[k][1][0], SIGMA12[k][1][1], SIGMA12[k][1][2]);
      }
 printf("sx sy sxy top : sx sy sxy bottom\n\n");
   fprintf(optimize,"\n       sx          sy         sxy     \n\n");
   for(k=1;k<=dy;k++)
      {
      printf("% 11.4lg  % 11.4lg  % 11.4lg : % 11.4lg  % 11.4lg  % 11.4lg\n",
            SIGMAxy[k][0][0], SIGMAxy[k][0][1], SIGMAxy[k][0][2],
            SIGMAxy[k][1][0], SIGMAxy[k][1][1], SIGMAxy[k][1][2]);
      fprintf(optimize,"% 11.4lg  % 11.4lg  % 11.4lg \n",
         SIGMAxy[k][0][0], SIGMAxy[k][0][1], SIGMAxy[k][0][2]);
      fprintf(optimize,"% 11.4lg  % 11.4lg  % 11.4lg \n",
         SIGMAxy[k][1][0], SIGMAxy[k][1][1], SIGMAxy[k][1][2]);
      }
   fprintf(optimize,"\n       ex          ey         exy     \n\n");
   for(k=1;k<=dy;k++)
      {
      printf("% 11.4lg  % 11.4lg  % 11.4lg : % 11.4lg  % 11.4lg  % 11.4lg\n",
            Exy[k][0][0], Exy[k][0][1], Exy[k][0][2],
            Exy[k][1][0], Exy[k][1][1], Exy[k][1][2]);
      fprintf(optimize,"% 11.4lg  % 11.4lg  % 11.4lg \n",
         Exy[k][0][0], Exy[k][0][1], Exy[k][0][2]);
      fprintf(optimize,"% 11.4lg  % 11.4lg  % 11.4lg \n",
         Exy[k][1][0], Exy[k][1][1], Exy[k][1][2]);
      }
 fclose(optimize);
   printf("Finish!!!!!!\a\a\a\a");getch();
}
/*****************************End of Main Program*********************************/

//**************************Stresses Evaluation Function**************************//
void evaluate_stress(int Dy,double Mx,int deg[MAXLAYERS],double sigma12[][2][3],double
sigmaxy[][2][3],double exy[][2][3])
{
   void MAT_Q(double e1,double e2,double u12, double g12,double q[3][3]);
   void MAT_Transform(double deg,double t[3][3]);

   void MAT3_copy(int k,double t[3][3],double T[][3][3]);
   void MAT3_Inverse(double u[3][3],double r[3][3]);
   void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3]);
   void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);
   void MAT3_T(double a[3][3],double r[3][3]);
   void vec3_plus(double u[3],double v[3],int k,double w[][3]);

   int i,j,k;
   double Q[3][3],t[3][3],T_kth[MAXLAYERS][3][3],Q_bar[MAXLAYERS][3][3],
          T_Transpose_Inverse[3][3],T_Inverse[3][3],
          A[3][3],B[3][3],D[3][3],B_prime[3][3],D_prime[3][3],
          a[3][3],b[3][3],d[3][3],
          M[3],N[3],
          epsilon_o[3],keppa_o[3],
          epsilon_xy[2][3],
          h,z[dy+1],
          temp[3][3],temp1[3][3];

   MAT_Q(E1,E2,v12,G12,Q);
   for(k=1;k<=Dy;k++)
 {
      MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
      MAT3_T(t,temp);
```

```
      MAT3_Inverse(temp,T_Transpose_Inverse);
      MAT3_Inverse(t,T_Inverse);
      Dot_MAT3(T_Inverse,Q,temp);
      Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);
 }

   h=H/Dy;
   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      {  A[i][j]=B[i][j]=D[i][j]=0.0;
         for(k=1;k<=dy;k++)
           {
            A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
            B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
                *(powl((-H/2 + h*k),2) - powl((-H/2 + h*(k-1)),2)));
            D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
                *(powl((-H/2 + h*k),3) - powl((-H/2 + h*(k-1)),3)));
           }
      }

   MAT3_Inverse(A,a);
   MAT3_Inverse(B,b);
   MAT3_Inverse(D,d);

   M[0]= Mx  ; M[1] = 0.0 ; M[2] = 0.0 ;
   N[0]= 0.0 ; N[1] = 0.0 ; N[2] = 0.0 ;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      B_prime[i][j]=D_prime[i][j]=temp[i][j]=0.0;

   //**************D'=(D-BA^(-1)B)^(-1)**************//
   Dot_MAT3(B,a,temp);
   Dot_MAT3(temp,B,temp1);
   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
     temp1[i][j]=D[i][j]-temp1[i][j];
   MAT3_Inverse(temp1,D_prime);

   //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
   Dot_MAT3(a,B,temp);
   MAT3_multip(temp,-1,temp1);
   Dot_MAT3(temp1,D_prime,B_prime);
   Dot_MAT3_vec3(B_prime,M,epsilon_o);
   Dot_MAT3_vec3(D_prime,M,keppa_o);

   //******** Determine stresses in plane 1-2 *******//
   for(k=1;k<=Dy;k++)
   {
      Dot_MAT3(T_kth[k],Q_bar[k],temp1);
      z[k-1]=-H/2+h*(k-1);
      z[k]=-H/2+h*(k);

      for(i=0;i<3;i++)
      {
       exy[k][0][i]=epsilon_o[i] + keppa_o[i]*z[k-1] ;
       exy[k][1][i]=epsilon_o[i] + keppa_o[i]*z[k]   ;
      }

      Dot_MAT3_vec3(temp1,exy[k][0],sigma12[k][0]);
      Dot_MAT3_vec3(temp1,exy[k][1],sigma12[k][1]);
      Dot_MAT3_vec3(Q_bar[k],exy[k][0],sigmaxy[k][0]);
      Dot_MAT3_vec3(Q_bar[k],exy[k][1],sigmaxy[k][1]);
   }
}
/***********************End of Stresses Evaluation Function ***********************/

/**************************Matrix Q Determination Function************************/
void MAT_Q(double e1, double e2, double u12, double g12,double q[3][3])
{
   double u21;

 u21=e2/e1*u12;
```

```
   q[0][0]=e1/(1-u12*u21);
   q[0][1]=q[1][0]=u12*e2/(1-u12*u21);
   q[1][1]=e2/(1-u12*u21);
   q[2][2]=g12;
   q[0][2]=q[1][2]=q[2][0]=q[2][1]=0.0;

   return;
}
/**********************End of Matrix Q Determination Function**********************/

/**********************Transformation Matrix ,[T], Function**********************/
void MAT_Transform(double deg,double t[3][3])
{
   t[0][0]=pow(cos(M_PI/180*deg),2);
   t[0][1]=pow(sin(M_PI/180*deg),2);
   t[0][2]=2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0]=pow(sin(M_PI/180*deg),2);
   t[1][1]=pow(cos(M_PI/180*deg),2);
   t[1][2]=-2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0]=-sin(M_PI/180*deg)*cosl(M_PI/180*deg);
   t[2][1]=sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2]=pow(cos(M_PI/180*deg),2)-pow(sin(M_PI/180*deg),2);

   return;
}
/*******************End of Transformation Matrix [T] Function **********************/

/*****************************Matrix Replication Function*****************************/
void MAT3_copy(int k,double t[3][3],double T[][3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    T[k][i][j]=t[i][j];
   return;
}
/**************************End of Matrix Replication Function**********************/

/*********************************Dot Matrix Function*********************************/
void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       w[i][j]=u[i][0]*v[0][j]+u[i][1]*v[1][j]+u[i][2]*v[2][j];
   return;
}
/*****************************End of Dot Matrix Function **************************/

/*********************************Dot Vector Function*********************************/
void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3])
{
   int i;

 for(i=0;i<3;i++)
    w[i]=u[i][0]*v[0]+u[i][1]*v[1]+u[i][2]*v[2];
   return;
}
/*****************************End of Dot Vector Function **************************/

/*********************************Inverse Matrix Function*****************************/
void MAT3_Inverse(double u[3][3],double r[3][3])
{
 double DET(double u[3][3]);
   void adj(double u[3][3],double r[3][3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);

   int i,j;
   double temp[3][3];
```

```
   adj(u,temp);
   if(DET(u)==0){
    for(i=0;i<3;i++)
     for(j=0;j<3;j++)
          r[i][j]=0.0;
      }else
   MAT3_multip(temp,1/DET(u),r);
   return;
}
/*****************************End of Inverse Matrix Function ************************/

/*********************************Determinant Function*******************************/
double DET(double u[3][3])
{
   double temp;

  temp=u[0][0]*u[1][1]*u[2][2]+u[0][1]*u[1][2]*u[2][0]+u[0][2]*u[1][0]*u[2][1]
     -(u[2][0]*u[1][1]*u[0][2]+u[2][1]*u[1][2]*u[0][0]+u[2][2]*u[1][0]*u[0][1]);
  return(temp);
}
/*****************************End of Determinant Function*****************************/

/*****************************Adjoint Matrix Function*******************************/

void adj(double u[3][3],double r[3][3])
{
   void MAT3_T(double a[3][3],double r[3][3]);
   void Cof(double a[3][3],double r[3][3]);

   double temp[3][3];

   Cof(u,temp);
   MAT3_T(temp,r);
   return;
}
/*****************************End of Determinant Function*****************************/

/*****************************Matrix Transpose Function*******************************/
void MAT3_T(double a[3][3],double r[3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=a[j][i];
   return;
}
/************************End of Matrix Transpose Function*************************/

/*****************************Cofactor Matrix Function*******************************/
void Cof(double a[3][3],double r[3][3])
{
   int i,j;
   double M[3][3];

   M[0][0]=a[1][1]*a[2][2]-a[2][1]*a[1][2];
   M[0][1]=a[1][0]*a[2][2]-a[2][0]*a[1][2];
   M[0][2]=a[1][0]*a[2][1]-a[2][0]*a[1][1];
   M[1][0]=a[0][1]*a[2][2]-a[2][1]*a[0][2];
   M[1][1]=a[0][0]*a[2][2]-a[2][0]*a[0][2];
   M[1][2]=a[0][0]*a[2][1]-a[2][0]*a[0][1];
   M[2][0]=a[0][1]*a[1][2]-a[1][1]*a[0][2];
   M[2][1]=a[0][0]*a[1][2]-a[1][0]*a[0][2];
   M[2][2]=a[0][0]*a[1][1]-a[1][0]*a[0][1];

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=powl(-1,i+j)*M[i][j];
   return;
}
/************************End of Cofactor Matrix Function*************************/

/************************Scalar-Multiply-Matrix Function*************************/
```

```
void MAT3_multip(double u[3][3],double k,double r[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=k*u[i][j];
   return;
}
/**********************End of Scalar-Multiply-Matrix Function**********************/

/**************************Combination Vector Function*****************************/
void vec3_plus(double u[3],double v[3],int k,double w[][3])
{
   int i;

 for(i=0;i<3;i++)
    w[k][i]=u[i]+v[i];
   return;
}
/**********************End of Combination Vector Function ************************/

/*****************************Minimum Value Function*******************************/
double find_min(double a[2],double a_min)
{
   int i;

   a_min=1.0e500;
 for(i=0;i<3;i++)
   {
      if(a[i]>0)
      if(a_min>a[i])a_min=a[i];
   }
   return a_min;
}
/**************************End of Minimum Value Function ***************************/
```

4. โปรแกรมวิเคราะห์ความไวต่อมุม

4.1 ตามเกณฑ์ของไซ-ฮิลล์

```
//*************************************************************************//
//                                                                         //
//                          Angle Sensitivity Analysis                     //
//                       According to Tsai-Hill Criteion                   //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

#define C1 -35.89E6        //   compression strength in the fiber direction
#define C2 -11.87E6        //   compression strength in the tangential direction
#define T1 56.684E6        //   tensile strength in the fiber direction
#define T2 2.795E6         //   tendile strength in the tangential direction
#define S12 15.985E6       //   shear strength in plane LT

#define dy 20              //   the number of layers
#define MAXLAYERS 30       //   the maximum number of layers allowance
#define H 0.020            //   thickness of OSL in meter

#define E1 28.05E9         //   MOE in the longitudinal (fiber:L) direction
#define E2 2.079E9         //   MOE in the transverse (tangential:T) direction
#define v12 0.35           //   possion's ratio in plane 1-2(LT)
#define G12 2.244E9        //   modulus of rigidity in plane 1-2(LT)

#define f_name "VarryEachLayerTHC.txt"

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy,int deg[MAXLAYERS],double sigma12[][2][3]);
   double find_min(double a[dy],double a_min);
   void display(int DEG[MAXLAYERS],double min_mx);

   int i,j,k,l,m;
    int DEG[MAXLAYERS], DEG_MAX_Mx[MAXLAYERS], DEG_optimize_Mx[MAXLAYERS],
DEG_min_Mx[MAXLAYERS], DEG_TMP[MAXLAYERS];
   double mx[2],min_mx[MAXLAYERS],MIN_Mx,optimize_MAX_Mx,optimize_Mx,optimize_min_Mx,
SIGMA[MAXLAYERS][2][3], X,Y;

   clrscr();
   optimize=fopen(f_name,"w+");

   for(i=1;i<=dy/2;i++)
    {
      printf("Enter Angle Layer %d = ",i);
      scanf("%d",&DEG_optimize_Mx[i]);
      DEG_optimize_Mx[dy-(i-1)] = -DEG_optimize_Mx[i];    /****** Anti-sym ******/
      min_mx[i] = 0.0;
      }
   printf("Optimize Moment = ");
   scanf("%lf",&optimize_Mx);

   for(l=1;l<=dy/2;l++)
    {
      optimize_MAX_Mx = optimize_Mx;
    optimize_min_Mx = optimize_Mx;
      for(m=1;m<=dy;m++)
      {
         DEG[m] = DEG_optimize_Mx[m];
         DEG_TMP[m] = DEG_optimize_Mx[m];
         DEG_MAX_Mx[m] = DEG_optimize_Mx[m];
         DEG_min_Mx[m] = DEG_optimize_Mx[m];
      }
```

```
    for(DEG[l]=DEG_TMP[l]-5;DEG[l]<=DEG_TMP[l]+5;DEG[l]=DEG[l]+1)
     {
      DEG[dy-(l-1)]=(-1)*DEG[l];

          evaluate_stress(dy,DEG,SIGMA);
                  for(i=1;i<=dy;i++)
                   {
                     if(SIGMA[i][0][0]>0) X = T1; else X = C1;
                     if(SIGMA[i][1][0]>0) Y = T2; else Y = C2;

                     mx[0] = sqrt(1.0/( (SIGMA[i][0][0]*SIGMA[i][0][0])/(X*X) -
                             (SIGMA[i][0][0]*SIGMA[i][0][1])/(X*X) +
                              (SIGMA[i][0][1]*SIGMA[i][0][1])/(Y*Y) +
                              (SIGMA[i][0][2]*SIGMA[i][0][2])/(S12*S12) )
                            );
              if(SIGMA[i][0][1]>0) X = T1; else X = C1;
                     if(SIGMA[i][1][1]>0) Y = T2; else Y = C2;

                     mx[1] = sqrt(1.0/( (SIGMA[i][1][0]*SIGMA[i][1][0])/(X*X) -
                             (SIGMA[i][1][0]*SIGMA[i][1][1])/(X*X) +
                              (SIGMA[i][1][1]*SIGMA[i][1][1])/(Y*Y) +
                              (SIGMA[i][1][2]*SIGMA[i][1][2])/(S12*S12) )
                            );
          min_mx[i] = mx[0];
               if(min_mx[i] > mx[1]) min_mx[i] = mx[1]; // find min Mx in each layer //
          }
                    MIN_Mx=min_mx[1];
                    for(i=1;i<=dy;i++)
                    if(MIN_Mx>min_mx[i])MIN_Mx=min_mx[i];

                    if(optimize_MAX_Mx<MIN_Mx)
                       {
                       optimize_MAX_Mx = MIN_Mx;
                       for(j=1;j<=dy;j++)
                          DEG_MAX_Mx[j] = DEG[j];
                       }
                    if(optimize_min_Mx>MIN_Mx)
                       {
                       optimize_min_Mx=MIN_Mx;
                       for(j=1;j<=dy;j++)
                          DEG_min_Mx[j]=DEG[j];
                       }
    }

    fprintf(optimize,"Varries Layer %d\n",l);
    fprintf(optimize,"\nOptimal Orientation : [% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,%
2d,% 2d,% 2d]T\n", DEG_MAX_Mx[1], DEG_MAX_Mx[2], DEG_MAX_Mx[3], DEG_MAX_Mx[4],
                   DEG_MAX_Mx[5], DEG_MAX_Mx[6], DEG_MAX_Mx[7], DEG_MAX_Mx[8],
                   DEG_MAX_Mx[9], DEG_MAX_Mx[10]);
    fprintf(optimize,"Maximum moment (Nm/m) = %.4le\n\n",optimize_Mx);
    fprintf(optimize,"\nOptimal Orientation : [% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,%
2d,% 2d,% 2d]T\n", DEG_min_Mx[1], DEG_min_Mx[2], DEG_min_Mx[3], DEG_min_Mx[4],
                   DEG_min_Mx[5], DEG_min_Mx[6], DEG_min_Mx[7], DEG_min_Mx[8],
                   DEG_min_Mx[9], DEG_min_Mx[10]);
    fprintf(optimize,"Minimum moment (Nm/m) = %.4le\n\n",optimize_min_Mx);
      fprintf(optimize,"*************************************************\n");
    }
 fclose(optimize);
   printf("Finish!!!!!!\a\a\a\a");getch();
}
/*****************************End of Main Program*********************************/

//**************************Stresses Evaluation Function*************************//
void evaluate_stress(int Dy,int deg[MAXLAYERS],double sigma12[][2][3])
{
   void MAT_Q(double e1,double e2,double u12, double g12,double q[3][3]);
   void MAT_Transform(double deg,double t[3][3]);

   void MAT3_copy(int k,double t[3][3],double T[][3][3]);
   void MAT3_Inverse(double u[3][3],double r[3][3]);
   void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3]);
   void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);
```

```
  void MAT3_T(double a[3][3],double r[3][3]);
  void vec3_plus(double u[3],double v[3],int k,double w[][3]);

  int i,j,k;
  double Q[3][3],t[3][3],T_kth[MAXLAYERS][3][3],Q_bar[MAXLAYERS][3][3],
         T_Transpose_Inverse[3][3],T_Inverse[3][3],
         A[3][3],B[3][3],D[3][3],B_prime[3][3],D_prime[3][3],
         a[3][3],b[3][3],d[3][3],
         M[3],N[3],
         epsilon_o[3],keppa_o[3],
         epsilon_xy[2][3],
         h,Mx,z[dy+1],
         temp[3][3],temp1[3][3];

  Mx = 1.0;
  MAT_Q(E1,E2,v12,G12,Q);
  for(k=1;k<=Dy;k++)
{
   MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
   MAT3_T(t,temp);
   MAT3_Inverse(temp,T_Transpose_Inverse);
   MAT3_Inverse(t,T_Inverse);
   Dot_MAT3(T_Inverse,Q,temp);
   Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);
}

  h=H/Dy;
  for(i=0;i<3;i++)
   for(j=0;j<3;j++)
      {  A[i][j]=B[i][j]=D[i][j]=0.0;
      for(k=1;k<=dy;k++)
         {
            A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
            B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
                *(powl((-H/2 + h*k),2) - powl((-H/2 + h*(k-1)),2)));
            D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
                *(powl((-H/2 + h*k),3) - powl((-H/2 + h*(k-1)),3)));
         }
     }

  MAT3_Inverse(A,a);
  MAT3_Inverse(B,b);
  MAT3_Inverse(D,d);

  M[0]= Mx  ; M[1] = 0.0 ; M[2] = 0.0 ;
  N[0]= 0.0 ; N[1] = 0.0 ; N[2] = 0.0 ;

  for(i=0;i<3;i++)
   for(j=0;j<3;j++)
      B_prime[i][j]=D_prime[i][j]=temp[i][j]=0.0;

  //**************D'=(D-BA^(-1)B)^(-1)**************//
  Dot_MAT3(B,a,temp);
  Dot_MAT3(temp,B,temp1);
  for(i=0;i<3;i++)
   for(j=0;j<3;j++)
    temp1[i][j]=D[i][j]-temp1[i][j];
  MAT3_Inverse(temp1,D_prime);

  //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
  Dot_MAT3(a,B,temp);
  MAT3_multip(temp,-1,temp1);
  Dot_MAT3(temp1,D_prime,B_prime);
  Dot_MAT3_vec3(B_prime,M,epsilon_o);
  Dot_MAT3_vec3(D_prime,M,keppa_o);

  //******** Determine stresses in plane 1-2 *******//
  for(k=1;k<=Dy;k++)
  {
   Dot_MAT3(T_kth[k],Q_bar[k],temp1);
      z[k-1]=-H/2+h*(k-1);
      z[k]=-H/2+h*(k);
```

```
      for(i=0;i<3;i++)
      {
       epsilon_xy[0][i]=epsilon_o[i] + keppa_o[i]*z[k-1] ;
       epsilon_xy[1][i]=epsilon_o[i] + keppa_o[i]*z[k]   ;
      }
      Dot_MAT3_vec3(temp1,epsilon_xy[0],sigma12[k][0]);
      Dot_MAT3_vec3(temp1,epsilon_xy[1],sigma12[k][1]);
   }

   //****************** Display ******************//
   printf("s1 s2 s12 top : s1 s2 s12 bottom\n\n");
   for(k=1;k<=dy;k++)
      {printf("% 11.4Le  % 11.4Le  % 11.4Le : % 11.4Le  % 11.4Le  % 11.4Le\n",
             sigma12[k][0][0], sigma12[k][0][1], sigma12[k][0][2],
             sigma12[k][1][0], sigma12[k][1][1], sigma12[k][1][2]);
      }getch();*/
}
/***********************End of Stresses Evaluation Function ***********************/

/***************************Matrix Q Determination Function*************************/
void MAT_Q(double e1, double e2, double u12, double g12,double q[3][3])
{
   double u21;

   u21=e2/e1*u12;

   q[0][0]=e1/(1-u12*u21);
   q[0][1]=q[1][0]=u12*e2/(1-u12*u21);
   q[1][1]=e2/(1-u12*u21);
   q[2][2]=g12;
   q[0][2]=q[1][2]=q[2][0]=q[2][1]=0.0;

   return;
}
/***********************End of Matrix Q Determination Function***********************/

/***********************Transformation Matrix ,[T], Function***********************/
void MAT_Transform(double deg,double t[3][3])
{
   t[0][0]=pow(cos(M_PI/180*deg),2);
   t[0][1]=pow(sin(M_PI/180*deg),2);
   t[0][2]=2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0]=pow(sin(M_PI/180*deg),2);
   t[1][1]=pow(cos(M_PI/180*deg),2);
   t[1][2]=-2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0]=-sin(M_PI/180*deg)*cosl(M_PI/180*deg);
   t[2][1]=sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2]=pow(cos(M_PI/180*deg),2)-pow(sin(M_PI/180*deg),2);

   return;
}
/*********************End of Transformation Matrix [T] Function **********************/

/*****************************Matrix Replication Function*****************************/
void MAT3_copy(int k,double t[3][3],double T[][3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    T[k][i][j]=t[i][j];
 return;
}
/*************************End of Matrix Replication Function************************/

/*******************************Dot Matrix Function*******************************/
void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
      w[i][j]=u[i][0]*v[0][j]+u[i][1]*v[1][j]+u[i][2]*v[2][j];
```

```
   return;
}
/*****************************End of Dot Matrix Function *************************/

/*******************************Dot Vector Function*******************************/
void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3])
{
   int i;

 for(i=0;i<3;i++)
    w[i]=u[i][0]*v[0]+u[i][1]*v[1]+u[i][2]*v[2];
 return;
}
/*****************************End of Dot Vector Function *************************/

/*******************************Inverse Matrix Function***************************/
void MAT3_Inverse(double u[3][3],double r[3][3])
{
   double DET(double u[3][3]);
   void adj(double u[3][3],double r[3][3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);

   int i,j;
   double temp[3][3];

   adj(u,temp);
   if(DET(u)==0){
    for(i=0;i<3;i++)
      for(j=0;j<3;j++)
           r[i][j]=0.0;
       }else
   MAT3_multip(temp,1/DET(u),r);
   return;
}
/*****************************End of Inverse Matrix Function *********************/

/*******************************Determinant Function******************************/
double DET(double u[3][3])
{
   double temp;

 temp=u[0][0]*u[1][1]*u[2][2]+u[0][1]*u[1][2]*u[2][0]+u[0][2]*u[1][0]*u[2][1]
     -(u[2][0]*u[1][1]*u[0][2]+u[2][1]*u[1][2]*u[0][0]+u[2][2]*u[1][0]*u[0][1]);
   return(temp);
}
/*****************************End of Determinant Function*************************/

/*******************************Adjoint Matrix Function***************************/
void adj(double u[3][3],double r[3][3])
{
   void MAT3_T(double a[3][3],double r[3][3]);
   void Cof(double a[3][3],double r[3][3]);

   double temp[3][3];

   Cof(u,temp);
   MAT3_T(temp,r);
   return;
}
/*****************************End of Determinant Function*************************/

/*******************************Matrix Transpose Function*************************/
void MAT3_T(double a[3][3],double r[3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=a[j][i];
   return;
}
/*************************End of Matrix Transpose Function*************************/
```

```
/*****************************Cofactor Matrix Function*****************************/
void Cof(double a[3][3],double r[3][3])
{
   int i,j;
   double M[3][3];

   M[0][0]=a[1][1]*a[2][2]-a[2][1]*a[1][2];
   M[0][1]=a[1][0]*a[2][2]-a[2][0]*a[1][2];
   M[0][2]=a[1][0]*a[2][1]-a[2][0]*a[1][1];
   M[1][0]=a[0][1]*a[2][2]-a[2][1]*a[0][2];
   M[1][1]=a[0][0]*a[2][2]-a[2][0]*a[0][2];
   M[1][2]=a[0][0]*a[2][1]-a[2][0]*a[0][1];
   M[2][0]=a[0][1]*a[1][2]-a[1][1]*a[0][2];
   M[2][1]=a[0][0]*a[1][2]-a[1][0]*a[0][2];
   M[2][2]=a[0][0]*a[1][1]-a[1][0]*a[0][1];

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=powl(-1,i+j)*M[i][j];
 return;
}
/************************End of Cofactor Matrix Function**************************/

/************************Scalar-Multiply-Matrix Function**************************/
void MAT3_multip(double u[3][3],double k,double r[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=k*u[i][j];
   return;
}
/**********************End of Scalar-Multiply-Matrix Function**********************/

/***************************Combination Vector Function*****************************/
void vec3_plus(double u[3],double v[3],int k,double w[][3])
{
   int i;

 for(i=0;i<3;i++)
    w[k][i]=u[i]+v[i];
   return;
}
/***********************End of Combination Vector Function ************************/

/*******************************Minimum Value Function********************************/
double find_min(double a[2],double a_min)
{
   int i;

   a_min=1.0e500;
 for(i=0;i<3;i++)
   {
      if(a[i]>0)
      if(a_min>a[i])a_min=a[i];
   }
   return a_min;
}
/*************************End of Minimum Value Function *************************/
```

4.2 ตามเกณฑ์ความเค้นสูงสุด

```
//*************************************************************************//
//                                                                         //
//                          Angle Sensitivity Analysis                     //
//                     According to Maximum Stress Criteion                //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

#define C1 -35.89E6        //   compression strength in the fiber direction
#define C2 -11.87E6        //   compression strength in the tangential direction
#define T1 56.684E6        //   tensile strength in the fiber direction
#define T2 2.795E6         //   tendile strength in the tangential direction
#define S12 15.985E6       //   shear strength in plane LT

#define dy 20              //   the number of layers
#define MAXLAYERS 30       //   the maximum number of layers allowance
#define H 0.020            //   thickness of OSL in meter

#define E1 28.05E9         //   MOE in the longitudinal (fiber:L) direction
#define E2 2.079E9         //   MOE in the transverse (tangential:T) direction
#define v12 0.35           //   possion's ratio in plane 1-2(LT)
#define G12 2.244E9        //   modulus of rigidity in plane 1-2(LT)

#define f_name "VarryEachLayerMSC.txt"

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy,int deg[MAXLAYERS],double sigma12[][2][3]);
   double find_min(double a[dy],double a_min);
   void display(int DEG[MAXLAYERS],double min_mx);

   int i,j,k,l,m;
   int DEG[MAXLAYERS], DEG_MAX_Mx[MAXLAYERS], DEG_optimize_Mx[MAXLAYERS],
DEG_min_Mx[MAXLAYERS], DEG_TMP[MAXLAYERS];
   double mx[2],min_mx[MAXLAYERS],MIN_Mx,optimize_MAX_Mx,optimize_Mx,optimize_min_Mx,
SIGMA[MAXLAYERS][2][3], X,Y;

   clrscr();
   optimize=fopen(f_name,"w+");

   for(i=1;i<=dy/2;i++)
    {
      printf("Enter Angle Layer %d = ",i);
      scanf("%d",&DEG_optimize_Mx[i]);
      DEG_optimize_Mx[dy-(i-1)] = -DEG_optimize_Mx[i];    /****** Anti-sym ******/
      min_mx[i] = 0.0;
      }
   printf("Optimize Moment = ");
   scanf("%lf",&optimize_Mx);

   for(l=1;l<=dy/2;l++)
    {
      optimize_MAX_Mx = optimize_Mx;
      optimize_min_Mx = optimize_Mx;
      for(m=1;m<=dy;m++)
      {
         DEG[m] = DEG_optimize_Mx[m];
         DEG_MAX_Mx[m] = DEG_optimize_Mx[m];
         DEG_min_Mx[m] = DEG_optimize_Mx[m];
      }

    for(DEG[l]=DEG_optimize_Mx[l]-5;DEG[l]<=DEG_optimize_Mx[l]+5;DEG[l]=DEG[l]+1)
     {
```

```
      DEG[dy-(l-1)]=(-1)*DEG[l];

          evaluate_stress(dy,DEG,SIGMA);
                  for(i = 1; i <= dy; i++)
                   {
           if(SIGMA[i][0][0]==0)mx[0]=1e300;   //*******************************//
           else mx[0]=(C1/SIGMA[i][0][0]);     //                               //
           if(SIGMA[i][1][0]==0)mx[1]=1e300;   //                               //
           else mx[1]=(C1/SIGMA[i][1][0]);     //                               //
                                               //                               //
           if(SIGMA[i][0][0]==0)mx[2]=1e300;   //       find Maximum Mx in      //
           else mx[2]=(T1/SIGMA[i][0][0]);     //         each failue mode      //
           if(SIGMA[i][1][0]==0)mx[3]=1e300;   //                               //
           else mx[3]=(T1/SIGMA[i][1][0]);     //                               //
                                               //                               //
           if(SIGMA[i][0][1]==0)mx[4]=1e300;   //                               //
           else mx[4]=(C2/SIGMA[i][0][1]);     //                               //
           if(SIGMA[i][1][1]==0)mx[5]=1e300;   //                               //
           else mx[5]=(C2/SIGMA[i][1][1]);     // ( if  a stress is zero then   //
                                               //                               //
           if(SIGMA[i][0][1]==0)mx[6]=1e300;   //      Mx will be set to        //
           else mx[6]=(T2/SIGMA[i][0][1]);     //         enormous value      ) //
           if(SIGMA[i][1][1]==0)mx[7]=1e300;   //                               //
           else mx[7]=(T2/SIGMA[i][1][1]);     //                               //
                                               //                               //
           if(SIGMA[i][0][2]==0)mx[8]=1e300;   //                               //
           else mx[8]=(S12/SIGMA[i][0][2]);    //                               //
           if(SIGMA[i][1][2]==0)mx[9]=1e300;   //                              //
           else mx[9]=(S12/SIGMA[i][1][2]);    //*******************************//

            min_mx[i]=find_min(mx,min_mx[i]);
                  }
                  MIN_Mx = min_mx[1];
                  for(i = 1; i <= dy ; i++)
                  if(MIN_Mx > min_mx[i])MIN_Mx = min_mx[i];

                  if(optimize_MAX_Mx<MIN_Mx)
                     {
                     optimize_MAX_Mx = MIN_Mx;
                     for(j = 1; j <= dy; j++)
                        DEG_MAX_Mx[j] = DEG[j];
                     }
                  if(optimize_min_Mx > MIN_Mx)
                     {
                     optimize_min_Mx = MIN_Mx;
                     for(j = 1; j <= dy; j++)
                        DEG_min_Mx[j] = DEG[j];
                     }
    }

    fprintf(optimize,"Varries Layer %d\n",l);
    fprintf(optimize,"\nOptimal Orientation : [% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,%
2d,% 2d,% 2d]T\n", DEG_MAX_Mx[1], DEG_MAX_Mx[2], DEG_MAX_Mx[3], DEG_MAX_Mx[4],
                   DEG_MAX_Mx[5], DEG_MAX_Mx[6], DEG_MAX_Mx[7], DEG_MAX_Mx[8],
                   DEG_MAX_Mx[9], DEG_MAX_Mx[10]);
    fprintf(optimize,"Maximum moment (Nm/m) = %.4le\n\n",optimize_Mx);
    fprintf(optimize,"\nOptimal Orientation : [% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,% 2d,%
2d,% 2d,% 2d]T\n", DEG_min_Mx[1], DEG_min_Mx[2], DEG_min_Mx[3], DEG_min_Mx[4],
                   DEG_min_Mx[5], DEG_min_Mx[6], DEG_min_Mx[7], DEG_min_Mx[8],
                   DEG_min_Mx[9], DEG_min_Mx[10]);
    fprintf(optimize,"Minimum moment (Nm/m) = %.4le\n\n",optimize_min_Mx);
    fprintf(optimize,"*************************************************\n");
    }
   fclose(optimize);
   printf("Finish!!!!!!\a\a\a\a");getch();
}
/*****************************End of Main Program*********************************/

//*************************Stresses Evaluation Function*************************//
void evaluate_stress(int Dy,int deg[MAXLAYERS],double sigma12[][2][3])
{
   void MAT_Q(double e1,double e2,double u12, double g12,double q[3][3]);
   void MAT_Transform(double deg,double t[3][3]);
```

```
  void MAT3_copy(int k,double t[3][3],double T[][3][3]);
  void MAT3_Inverse(double u[3][3],double r[3][3]);
  void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3]);
  void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3]);
  void MAT3_multip(double u[3][3],double k,double r[3][3]);
   void MAT3_T(double a[3][3],double r[3][3]);
  void vec3_plus(double u[3],double v[3],int k,double w[][3]);

  int i,j,k;
  double Q[3][3],t[3][3],T_kth[MAXLAYERS][3][3],Q_bar[MAXLAYERS][3][3],
         T_Transpose_Inverse[3][3],T_Inverse[3][3],
         A[3][3],B[3][3],D[3][3],B_prime[3][3],D_prime[3][3],
         a[3][3],b[3][3],d[3][3],
         M[3],N[3],
         epsilon_o[3],keppa_o[3],
         epsilon_xy[2][3],
         h,Mx,z[dy+1],
         temp[3][3],temp1[3][3];

  Mx = 1.0;
  MAT_Q(E1,E2,v12,G12,Q);
  for(k=1;k<=Dy;k++)
{
   MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
   MAT3_T(t,temp);
   MAT3_Inverse(temp,T_Transpose_Inverse);
   MAT3_Inverse(t,T_Inverse);
   Dot_MAT3(T_Inverse,Q,temp);
   Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);
}

  h=H/Dy;
  for(i=0;i<3;i++)
   for(j=0;j<3;j++)
      {  A[i][j]=B[i][j]=D[i][j]=0.0;
      for(k=1;k<=dy;k++)
         {
            A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
            B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
                *(powl((-H/2 + h*k),2) - powl((-H/2 + h*(k-1)),2)));
            D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
                *(powl((-H/2 + h*k),3) - powl((-H/2 + h*(k-1)),3)));
         }
     }

  MAT3_Inverse(A,a);
  MAT3_Inverse(B,b);
  MAT3_Inverse(D,d);

  M[0]= Mx  ; M[1] = 0.0 ; M[2] = 0.0 ;
  N[0]= 0.0 ; N[1] = 0.0 ; N[2] = 0.0 ;

  for(i=0;i<3;i++)
   for(j=0;j<3;j++)
      B_prime[i][j]=D_prime[i][j]=temp[i][j]=0.0;

  //**************D'=(D-BA^(-1)B)^(-1)**************//
  Dot_MAT3(B,a,temp);
  Dot_MAT3(temp,B,temp1);
  for(i=0;i<3;i++)
   for(j=0;j<3;j++)
     temp1[i][j]=D[i][j]-temp1[i][j];
  MAT3_Inverse(temp1,D_prime);

  //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
  Dot_MAT3(a,B,temp);
  MAT3_multip(temp,-1,temp1);
  Dot_MAT3(temp1,D_prime,B_prime);
  Dot_MAT3_vec3(B_prime,M,epsilon_o);
  Dot_MAT3_vec3(D_prime,M,keppa_o);

  //******** Determine stresses in plane 1-2 *******//
```

```
   for(k=1;k<=Dy;k++)
   {
    Dot_MAT3(T_kth[k],Q_bar[k],temp1);
      z[k-1]=-H/2+h*(k-1);
      z[k]=-H/2+h*(k);

      for(i=0;i<3;i++)
      {
       epsilon_xy[0][i]=epsilon_o[i] + keppa_o[i]*z[k-1] ;
       epsilon_xy[1][i]=epsilon_o[i] + keppa_o[i]*z[k]   ;
      }
      Dot_MAT3_vec3(temp1,epsilon_xy[0],sigma12[k][0]);
      Dot_MAT3_vec3(temp1,epsilon_xy[1],sigma12[k][1]);
   }

   //****************** Display ******************//
   printf("s1 s2 s12 top : s1 s2 s12 bottom\n\n");
   for(k=1;k<=dy;k++)
      {printf("% 11.4Le  % 11.4Le  % 11.4Le : % 11.4Le  % 11.4Le  % 11.4Le\n",
             sigma12[k][0][0], sigma12[k][0][1], sigma12[k][0][2],
             sigma12[k][1][0], sigma12[k][1][1], sigma12[k][1][2]);
      }getch();*/
}
/************************End of Stresses Evaluation Function ************************/

/**************************Matrix Q Determination Function*************************/
void MAT_Q(double e1, double e2, double u12, double g12,double q[3][3])
{
   double u21;

   u21=e2/e1*u12;

   q[0][0]=e1/(1-u12*u21);
   q[0][1]=q[1][0]=u12*e2/(1-u12*u21);
   q[1][1]=e2/(1-u12*u21);
   q[2][2]=g12;
   q[0][2]=q[1][2]=q[2][0]=q[2][1]=0.0;

   return;
}
/************************End of Matrix Q Determination Function************************/

/************************Transformation Matrix ,[T], Function************************/
void MAT_Transform(double deg,double t[3][3])
{
   t[0][0]=pow(cos(M_PI/180*deg),2);
   t[0][1]=pow(sin(M_PI/180*deg),2);
   t[0][2]=2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0]=pow(sin(M_PI/180*deg),2);
   t[1][1]=pow(cos(M_PI/180*deg),2);
   t[1][2]=-2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0]=-sin(M_PI/180*deg)*cosl(M_PI/180*deg);
   t[2][1]=sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2]=pow(cos(M_PI/180*deg),2)-pow(sin(M_PI/180*deg),2);

   return;
}
/********************End of Transformation Matrix [T] Function ********************/

/****************************Matrix Replication Function****************************/
void MAT3_copy(int k,double t[3][3],double T[][3][3])
{
   int i,j;

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    T[k][i][j]=t[i][j];
 return;
}
/*************************End of Matrix Replication Function************************/

/*********************************Dot Matrix Function*********************************/
void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3])
```

```
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       w[i][j]=u[i][0]*v[0][j]+u[i][1]*v[1][j]+u[i][2]*v[2][j];
   return;
}
/*****************************End of Dot Matrix Function *************************/

/*******************************Dot Vector Function*******************************/
void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3])
{
   int i;

 for(i=0;i<3;i++)
    w[i]=u[i][0]*v[0]+u[i][1]*v[1]+u[i][2]*v[2];
 return;
}
/*****************************End of Dot Vector Function *************************/

/*******************************Inverse Matrix Function***************************/
void MAT3_Inverse(double u[3][3],double r[3][3])
{
   double DET(double u[3][3]);
   void adj(double u[3][3],double r[3][3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);

   int i,j;
   double temp[3][3];

   adj(u,temp);
   if(DET(u)==0){
    for(i=0;i<3;i++)
      for(j=0;j<3;j++)
           r[i][j]=0.0;
       }else
   MAT3_multip(temp,1/DET(u),r);
   return;
}
/*****************************End of Inverse Matrix Function *********************/

/*******************************Determinant Function******************************/
double DET(double u[3][3])
{
   double temp;

 temp=u[0][0]*u[1][1]*u[2][2]+u[0][1]*u[1][2]*u[2][0]+u[0][2]*u[1][0]*u[2][1]
     -(u[2][0]*u[1][1]*u[0][2]+u[2][1]*u[1][2]*u[0][0]+u[2][2]*u[1][0]*u[0][1]);
   return(temp);
}
/*****************************End of Determinant Function*************************/

/*******************************Adjoint Matrix Function***************************/
void adj(double u[3][3],double r[3][3])
{
   void MAT3_T(double a[3][3],double r[3][3]);
   void Cof(double a[3][3],double r[3][3]);

   double temp[3][3];

   Cof(u,temp);
   MAT3_T(temp,r);
   return;
}
/*****************************End of Determinant Function*************************/

/*******************************Matrix Transpose Function*************************/
void MAT3_T(double a[3][3],double r[3][3])
{
   int i,j;

 for(i=0;i<3;i++)
```

```
    for(j=0;j<3;j++)
       r[i][j]=a[j][i];
   return;
}
/************************End of Matrix Transpose Function**************************/

/*****************************Cofactor Matrix Function*****************************/
void Cof(double a[3][3],double r[3][3])
{
   int i,j;
   double M[3][3];

   M[0][0]=a[1][1]*a[2][2]-a[2][1]*a[1][2];
   M[0][1]=a[1][0]*a[2][2]-a[2][0]*a[1][2];
   M[0][2]=a[1][0]*a[2][1]-a[2][0]*a[1][1];
   M[1][0]=a[0][1]*a[2][2]-a[2][1]*a[0][2];
   M[1][1]=a[0][0]*a[2][2]-a[2][0]*a[0][2];
   M[1][2]=a[0][0]*a[2][1]-a[2][0]*a[0][1];
   M[2][0]=a[0][1]*a[1][2]-a[1][1]*a[0][2];
   M[2][1]=a[0][0]*a[1][2]-a[1][0]*a[0][2];
   M[2][2]=a[0][0]*a[1][1]-a[1][0]*a[0][1];

 for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=powl(-1,i+j)*M[i][j];
 return;
}
/************************End of Cofactor Matrix Function**************************/

/************************Scalar-Multiply-Matrix Function**************************/
void MAT3_multip(double u[3][3],double k,double r[3][3])
{
   int i,j;

   for(i=0;i<3;i++)
    for(j=0;j<3;j++)
       r[i][j]=k*u[i][j];
   return;
}
/**********************End of Scalar-Multiply-Matrix Function**********************/

/***************************Combination Vector Function*****************************/
void vec3_plus(double u[3],double v[3],int k,double w[][3])
{
   int i;

 for(i=0;i<3;i++)
    w[k][i]=u[i]+v[i];
   return;
}
/***********************End of Combination Vector Function ************************/

/*******************************Minimum Value Function*******************************/
double find_min(double a[2],double a_min)
{
   int i;

   a_min=1.0e500;
 for(i=0;i<3;i++)
   {
      if(a[i]>0)
      if(a_min>a[i])a_min=a[i];
   }
   return a_min;
}
/*************************End of Minimum Value Function *************************/
```

5. โปรแกรมคำนวณโมเมนต์สูงสุดสำหรับโมดูลัสยืดหยุ่น 4 ค่า

```
//*************************************************************************//
//                                                                         //
//                          Evaluate Maximum Moment                        //
//                                                                         //
//*************************************************************************//
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>

// Beam feature
#define dy 20                //   the number of layers
#define MAXLAYERS 30         //   the maximum number of layers allowance
#define H 0.02072            //   thickness of OSL in meter

// OSL properties
#define C1 49.53218539E6      //   compression strength in the fiber direction
#define T1 65.83701053E6      //   tensile strength in the fiber direction

#define C2 12.08579058E6      //   compression strength in the perpendicular of fiber
direction
#define T2 1.637675473E6      //   tensile strength in the perpendicular of fiber
direction

#define S12 19.45197537E6     //   shear strength in plane LT

#define EC1 4869.274965E6     //   MOE in the longitudinal direction
#define ET1 2005.741868E6     //   MOE in the longitudinal direction

#define EC2 290.3484232E6     //   MOE in the transverse direction @compression region
#define ET2 288.1998174E6     //   MOE in the transverse direction @tension region

//#define v12 0.35            //   possion's ratio in plane 1-2(LT)
//#define G12 0.08E9          //   modulus of rigidity in plane 1-2(LT)

// File
#define f_name "FindMx2.txt"

double NA_in, NA_out;

main()
{
   FILE *optimize;

   void evaluate_stress(int Dy,int deg[MAXLAYERS],double v12,double G12,double
sigma12[][2][3],double sigmaxy[][2][3],double e12[][2][3]);
   double find_min(double a[dy],double a_min);

   int i,j,k,l,m;
   double n;
   int DEG[MAXLAYERS], DEG_MAX_Mx[MAXLAYERS], DEG_optimize_Mx[MAXLAYERS],
DEG_optimize_min_Mx[MAXLAYERS], DEG_TMP[MAXLAYERS];
   double mx[2],min_mx[MAXLAYERS],MIN_Mx,MAX_Mx,optimize_Mx,optimize_min_Mx,
          SIGMA12[MAXLAYERS][2][3],SIGMAxy[MAXLAYERS][2][3],
          e12[MAXLAYERS][2][3],
          x0, y0, s0,
          x1, y1, s1,
          X, Y, C[MAXLAYERS], T[MAXLAYERS],
          F1, F2, F11, F22, F12, F66,
          THC_Mx, MSC_Mx, TWC1_Mx, TWC2_Mx, HF_Mx,
          v12,G12;


   clrscr();
   optimize = fopen(f_name,"w+");
   optimize_Mx = 0.0;
```

```
for(i = 1; i <= dy; i++)
 {
   printf("Enter Angle Layer %d = ",i);
   scanf("%d",&DEG[i]);
   fprintf(optimize,"%d ",DEG[i]);
   DEG[dy-(i-1)] = -DEG[i];
   min_mx[i] = 0.0;
 }

printf("\nv12;G12;THC;MSC;TWC1;TWC2\n");
fprintf(optimize,"\nv12;G12;THC;MSC;TWC1;TWC2\n");
for( k = 0; k <= 20; k++)
  {
    v12 = k/20.0*0.6;
    for( l = 0; l <= 40; l++)
     {
       G12 = l/40.0*1.0e9;
       m = 0;

       do{
           m = m + 1;
           NA_in = m*H/dy;
           evaluate_stress(dy,DEG,v12,G12,SIGMA12,SIGMAxy,e12);
}while((NA_out - NA_in) > 0.0005);

/*******Find moment following that Tsai-Hill Criterion*******/
for(i = 1; i <= dy; i++)
 {
   x0 = SIGMA12[i][0][0];
   y0 = SIGMA12[i][0][1];
   s0 = SIGMA12[i][0][2];
   if(x0 > 0) X = T1; else X = C1;
   if(y0 > 0) Y = T2; else Y = C2;
   mx[0] = sqrt(1.0/( (x0*x0)/(X*X) - (x0*y0)/(X*X) + (y0*y0)/(Y*Y) +
                      (s0*s0)/(S12*S12)
                    )
         );

   x1 = SIGMA12[i][1][0];
   y1 = SIGMA12[i][1][1];
   s1 = SIGMA12[i][1][2];
   if(x1 > 0) X = T1; else X = C1;
   if(y1 > 0) Y = T2; else Y = C2;
   mx[1] = sqrt(1.0/( (x1*x1)/(X*X) - (x1*y1)/(X*X) + (y1*y1)/(Y*Y) +
                      (s1*s1)/(S12*S12)
                    )
        );

   min_mx[i] = mx[0];
   if(min_mx[i] > mx[1]) min_mx[i] = mx[1];
 }
MIN_Mx = min_mx[1];
for(j = 1; j <= dy; j++)
 if(MIN_Mx > min_mx[j]) MIN_Mx = min_mx[j];
THC_Mx = MIN_Mx;

/*******Find moment following that Maximum Stress Criterion*******/
for(i = 1; i <= dy; i++)
 {
   x0 = SIGMA12[i][0][0];
   y0 = SIGMA12[i][0][1];
   s0 = SIGMA12[i][0][2];

   x1 = SIGMA12[i][1][0];
   y1 = SIGMA12[i][1][1];
   s1 = SIGMA12[i][1][2];

   if(x0 == 0) mx[0] = 1e300;
   else mx[0] = (-C1/x0);
   if(x1 == 0) mx[1] = 1e300;
   else mx[1] = (-C1/x1);

   if(x0 == 0) mx[2] = 1e300;
```

```
      else mx[2] = (T1/x0);
      if(x1 == 0) mx[3] = 1e300;
      else mx[3] = (T1/x1);

      if(y0 == 0) mx[4] = 1e300;
      else mx[4] = (-C2/y0);
      if(y1 == 0) mx[5] = 1e300;
      else mx[5] = (-C2/y1);

      if(y0 == 0) mx[6] = 1e300;
      else mx[6] = (T2/y0);
      if(y1 == 0) mx[7] = 1e300;
      else mx[7] = (T2/y1);

      if(s0 == 0) mx[8] = 1e300;
      else mx[8] = (S12/s0);
      if(s1 == 0) mx[9] = 1e300;
      else mx[9] = (S12/fabs(s1));

      min_mx[i] = 1.0e500;
      for(j = 0; j < 10; j++)
      {
         if(mx[j] > 0)
          if(min_mx[i] > mx[j])min_mx[i] = mx[j];
         }
      }

   MIN_Mx = min_mx[1];
   for(j = 1; j <= dy; j++)
      if(MIN_Mx > min_mx[j]) MIN_Mx = min_mx[j];
   MSC_Mx = MIN_Mx;

   printf("%lg;",THC_Mx);
   fprintf(optimize,"%lg;",THC_Mx);
   printf("%lg;",MSC_Mx);
   fprintf(optimize,"%lg;",MSC_Mx);
   }
   }
   fclose(optimize);
   printf("Finish!!!!!!\a");
}
/*********************************End of Main Program*******************************/

// Stresses Evaluation Function
void evaluate_stress(int Dy, int deg[MAXLAYERS],double v12,double G12,
        double sigma12[][2][3], double sigmaxy[][2][3], double e12[][2][3])
{
   void MAT_Q(double e1,double e2,double u12, double g12,double q[3][3]);
   void MAT_Transform(double deg,double t[3][3]);

   void MAT3_copy(int k,double t[3][3],double T[][3][3]);
   void MAT3_Inverse(double u[3][3],double r[3][3]);
   void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3]);
   void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);
   void MAT3_T(double a[3][3],double r[3][3]);
   void vec3_plus(double u[3],double v[3],int k,double w[][3]);

   int i,j,k;
   double Q[MAXLAYERS][3][3], S[MAXLAYERS][3][3],
          t[3][3],T_kth[MAXLAYERS][3][3],Q_bar[MAXLAYERS][3][3],
          T_Transpose_Inverse[3][3],T_Inverse[3][3],
          A[3][3],B[3][3],D[3][3],B_prime[3][3],D_prime[3][3],
          a[3][3],b[3][3],d[3][3],
          M[3],N[3],
          epsilon_o[3],keppa_o[3],
          epsilon_xy[MAXLAYERS][2][3],
          h,Mx,z[dy+1],
          temp[3][3],temp1[3][3],
          e1,e2;

   Mx = 1.0;
   //**********  Find Matrix Q_bar at Layer kth  **********//
```

```
 for(k = 1; k <= Dy; k++)
{
    if(NA_in >= k*H/Dy){
       e1 = EC1;
       e2 = EC2;
       }
    else{
       e1 = ET1;
       e2 = ET2;
       }
    MAT_Q(e1,e2,v12,G12,Q[k]);
    MAT_Transform(deg[k],t);MAT3_copy(k,t,T_kth);
    MAT3_T(t,temp);
    MAT3_Inverse(temp,T_Transpose_Inverse);
    MAT3_Inverse(t,T_Inverse);
    Dot_MAT3(T_Inverse,Q[k],temp);
    Dot_MAT3(temp,T_Transpose_Inverse,Q_bar[k]);

 }

 //***************** Find Matrix A B D *****************//
 h = H/Dy;
 for(i = 0; i < 3; i++)
  for(j = 0; j < 3; j++)
    {  A[i][j] = B[i][j] = D[i][j] = 0.0;
    for(k = 1; k <= dy; k++)
       {
          A[i][j] = A[i][j] + (Q_bar[k][i][j]*((-H/2 + h*k)-(-H/2 + h*(k-1))));
          B[i][j] = B[i][j] + (Q_bar[k][i][j]/2
              *(pow((-H/2 + h*k),2) - pow((-H/2 + h*(k-1)),2)));
          D[i][j] = D[i][j] + (Q_bar[k][i][j]/3
              *(pow((-H/2 + h*k),3) - pow((-H/2 + h*(k-1)),3)));
       }
}

 MAT3_Inverse(A,a);
 MAT3_Inverse(B,b);
 MAT3_Inverse(D,d);

 M[0]= Mx  ; M[1] = 0.0 ; M[2] = 0.0 ;
 N[0]= 0.0 ; N[1] = 0.0 ; N[2] = 0.0 ;

 for(i = 0; i < 3; i++)
  for(j = 0; j < 3; j++)
    B_prime[i][j] = D_prime[i][j] = temp[i][j]=0.0;

 //**************D'=(D-BA^(-1)B)^(-1)**************//
 Dot_MAT3(B,a,temp);
 Dot_MAT3(temp,B,temp1);
 for(i = 0; i < 3; i++)
  for(j = 0; j < 3; j++)
   temp1[i][j] = D[i][j] - temp1[i][j];
 MAT3_Inverse(temp1,D_prime);

 //**********B'=-A^(-1)B(D-BA^(-1)B)^(-1)**********//
 Dot_MAT3(a,B,temp);
 MAT3_multip(temp,-1,temp1);
 Dot_MAT3(temp1,D_prime,B_prime);

 Dot_MAT3_vec3(B_prime,M,epsilon_o);
 Dot_MAT3_vec3(D_prime,M,keppa_o);

 //*****************Determine N.A.*****************//
 //             (refer to top surface)             //
 NA_out = H/2 - epsilon_o[0]/keppa_o[0];

 //******** Determine stresses in plane 1-2 *******//
 for(k = 1; k <= Dy; k++)
 {
  Dot_MAT3(T_kth[k],Q_bar[k],temp1);
    z[k-1] = -H/2 + h*(k-1);
    z[k]   = -H/2 + h*(k);
```

```
      for(i = 0; i < 3; i++)
      {
        epsilon_xy[k][0][i] = epsilon_o[i] + keppa_o[i]*z[k-1] ;
        epsilon_xy[k][1][i] = epsilon_o[i] + keppa_o[i]*z[k]   ;
      }

      Dot_MAT3_vec3(temp1,epsilon_xy[k][0],sigma12[k][0]);
      Dot_MAT3_vec3(temp1,epsilon_xy[k][1],sigma12[k][1]);
      Dot_MAT3_vec3(Q_bar[k],epsilon_xy[k][0],sigmaxy[k][0]);
      Dot_MAT3_vec3(Q_bar[k],epsilon_xy[k][1],sigmaxy[k][1]);

      MAT3_Inverse(Q[k],S[k]);
      Dot_MAT3_vec3(S[k],sigma12[k][0],e12[k][0]);
      Dot_MAT3_vec3(S[k],sigma12[k][1],e12[k][1]);
   }

   //****************** Display ******************//
   printf("s1 s2 s12 top : s1 s2 s12 bottom\n\n");
   for(k=1;k<=dy;k++)
      {printf("% 11.4Le  % 11.4Le  % 11.4Le : % 11.4Le  % 11.4Le  % 11.4Le\n",
             sigma12[k][0][0], sigma12[k][0][1], sigma12[k][0][2],
             sigma12[k][1][0], sigma12[k][1][1], sigma12[k][1][2]);
      }getch();*/
}

void MAT_Q(double e1, double e2, double u12,
      double g12,double q[3][3])
{
   double u21;

   u21 = (e2/e1)*u12;

   q[0][0] = e1/(1 - u12*u21);
   q[0][1] = q[1][0] = u12*e2/(1 - u12*u21);
   q[1][1] = e2/(1 - u12*u21);
   q[2][2] = g12;
   q[0][2] = q[1][2] = q[2][0] = q[2][1] = 0.0;

   return;
}

void MAT_Transform(double deg,double t[3][3])
{
   t[0][0] = pow(cos(M_PI/180*deg),2);
   t[0][1] = pow(sin(M_PI/180*deg),2);
   t[0][2] = 2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[1][0] = pow(sin(M_PI/180*deg),2);
   t[1][1] = pow(cos(M_PI/180*deg),2);
   t[1][2] = -2*sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][0] = -sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][1] = sin(M_PI/180*deg)*cos(M_PI/180*deg);
   t[2][2] = pow(cos(M_PI/180*deg),2) - pow(sin(M_PI/180*deg),2);

   return;
}

void MAT3_copy(int k,double t[3][3],double T[][3][3])
{
   int i,j;

 for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
    T[k][i][j] = t[i][j];
 return;
}

void Dot_MAT3(double u[3][3],double v[3][3],double w[3][3])
{
   int i,j;

   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
      w[i][j] = u[i][0]*v[0][j] + u[i][1]*v[1][j] + u[i][2]*v[2][j];
```

```
   return;
}

void Dot_MAT3_vec3(double u[3][3],double v[3],double w[3])
{
   int i;

 for(i = 0; i < 3; i++)
    w[i] = u[i][0]*v[0] + u[i][1]*v[1] + u[i][2]*v[2];
 return;
}

void MAT3_Inverse(double u[3][3],double r[3][3])
{
   double DET(double u[3][3]);
   void adj(double u[3][3],double r[3][3]);
   void MAT3_multip(double u[3][3],double k,double r[3][3]);

   int i,j;
   double temp[3][3];

   adj(u,temp);
   if(DET(u) == 0){
    for(i = 0; i < 3; i++)
     for(j = 0; j < 3; j++)
          r[i][j] = 0.0;
      }else
   MAT3_multip(temp,1/DET(u),r);
   return;
}

double DET(double u[3][3])
{
   double temp;

 temp =  u[0][0]*u[1][1]*u[2][2] + u[0][1]*u[1][2]*u[2][0] + u[0][2]*u[1][0]*u[2][1]
      -(u[2][0]*u[1][1]*u[0][2] + u[2][1]*u[1][2]*u[0][0] + u[2][2]*u[1][0]*u[0][1]);
 return(temp);
}

void adj(double u[3][3],double r[3][3])
{
   void MAT3_T(double a[3][3],double r[3][3]);
   void Cof(double a[3][3],double r[3][3]);

   double temp[3][3];

   Cof(u,temp);
   MAT3_T(temp,r);
   return;
}

void MAT3_T(double a[3][3],double r[3][3])
{
   int i,j;

 for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
      r[i][j] = a[j][i];
 return;
}

void Cof(double a[3][3],double r[3][3])
{
   int i,j;
   double M[3][3];

   M[0][0] = a[1][1]*a[2][2] - a[2][1]*a[1][2];
   M[0][1] = a[1][0]*a[2][2] - a[2][0]*a[1][2];
   M[0][2] = a[1][0]*a[2][1] - a[2][0]*a[1][1];
   M[1][0] = a[0][1]*a[2][2] - a[2][1]*a[0][2];
   M[1][1] = a[0][0]*a[2][2] - a[2][0]*a[0][2];
   M[1][2] = a[0][0]*a[2][1] - a[2][0]*a[0][1];
```

```
   M[2][0] = a[0][1]*a[1][2] - a[1][1]*a[0][2];
   M[2][1] = a[0][0]*a[1][2] - a[1][0]*a[0][2];
   M[2][2] = a[0][0]*a[1][1] - a[1][0]*a[0][1];

 for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       r[i][j] = pow(-1,i+j)*M[i][j];
 return;
}

void MAT3_multip(double u[3][3],double k,double r[3][3])
{
   int i,j;

   for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
       r[i][j] = k*u[i][j];
   return;
}

void vec3_plus(double u[3],double v[3],int k,double w[][3])
{
   int i;

 for(i = 0; i < 3; i++)
    w[k][i] = u[i] + v[i];
 return;
}
/*****************************************************************************/

// Finding minimum value
double find_min(double a[dy],double a_min)
{
   int i;

   a_min=1.0e300;
 for(i = 0;i < dy; i++)
   {
      if(a[i] > 0)
      if(a_min > a[i]) a_min = a[i];
   }
 return a_min;
}
/*****************************************************************************/
```

ภาคผนวก ข

โมดูลัสเฉือนและอัตราส่วนพัวส์ซองของไม้จริงบางชนิด

โมดูลัสเฉือนเฉลี่ยจากไม้จริงทั้งจากไม้ประเภทไม้สนและไม้ใบกว้าง มีค่าเฉลี่ยเท่ากับ 0.079 เท่าของโมดูลัสยืดหยุ่นตามเสี้ยน ($G_{12} = 0.079E_1$) และมีค่าอยู่ในช่วง 0.032 – 0.210 เท่าของโมดูลัสยืดหยุ่นตามเสี้ยน อัตราส่วนพัวส์ซองเฉลี่ยเท่ากับ 0.389 และมีช่วงอยู่ระหว่าง 0.276 – 0.641 ดังแสดงไว้ใน Table B.1

Table B.1 Ratios of elastic for some species of softwoods and Hardwoods at 12% moisture content

Species	Specific Gravity	E_T/E_L	E_R/E_L	GL_R/E_L	G_{LT}/E_L	ν_{LR}	ν_{LT}
				Softwoods			
Baldcypress	0.46	0.039	0.084	0.063	0.054	0.338	0.326
Cedar, northern white	0.31	0.081	0.183	0.210	0.187	0.337	0.340
Cedar, western red	0.32	0.055	0.081	0.087	0.086	0.378	0.296
Douglas-fir	0.46-0.50	0.050	0.068	0.064	0.078	0.292	0.449
Fir, subalpine	0.32	0.039	0.102	0.070	0.058	0.341	0.332
Hemlock, western	0.45	0.031	0.058	0.038	0.032	0.485	0.423
Larch, western	0.52	0.065	0.079	0.063	0.069	0.355	0.276
Pine							
Loblolly	0.51	0.078	0.113	0.082	0.081	0.328	0.292
Lodgepole	0.41	0.068	0.102	0.049	0.046	0.316	0.347
Longleaf	0.59	0.055	0.102	0.071	0.060	0.332	0.365
Pond	0.56	0.041	0.071	0.050	0.045	0.280	0.364
Ponderosa	0.40	0.083	0.122	0.138	0.115	0.337	0.400
Red	0.46	0.044	0.088	0.096	0.081	0.347	0.315
Slash	0.59	0.045	0.074	0.055	0.053	0.392	0.444
Sugar	0.36	0.087	0.131	0.124	0.113	0.356	0.349
Western white	0.38	0.038	0.078	0.052	0.048	0.329	0.344
Redwood	0.35-0.40	0.089	0.087	0.066	0.077	0.360	0.346
Spruce, Sitka	0.36	0.043	0.078	0.064	0.061	0.372	0.467
Spruce, Engelmann	0.35	0.059	0.128	0.124	0.120	0.422	0.462
Average	0.432	0.057	0.096	0.082	0.077	0.352	0.365
					0.080		0.359
range	0.31-0.59	0.031-0.089	0.058-0.183	0.038-0.210	0.032-0.187	0.280-0.485	0.276-0.467
				Hardwoods			
Ash, white	0.60	0.080	0.125	0.109	0.077	0.371	0.440
Aspen, quaking	0.38	—	—	—	—	0.489	0.374
Balsa	0.16*	0.015	0.046	0.054	0.037	0.229	0.488
Basswood	0.37	0.027	0.066	0.056	0.046	0.364	0.406
Birch, yellow	0.62	0.050	0.078	0.074	0.068	0.426	0.451
Cherry, black	0.50	0.086	0.197	0.147	0.097	0.392	0.428
Cottonwood, eastern	0.40	0.047	0.083	0.076	0.052	0.344	0.420
Mahogany, African	0.42*	0.050	0.111	0.088	0.059	0.297	0.641
Mahogany, Honduras	—	0.064	0.107	0.066	0.086	0.314	0.533
Maple, sugar	0.63	0.065	0.132	0.111	0.063	0.424	0.476
Maple, red	0.54	0.067	0.140	0.133	0.074	0.434	0.509
Oak, red	0.59-0.69	0.082	0.154	0.089	0.081	0.350	0.448
Oak, white	0.63-0.88	0.072	0.163	0.086	—	0.369	0.428
Sweet gum	0.52	0.050	0.115	0.089	0.061	0.325	0.403
Walnut, black	0.55	0.056	0.106	0.085	0.062	0.495	0.632
Yellow-poplar	0.42	0.043	0.092	0.075	0.069	0.318	0.392
Average	0.503	0.057	0.114	0.089	0.067	0.371	0.467
					0.079		0.419
range	0.16-0.88	0.015-0.086	0.046-0.197	0.054-0.147	0.037-0.097	0.229-0.495	0.374-0.641

Source : Forest Products Lab. 1999. Wood Handbook-Wood as as enginering material. Madison, WI : U.S.

ภาคผนวก ค

การหาแบบการเรียงแถบไม้ที่ให้ค่าความแข็งแรงสูงสุด

ตัวอย่างผลการสุ่มมุมเริ่มต้นที่แตกต่างกันโดยคอมพิวเตอร์และจัดลำดับ 50 แบบการเรียงที่ดีที่สุดของแต่ละเกณฑ์การวิบัติ

1. โอเอสแอลแบบ I

1.1 เกณฑ์ความเค้นสูงสุด

ผลการคำนวณด้วยคอมพิวเตอร์ตามเกณฑ์ความเค้นสูงสุดจากการสุ่มมุมเริ่มต้น 1000 ค่า แสดงแบบการเรียงมุมที่ดีที่สุด 50 ค่าโดยเรียงลำดับจากมากไปหาน้อย

```
No.: 0
Optimal Orientation : [ 31, 28, 25, 21, 16, 10, 1, 172, 169, 153]A
Moment (Nm/m) = 3.2441e+03

No.: 1
Optimal Orientation : [ 31, 28, 25, 21, 16, 10, 1, 173, 169, 153]A
Moment (Nm/m) = 3.2428e+03

No.: 2
Optimal Orientation : [ 31, 28, 25, 21, 16, 10, 1, 174, 167, 153]A
Moment (Nm/m) = 3.2409e+03

No.: 3
Optimal Orientation : [ 31, 28, 25, 21, 16, 10, 180, 167, 162, 154]A
Moment (Nm/m) = 3.2315e+03

No.: 4
Optimal Orientation : [ 149, 152, 155, 159, 164, 170, 180, 11, 20, 26]A
Moment (Nm/m) = 3.2309e+03

No.: 5
Optimal Orientation : [ 149, 152, 155, 159, 164, 170, 180, 11, 20, 26]A
Moment (Nm/m) = 3.2309e+03

No.: 6
Optimal Orientation : [ 150, 153, 156, 160, 165, 172, 1, 9, 21, 18]A
Moment (Nm/m) = 3.2304e+03

No.: 7
Optimal Orientation : [ 150, 153, 156, 160, 165, 172, 1, 8, 22, 18]A
Moment (Nm/m) = 3.2285e+03

No.: 8
Optimal Orientation : [ 30, 27, 24, 20, 15, 8, 178, 160, 157, 177]A
Moment (Nm/m) = 3.2282e+03

No.: 9
Optimal Orientation : [ 30, 27, 24, 20, 15, 8, 178, 159, 158, 177]A
Moment (Nm/m) = 3.2275e+03

No.: 10
Optimal Orientation : [ 30, 27, 24, 20, 15, 8, 178, 174, 164, 148]A
Moment (Nm/m) = 3.2166e+03

No.: 11
Optimal Orientation : [ 150, 152, 155, 159, 164, 171, 0, 15, 15, 15]A
Moment (Nm/m) = 3.2162e+03

No.: 12
Optimal Orientation : [ 30, 28, 25, 21, 16, 9, 179, 164, 165, 157]A
```

```
Moment (Nm/m) = 3.2090e+03

No.: 13
Optimal Orientation : [ 30, 28, 25, 21, 16, 9, 179, 165, 165, 156]A
Moment (Nm/m) = 3.2076e+03

No.: 14
Optimal Orientation : [ 30, 28, 25, 21, 16, 9, 179, 166, 165, 155]A
Moment (Nm/m) = 3.2058e+03

No.: 15
Optimal Orientation : [ 30, 28, 20, 20, 15, 8, 178, 174, 162, 148]A
Moment (Nm/m) = 3.2031e+03

No.: 16
Optimal Orientation : [ 151, 152, 155, 159, 164, 171, 179, 15, 16, 16]A
Moment (Nm/m) = 3.2029e+03

No.: 17
Optimal Orientation : [ 31, 28, 25, 21, 16, 9, 180, 164, 162, 155]A
Moment (Nm/m) = 3.2029e+03

No.: 18
Optimal Orientation : [ 135, 35, 33, 30, 27, 22, 144, 8, 168, 178]A
Moment (Nm/m) = 3.2026e+03

No.: 19
Optimal Orientation : [ 152, 155, 158, 162, 168, 175, 4, 10, 6, 6]A
Moment (Nm/m) = 3.2011e+03

No.: 20
Optimal Orientation : [ 30, 28, 25, 21, 16, 10, 1, 145, 171, 177]A
Moment (Nm/m) = 3.2005e+03

No.: 21
Optimal Orientation : [ 148, 150, 153, 156, 161, 166, 42, 8, 5, 6]A
Moment (Nm/m) = 3.1998e+03

No.: 22
Optimal Orientation : [ 30, 28, 25, 21, 16, 10, 1, 144, 171, 177]A
Moment (Nm/m) = 3.1994e+03

No.: 23
Optimal Orientation : [ 28, 25, 22, 18, 12, 5, 176, 174, 170, 172]A
Moment (Nm/m) = 3.1988e+03

No.: 24
Optimal Orientation : [ 31, 28, 25, 21, 16, 9, 1, 164, 162, 154]A
Moment (Nm/m) = 3.1985e+03

No.: 25
Optimal Orientation : [ 28, 25, 22, 18, 12, 5, 176, 175, 169, 172]A
Moment (Nm/m) = 3.1977e+03

No.: 26
Optimal Orientation : [ 28, 25, 22, 18, 12, 5, 176, 174, 176, 166]A
Moment (Nm/m) = 3.1930e+03

No.: 27
Optimal Orientation : [ 28, 25, 22, 18, 12, 5, 176, 175, 176, 165]A
Moment (Nm/m) = 3.1907e+03

No.: 28
Optimal Orientation : [ 28, 26, 22, 18, 13, 5, 174, 163, 171, 177]A
Moment (Nm/m) = 3.1876e+03

No.: 29
Optimal Orientation : [ 151, 154, 157, 161, 166, 174, 4, 5, 14, 29]A
Moment (Nm/m) = 3.1868e+03

No.: 30
Optimal Orientation : [ 151, 154, 157, 161, 166, 174, 5, 12, 4, 31]A
Moment (Nm/m) = 3.1866e+03
```

No.: 31
Optimal Orientation : [151, 154, 157, 161, 166, 174, 4, 12, 4, 31]A
Moment (Nm/m) = 3.1862e+03

No.: 32
Optimal Orientation : [152, 154, 158, 162, 167, 175, 5, 11, 9, 11]A
Moment (Nm/m) = 3.1862e+03

No.: 33
Optimal Orientation : [29, 27, 23, 19, 14, 6, 174, 170, 159, 155]A
Moment (Nm/m) = 3.1861e+03

No.: 34
Optimal Orientation : [152, 154, 158, 162, 167, 174, 4, 9, 7, 6]A
Moment (Nm/m) = 3.1855e+03

No.: 35
Optimal Orientation : [152, 154, 158, 162, 167, 175, 5, 11, 8, 12]A
Moment (Nm/m) = 3.1852e+03

No.: 36
Optimal Orientation : [29, 27, 23, 19, 14, 6, 174, 171, 158, 155]A
Moment (Nm/m) = 3.1845e+03

No.: 37
Optimal Orientation : [152, 154, 158, 162, 167, 175, 5, 10, 8, 13]A
Moment (Nm/m) = 3.1837e+03

No.: 38
Optimal Orientation : [28, 26, 22, 18, 13, 5, 177, 161, 171, 175]A
Moment (Nm/m) = 3.1823e+03

No.: 39
Optimal Orientation : [152, 155, 158, 162, 168, 176, 8, 11, 15, 10]A
Moment (Nm/m) = 3.1821e+03

No.: 40
Optimal Orientation : [152, 154, 158, 162, 167, 175, 5, 9, 8, 14]A
Moment (Nm/m) = 3.1821e+03

No.: 41
Optimal Orientation : [152, 154, 158, 162, 167, 175, 6, 9, 12, 14]A
Moment (Nm/m) = 3.1816e+03

No.: 42
Optimal Orientation : [152, 154, 158, 162, 167, 175, 5, 8, 8, 15]A
Moment (Nm/m) = 3.1803e+03

No.: 43
Optimal Orientation : [29, 26, 23, 18, 13, 6, 177, 175, 166, 178]A
Moment (Nm/m) = 3.1776e+03

No.: 44
Optimal Orientation : [152, 155, 158, 162, 168, 176, 7, 10, 10, 19]A
Moment (Nm/m) = 3.1776e+03

No.: 45
Optimal Orientation : [152, 155, 158, 163, 168, 175, 4, 6, 5, 2]A
Moment (Nm/m) = 3.1769e+03

No.: 46
Optimal Orientation : [29, 26, 23, 18, 13, 6, 178, 174, 166, 178]A
Moment (Nm/m) = 3.1765e+03

No.: 47
Optimal Orientation : [28, 25, 22, 18, 12, 4, 174, 159, 169, 159]A
Moment (Nm/m) = 3.1745e+03

No.: 48
Optimal Orientation : [28, 25, 22, 18, 12, 4, 174, 159, 169, 175]A
Moment (Nm/m) = 3.1744e+03

```
No.: 49
Optimal Orientation : [ 149, 151, 154, 157, 162, 169, 32, 20, 10, 18]A
Moment (Nm/m) = 3.1741e+03

No.: 50
Optimal Orientation : [ 151, 154, 157, 21, 167, 174, 6, 8, 9, 35]A
Moment (Nm/m) = 3.1738e+03
```

1.2 เกณฑ์ของไซ-ฮิลล์

ผลการคำนวณด้วยคอมพิวเตอร์ตามเกณฑ์ของไซ-ฮิลล์จากการสุ่มมุมเริ่มต้น 1000 ค่า แสดงแบบการเรียงมุมที่ดีที่สุด 50 ค่าโดยเรียงลำดับจากมากไปหาน้อย

```
Univariate Search
No.: 1
Optimal Orientation : [ 29, 25, 21, 16, 9, 0, 174, 172, 174, 169]A
                      [ 29, 25, 21, 16, 9, 0, -6, -8, -6, -11]A
Moment  (Nm/m) =  2.8775e+03

No.: 2
Optimal Orientation : [ 153, 157, 161, 167, 174, 5, 6, 6, 6, 5]A
                      [ -27, -23, -19, -13, -6, 5, 6, 6, 6, 5]A
Moment  (Nm/m) =  2.8769e+03

No.: 3
Optimal Orientation : [ 153, 157, 161, 167, 174, 5, 5, 6, 7, 5]A
                      [ -27, -23, -19, -13, -6, 5, 5, 6, 7, 5]A
Moment  (Nm/m) =  2.8769e+03

No.: 4
Optimal Orientation : [ 153, 157, 161, 167, 174, 4, 6, 7, 7, 4]A
                      [  -27, -23, -19, -13, -6, 4, 6, 7, 7, 4]A
Moment  (Nm/m) =  2.8768e+03

No.: 5
Optimal Orientation : [ 153, 157, 161, 167, 174, 5, 6, 6, 5, 5]A
                      [ -27, -23, -19, -13, -6, 5, 5, 6, 5, 5]A
Moment  (Nm/m) =  2.8766e+03

No.: 6
Optimal Orientation : [ 27, 23, 19, 13, 6, 175, 175, 175, 172, 175]A
                      [ 27, 23, 19, 13, 6, -5, -5, -5, -8, -5]A
Moment  (Nm/m) =  2.8766e+03

No.: 7
Optimal Orientation : [ 153, 157, 161, 167, 174, 4, 5, 6, 7, 6]A
                      [ -27, -23, -19, -13, -6, 4, 5, 6, 7, 6]A
Moment  (Nm/m) =  2.8764e+03

No.: 8
Optimal Orientation : [ 153, 157, 161, 167, 174, 4, 7, 7, 6, 3]A
                      [ -27, -23, -19, -13, -6, 4, 7, 7, 6, 3]A
Moment  (Nm/m) =  2.8764e+03

No.: 9
Optimal Orientation : [ 149, 152, 156, 161, 167, 175, 6, 11, 11, 6]A
                      [ -31, -28, -23, -19, -13, -5, 6, 11, 11, 6]A
Moment  (Nm/m) =  2.8762e+03

No.: 10
Optimal Orientation : [ 153, 157, 161, 167, 174, 3, 6, 7, 8, 5]A
                      [ -27, -23, -19, -13, -6, 3, 6, 7, 8, 5]A
Moment  (Nm/m) =  2.8761e+03

No.: 11
Optimal Orientation : [ 27, 23, 19, 13, 6, 175, 175, 176, 171, 175]A
                      [ 27, 23, 19, 13, 6, -5, -5, -4, -9, -5]A
Moment  (Nm/m) =  2.8760e+03

No.: 12
```

```
Optimal Orientation : [ 153, 157, 161, 167, 174, 3, 6, 8, 8, 3]A
                      [ -27, -23, -19, -13, -6, 3, 6, 8, 8, 3]A
Moment  (Nm/m) =  2.8759e+03

No.: 13
Optimal Orientation : [ 153, 157, 161, 167, 174, 3, 6, 5, 8, 7]A
                      [ -27, -23, -19, -13, -6, 3, 6, 5, 8, 7]A
Moment  (Nm/m) =  2.8754e+03

No.: 14
Optimal Orientation : [ 153, 157, 161, 167, 174, 5, 5, 5, 4, 7]A
                      [ -27, -23, -19, -13, -6, 5, 5, 5, 4, 7]A
Moment  (Nm/m) =  2.8753e+03

No.: 15
Optimal Orientation : [ 29, 25, 21, 16, 9, 1, 173, 172, 174, 169]A
                      [ 29, 25, 21, 16, 9, 1, -7, -8, -6, -11]A
Moment  (Nm/m) =  2.8747e+03

No.: 16
Optimal Orientation : [ 153, 157, 161, 167, 174, 2, 7, 9, 7, 3]A
                      [ -27, -23, -19, -13, -6, 2, 7, 9, 7, 3]A
Moment  (Nm/m) =  2.8741e+03

No.: 17
Optimal Orientation : [ 149, 152, 156, 161, 167, 175, 7, 7, 17, 12]A
                      [ -31, -28, -24, -19, -13, -5, 7, 7, 17, 12]A
Moment  (Nm/m) =  2.8724e+03

No.: 18
Optimal Orientation : [ 152, 156, 160, 165, 172, 2, 7, 6, 6, 4]A
                      [ -29, -24, -20, -15, -8, 2, 7, 6, 6, 4]A
Moment  (Nm/m) =  2.8723e+03

No.: 19
Optimal Orientation : [ 152, 156, 160, 165, 172, 2, 7, 7, 6, 2]A
                      [ -28, -24, -20, -15, -8, 2, 7, 7, 6, 2]A
Moment  (Nm/m) =  2.8722e+03

No.: 20
Optimal Orientation : [ 152, 156, 160, 165, 172, 1, 5, 6, 5, 2]A
                      [ -28, -24, -20, -15, -8, 1, 5, 6, 5, 2]A
Moment  (Nm/m) =  2.8718e+03

No.: 21
Optimal Orientation : [ 153, 156, 161, 166, 173, 4, 7, 6, 5, 3]A
                      [ -27, -24, -19, -14, -7, 4, 7, 6, 5, 3]A
Moment  (Nm/m) =  2.8717e+03

No.: 22
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 173, 171, 169, 168]A
                      [ 28, 25, 21, 15, 8, -2, -7, -9, -11, -12]A
Moment  (Nm/m) =  2.8716e+03

No.: 23
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 173, 171, 169, 168]A
                      [ 28, 25, 21, 15, 8, -2, -7, -9, -11, -12]A
Moment  (Nm/m) =  2.8716e+03

No.: 24
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 173, 171, 169, 168]A
                      [ 28, 25, 21, 15, 8, -2, -7, -9, -11, -12]A
Moment  (Nm/m) =  2.8716e+03

No.: 25
Optimal Orientation : [ 152, 155, 159, 165, 172, 2, 7, 9, 11, 12]A
                      [ -28, -25, -21, -15, -8, 2, 7, 9, 11, 12]A
Moment  (Nm/m) =  2.8716e+03

No.: 26
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 173, 172, 169, 167]A
                      [ 28, 25, 21, 15, 8, -2, -7, -8, -11, -13]A
Moment  (Nm/m) =  2.8715e+03
```

```
No.: 27
Optimal Orientation : [ 153, 157, 161, 167, 174, 3, 4, 5, 4, 11]A
                      [ -27, -23, -19, -13, -6, 3, 4, 5, 4, 11]A
Moment  (Nm/m) =  2.8714e+03

No.: 28
Optimal Orientation : [ 27, 24, 19, 14, 7, 176, 173, 173, 175, 178]A
                      [ 27, 24, 19, 14, 7, -4, -7, -7, -15, -12]A
Moment  (Nm/m) =  2.8714e+03

No.: 29
Optimal Orientation : [ 152, 156, 160, 165, 172, 1, 7, 6, 5, 2]A
                      [ -28, -24,-20, -15, -8, 1, 7, 6, 5, 2]A
Moment  (Nm/m) =  2.8714e+03

No.: 30
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 172, 171, 170, 168]A
                      [ 28, 25, 21, 15, 8, -2, -8, -9, -10, -12]A
Moment  (Nm/m) =  2.8713e+03

No.: 31
Optimal Orientation : [ 152, 155, 159, 165, 172, 2, 8, 9, 10, 12]A
                      [ -28, -25, -21, -15, -8, 2, 8, 9, 10, 12]A
Moment  (Nm/m) =  2.8713e+03

No.: 32
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 172, 171, 171, 168]A
                      [ 28, 25, 21, 15, 8, -2, -8, -9, -9, -12]A
Moment  (Nm/m) =  2.8712e+03

No.: 33
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 9, 13, 10, 5]A
                      [ -30, -27, -23, -18, -12, -3, 9, 13, 10, 5]A
Moment  (Nm/m) =  2.8710e+03

No.: 34
Optimal Orientation : [ 27, 24, 19, 14, 7, 176, 172, 175, 175, 178]A
                      [ 27, 24, 19, 14, 7, -4, -8, -5, -5, -12]A
Moment  (Nm/m) =  2.8709e+03

No.: 35
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 172, 171, 172, 167]A
                      [ 28, 25, 21, 15, 8, -2, -8, -9, -8, -13]A
Moment  (Nm/m) =  2.8709e+03

No.: 36
Optimal Orientation : [ 152, 155, 159, 165, 172, 2, 8, 9, 8, 13]A
                      [ -28, -25, -21, -15, -8, 2, 8, 9, 8, 13]A
Moment  (Nm/m) =  2.8709e+03

No.: 37
Optimal Orientation : [ 149, 152, 156, 161, 167, 175, 5, 7, 11, 14]A
                      [ 28, 25, 21, 15, 8, -2, -8, -9, -10, -12]A
Moment  (Nm/m) =  2.8709e+03

No.: 38
Optimal Orientation : [ 30, 27, 23, 18, 12, 3, 172, 171, 169, 169]A
                      [ 30, 27, 23, 18, 12, 3, -8, -9, -11, -11]A
Moment  (Nm/m) =  2.8708e+03

No.: 39
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 8, 9, 11, 11]A
                      [ -30, -27, -23, -18, -12, -3, 8, 9, 11, 11]A
Moment  (Nm/m) =  2.8708e+03

No.: 40
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 8, 13, 12, 4]A
                      [ -30, -27, -23, -18, -12, -3, 8, 13, 12, 4]A
Moment  (Nm/m) =  2.8707e+03

No.: 41
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 173, 174, 168, 166]A
```

```
                    [ 28, 25, 21, 15, 8, -2, -8, -9, -10, -12]A
Moment  (Nm/m) =  2.8706e+03

No.: 42
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 9, 9, 9, 12]A
                    [ -30, -27, -23, -18, -12, -3, 9, 9, 9, 12]A
Moment  (Nm/m) =  2.8705e+03

No.: 43
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 171, 170, 170, 85]A
                    [ 28, 25, 21, 15, 8, -2, -9, -10, -10, 85]A
Moment  (Nm/m) =  2.8703e+03

No.: 44
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 8, 9, 10, 12]A
                    [ -30, -27, -23, -18, -12, -3, 8, 9, 10, 12]A
Moment  (Nm/m) =  2.8702e+03

No.: 45
Optimal Orientation : [ 30, 27, 23, 18, 12, 3, 173, 170, 169, 169]A
                    [ 30, 27, 23, 18, 12, 3, -7, -10, -11, -11]A
Moment  (Nm/m) =  2.8702e+03

No.: 46
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 8, 14, 11, 4]A
                    [ -30, -27, -23, -18, -12, -3, 8, 14, 11, 4]A
Moment  (Nm/m) =  2.8701e+03

No.: 47
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 9, 14, 10, 4]A
                    [ -30, -27, -23, -18, -12, -3, 9, 14, 10, 4]A
Moment  (Nm/m) =  2.8700e+03

No.: 48
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 9, 9, 8, 13]A
                    [ -30, -27, -23, -18, -12, -3, 9, 9, 8, 13]A
Moment  (Nm/m) =  2.8699e+03

No.: 49
Optimal Orientation : [ 150, 153, 157, 162, 168, 177, 9, 9, 8, 13]A
                    [ -30, -27, -23, -18, -12, -3, 9, 9, 8, 13]A
Moment  (Nm/m) =  2.8699e+03

No.: 50
Optimal Orientation : [ 28, 25, 21, 15, 8, 178, 172, 173, 173, 164]A
                    [ 28, 25, 21, 15, 8, -2, -8, -7, -7, -16]A
Moment  (Nm/m) =  2.8698e+03
```

2. โอเอสแอลแบบ II

2.1 เกณฑ์ความเค้นสูงสุด

ผลการคำนวณด้วยคอมพิวเตอร์ตามเกณฑ์ความเค้นสูงสุดจากการสุ่มมุมเริ่มต้น 1000 ค่า แสดงแบบการเรียงมุมที่ดีที่สุด 50 ค่าโดยเรียงลำดับจากมากไปหาน้อย

```
Univariate Search
No.: 1
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 2
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 3
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 4
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
```

```
Moment (Nm/m) =  3096.17

No.: 5
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 6
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 7
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 8
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 9
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 10
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 11
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 12
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 13
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 14
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 15
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 16
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 17
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 18
Optimal Orientation : [ 155, 158, 162, 166, 172, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 19
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 20
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 21
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 22
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17
```

```
No.: 23
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 24
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 25
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 26
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 27
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 28
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 29
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 30
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 31
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 32
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 33
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 34
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 35
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 36
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 37
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 38
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 39
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 40
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17
```

```
No.: 41
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 42
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 43
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 44
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 45
Optimal Orientation : [ 25, 22, 18, 14, 8, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3096.17

No.: 46
Optimal Orientation : [ 155, 158, 162, 167, 173, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3086.5

No.: 47
Optimal Orientation : [ 155, 158, 162, 167, 173, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3086.5

No.: 48
Optimal Orientation : [ 155, 158, 162, 167, 173, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3086.5

No.: 49
Optimal Orientation : [ 155, 158, 162, 167, 173, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3086.5

No.: 50
Optimal Orientation : [ 155, 158, 162, 167, 173, 0, 0, 0, 0, 0]A
Moment (Nm/m) =  3086.5
```

2.2 เกณฑ์ของไซ-ฮิลล์

ผลการคำนวณด้วยคอมพิวเตอร์ตามเกณฑ์ของไซ-ฮิลล์จากการสุ่มมุมเริ่มต้น 1000 ค่า แสดงแบบการเรียงมุมที่ดีที่สุด 50 ค่าโดยเรียงลำดับจากมากไปหาน้อย

```
Univariate Search
No.: 1
Optimal Orientation : [ 26, 22, 18, 12, 5, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 2
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 3
Optimal Orientation : [ 26, 22, 18, 12, 5, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 4
Optimal Orientation : [ 26, 22, 18, 12, 5, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 5
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 6
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03
```

```
No.: 7
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 8
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 9
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 10
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 11
Optimal Orientation : [ 26, 22, 18, 12, 5, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 12
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 13
Optimal Orientation : [ 26, 22, 18, 12, 5, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 14
Optimal Orientation : [ 26, 22, 18, 12, 5, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 15
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 16
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 17
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 18
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 19
Optimal Orientation : [ 154, 158, 162, 168, 175, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8426e+03

No.: 20
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 21
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 22
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 23
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 24
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03
```

```
No.: 25
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 26
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 27
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 28
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 29
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 30
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 31
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 32
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 33
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 34
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 35
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 36
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 37
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 38
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 39
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 40
Optimal Orientation : [ 24, 20, 15, 9, 1, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 41
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 42
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 43
```

```
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 44
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 45
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 46
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 47
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 48
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 49
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03

No.: 50
Optimal Orientation : [ 156, 160, 165, 171, 179, 0, 0, 0, 0, 0]A
Moment  (Nm/m) =  2.8403e+03
```

ภาคผนวก ง

การกระจายความเค้น ณ จุดออกแบบ

Table D.1 Stress in each layer at optimum orientation [31°, 28°, 25°, 21°, 16°, 10°, 1°, -8°, -11°, -27°]$_A$ by Maximum Stress Criterion (MSC)

Layer (distance from N.A.)	Maximum Stress Criterion (MPa)					
	σ_1	σ_2	τ_{12}	σ_x	σ_y	σ_{xy}
1 (z = -10 mm)	-34.63	-2.02	11.47	-36.10	-0.54	-9.01
2 (z = -9 mm)	-35.68	-1.59	10.35	-36.75	-0.53	-8.34
3 (z = -8 mm)	-35.44	-1.23	9.25	-36.41	-0.26	-7.16
4 (z = -7 mm)	-35.72	-0.84	8.09	-36.65	0.09	-5.66
5 (z = -6 mm)	-35.89	-0.46	6.88	-36.85	0.49	-3.55
6 (z = -5 mm)	-35.44	-0.11	5.66	-36.31	0.76	-0.72
7 (z = -4 mm)	-35.89	0.28	4.20	-36.02	0.42	3.57
8 (z = -3 mm)	-33.67	0.55	3.00	-32.18	-0.94	7.60
9 (z = -2 mm)	-26.75	0.58	2.95	-24.65	-1.52	7.85
10 (z = -1 mm)	-26.60	0.94	1.51	-19.71	-5.95	12.03

Table D.2 Stress in each layer at optimum orientation [29°, 25°, 21°, 16°, 9°, 0°, -6°, -8°, -6°, -11°]$_A$ by Tsai-Hill Criterion (THC)

Layer (distance from N.A.)	Tsai-Hill Criterion (MPa)					
	σ_1	σ_2	τ_{12}	σ_x	σ_y	σ_{xy}
1 (z = -10 mm)	-29.55	-1.16	8.69	-30.24	-0.47	-7.43
2 (z = -9 mm)	-31.77	-0.79	7.70	-32.13	-0.43	-6.92
3 (z = -8 mm)	-32.58	-0.49	6.70	-32.95	-0.12	-5.76
4 (z = -7 mm)	-33.31	-0.19	5.64	-33.78	0.28	-3.99
5 (z = -6 mm)	-34.40	0.13	4.40	-34.91	0.64	-1.15
6 (z = -5 mm)	-34.65	0.40	2.99	-34.65	0.40	2.99
7 (z = -4 mm)	-31.09	0.49	2.23	-30.28	-0.32	5.46
8 (z = -3 mm)	-25.26	0.46	2.18	-24.16	-0.64	5.64
9 (z = -2 mm)	-17.46	0.34	2.58	-16.73	-0.39	4.37
10 (z = -1 mm)	-13.44	0.40	2.46	-12.01	-1.02	4.87

ภาคผนวก จ

ผลการทดสอบเบื้องต้น

ผลการทดสอบเบื้องต้นแสดงไว้ใน Table E.1 – Table E.10 โดยแบ่งการทดสอบออกเป็น 3 การทดสอบใหญ่ ๆคือ

1. การทดสอบการดึง
   1.1 การทดสอบการดึงตามเสี้ยน Table E.1 – Table E.2
   1.2 การทดสอบการดึงตั้งฉากเสี้ยน Table E.3 – Table E.4
2. การทดสอบการอัด
   2.1 การทดสอบการอัดตามเสี้ยน Table E.5 – Table E.6
   2.2 การทดสอบการอัดตั้งฉากเสี้ยน Table E.7 – Table E.8
3. การทดสอบการดัดสถิต แสดงใน Table E.9 – Table E.10

Tensile Test

Table E.1 Specific gravity determination of specimen in tensile test along grain direction

Sample	m_1 (g)	m_0 (g)	MC (%)	t_1 (mm)	t_2 (mm)	t_{ave} (mm)	l_1 (mm)	l_2 (mm)	l_{ave} (mm)	w_1 (mm)	w_2 (mm)	w_{ave} (mm)	Density (g/mm^3)	SG
xv1Pa	13.03	12.02	8.40	6.80	6.85	6.83	50.70	50.80	50.75	43.40	44.40	43.90	0.859	0.793
xv2/1Pa	10.51	9.65	8.91	6.70	6.80	6.75	50.75	50.85	50.80	43.85	44.75	44.30	0.696	0.639
xv3/2Pa	12.32	11.34	8.64	6.65	6.70	6.68	50.25	50.85	50.55	44.55	42.85	43.70	0.834	0.767
x1/2Pa	13.47	12.41	8.54	6.65	6.60	6.63	50.55	50.35	50.45	51.80	51.45	51.63	0.779	0.718
x2/1Pa	13.82	12.73	8.56	6.65	6.55	6.60	50.50	50.70	50.60	53.55	53.30	53.43	0.767	0.707
x3/1Pa	12.9	11.89	8.49	6.55	6.75	6.65	51.25	51.15	51.20	43.80	44.55	44.18	0.872	0.803
v1/2Pa	13.23	12.2	8.44	6.85	6.65	6.75	49.80	49.60	49.70	46.85	47.00	46.93	0.830	0.765
v2/2Pa	13.79	12.72	8.41	7.00	7.00	7.00	50.85	50.55	50.70	45.30	44.30	44.80	0.870	0.802
v3/2Pa	12.96	11.96	8.36	6.85	6.85	6.85	50.95	50.55	50.75	44.80	44.30	44.55	0.840	0.775

Table E.2 Results from tensile test along grain direction

Sample	Thickness (mm)	Width (mm)	Area (mm^2)	Max Load (N)	Deflection at Max Load (mm)	Stress at Max Load (MPa)	% Strain at Max Load	Stiffness (MN/m)	Young's Modulus (MPa)	Load at Break (N)	Deflection at Break (mm)
xv1Pa	6.90	39.08	269.62	15833.77	7.204	60.277	14.18	13.075	2528.57	11083.64	7.252
xv2/1Pa	6.93	39.30	272.15	13601.84	3.700	50.442	7.28	12.566	2367.24	9521.29	3.814
xv3/2Pa	6.85	38.98	266.98	15213.81	3.296	57.944	6.49	14.250	2757.14	10649.67	3.428
x1/2Pa	6.85	38.45	263.38	10973.04	3.294	41.662	6.48	11.647	2246.38	7681.13	3.320
x2/1Pa	6.70	37.10	248.57	9413.286	2.621	37.870	5.16	10.925	2232.75	6589.30	2.692
x3/1Pa	6.78	38.45	260.50	11790.47	3.522	45.261	6.93	10.269	2002.60	8253.33	3.550
v1/2Pa	7.00	38.15	267.05	10668.32	2.825	39.949	5.56	12.054	2292.95	7467.83	2.849
v2/2Pa	7.03	38.88	273.10	9190.759	2.519	33.650	4.96	12.158	2261.28	6433.53	2.652
v3/2Pa	7.00	38.95	272.65	11538.95	3.146	42.321	6.19	13.364	2489.94	8077.26	3.359

Table E.3 Specific gravity determination of specimen in tensile test perpendicular to grain direction

Sample	m_1 (g)	m_0 (g)	MC (%)	t_1 (mm)	t_2 (mm)	t_{ave} (mm)	l_1 (mm)	l_2 (mm)	l_{ave} (mm)	w_1 (mm)	w_2 (mm)	w_{ave} (mm)	Density (g/mm^3)	SG
xv1/1Pe	9.10	8.4	8.33	6.35	6.40	6.38	42.50	42.75	42.63	38.40	38.35	38.38	0.873	0.806
xv2/2Pe	6.10	5.62	8.54	6.75	6.50	6.63	38.70	38.95	38.83	34.75	34.80	34.78	0.682	0.615
xv3/1Pe	8.81	8.15	8.10	6.25	6.10	6.18	42.25	42.80	42.53	38.50	38.35	38.43	0.873	0.793
x1/1Pe	6.04	5.59	8.05	6.80	6.80	6.80	38.70	38.50	38.60	27.85	27.75	27.80	0.828	0.768
x2/2Pe	7.08	6.52	8.59	6.65	6.65	6.65	36.50	36.70	36.60	37.95	38.10	38.03	0.765	0.703
x3/2Pe	7.29	6.74	8.16	6.45	6.60	6.53	38.30	38.55	38.43	36.75	36.00	36.38	0.799	0.745
v1/1Pe	10.31	9.5	8.53	6.90	6.90	6.90	44.15	44.35	44.25	38.55	38.50	38.53	0.877	0.806
v2/1Pe	7.89	7.29	8.23	6.35	6.70	6.53	38.90	38.65	38.78	33.20	32.10	32.65	0.955	0.910
v3/1Pe	7.64	7.05	8.37	6.80	6.80	6.80	36.80	37.55	37.18	33.65	35.05	34.35	0.880	0.804

Table E.4 Results from tensile test perpendicular to grain direction

Sample	Thickness (mm)	Width (mm)	Area (mm^2)	Max Load (N)	Deflection at Max Load (mm)	Stress at Max Load (MPa)	% Strain at Max Load	Stiffness (MN/m)	Young's Modulus (MPa)	Load at Break (N)	Deflection at Break (mm)
xv1/1Pe	6.90	39.55	272.90	293.55	0.533	1.076	1.05	4.767	887.33	-	-
xv2/2Pe	6.65	38.625	256.86	225.51	0.722	0.878	1.42	1.456	287.85	-	-
xv3/1Pe	6.175	38.475	237.58	327.52	0.820	1.378	1.61	1.078	230.47	229.26	0.878
x1/1Pe	6.675	38.55	257.32	219.85	0.721	0.854	1.42	1.448	285.83	-	-
x2/2Pe	6.575	38	249.85	315.66	0.895	1.263	1.76	1.014	206.09	220.96	0.931
x3/2Pe	6.425	37.625	241.74	328.19	0.798	1.357	1.57	1.290	270.99	229.73	0.809
v1/1Pe	6.825	38.75	264.47	604.53	1.056	2.286	2.08	1.938	372.33	423.17	1.089
v2/1Pe	6.55	38.975	255.29	523.77	0.958	2.051	1.89	1.701	338.47	366.64	1.054
v3/1Pe	6.70	37.7	252.59	583.24	0.955	2.309	1.88	1.597	321.23	408.27	1.032

Compressive Test

Table E.5 Specific gravity determination of specimen in compressive test along grain direction

Sample	m_1 (g)	m_0 (g)	MC (%)	t_1 (mm)	t_2 (mm)	t_{ave} (mm)	l_1 (mm)	l_2 (mm)	l_{ave} (mm)	w_1 (mm)	w_2 (mm)	w_{ave} (mm)	Density (g/mm^3)	SG
xv1Pa	29.68	27.50	7.93	78.65	20.85	20.90	20.90	20.88	20.75	21.00	21.95	21.23	0.851	0.789
xv2Pa	27.35	25.21	8.49	78.55	20.90	21.00	20.80	20.90	21.00	21.25	21.05	21.10	0.790	0.728
xv3Pa	31.03	28.73	8.01	78.50	21.00	21.05	20.95	21.00	20.80	21.10	21.05	20.98	0.897	0.831
x1Pa	28.08	25.90	8.42	80.45	21.25	21.25	20.90	21.13	20.75	21.05	20.95	20.92	0.790	0.728
x2Pa	28.53	26.31	8.44	78.55	21.05	21.05	20.85	20.98	21.15	21.30	21.15	21.20	0.816	0.753
x3Pa	29.74	27.48	8.22	78.55	21.25	21.15	21.10	21.17	20.80	20.10	20.05	20.32	0.880	0.814
v1Pa	29.50	27.28	8.14	78.50	21.30	21.45	21.60	21.45	21.45	21.40	21.85	21.57	0.812	0.751
v2Pa	29.05	26.84	8.23	78.50	21.15	21.20	21.25	21.20	21.25	21.30	20.95	21.17	0.825	0.762
v3Pa	28.78	26.54	8.44	78.55	20.80	20.95	20.80	20.85	20.75	21.10	21.00	20.95	0.839	0.774

Table E.6 Results from compressive test along grain direction

Sample	Height (mm)	Width (mm)	Breadth (mm)	Area (mm^2)	Max Load (N)	Deflection at Max Load (mm)	Stiffness (MN/m)	Young's Modulus (MPa)	Load at Break (N)	Stress at Break (MPa)	Stress at Maximum Load (MPa)
XV1-Pa	78.65	20.88	20.23	422.40	20.219	1.046	33.658	6267.035			47.866
XV2-Pa	78.55	20.90	21.11	441.20	16.807	1.032	31.296	5571.812	8.4036	19.0473	38.095
XV3-Pa	78.55	21.00	20.98	440.58	22.699	1.002	42.102	7506.245	11.3494	25.7602	51.520
X1-Pa	82.60	21.13	20.92	442.04	17.623	0.995	34.296	6408.588	12.3363	27.9077	39.868
X2-Pa	78.55	20.98	21.20	444.78	17.395	0.836	37.552	6631.838	12.1763	27.3762	39.109
X3-Pa	78.55	21.17	20.31	429.96	18.028	0.917	36.844	6730.980	12.6196	29.3505	41.929
V1-Pa	78.50	21.45	21.57	462.68	17.243	0.879	35.357	5998.810	-	-	37.267
V2-Pa	78.50	21.20	21.17	448.80	16.899	0.897	33.120	5792.927	11.8292	26.3572	37.653
V3-Pa	78.55	20.85	20.95	436.81	16.866	0.841	33.679	6056.462	11.8063	27.0286	38.612

Table E.7 Specific gravity determination of specimen in compressive test perpendicular to grain direction

Sample	m_1 (g)	m_0 (g)	MC (%)	t_1 (mm)	t_2 (mm)	t_{ave} (mm)	l_1 (mm)	l_2 (mm)	l_{ave} (mm)	w_1 (mm)	w_2 (mm)	w_{ave} (mm)	Density (g/mm^3)	SG
xv1Pe	39.03	36.11	8.09	81.85	21.80	21.00	21.90	21.57	21.70	22.15	22.30	22.05	1.003	0.928
xv2Pe	32.54	29.98	8.54	82.75	21.90	22.10	22.10	22.03	21.65	21.65	21.45	21.58	0.827	0.762
xv3Pe	32.23	29.76	8.30	81.65	21.85	22.00	21.90	21.92	21.00	21.10	20.90	21.00	0.858	0.792
x1Pe	29.49	27.18	8.50	82.80	21.20	21.30	21.05	21.18	21.10	21.25	21.30	21.22	0.792	0.730
x2Pe	30.10	28.16	6.89	82.70	21.20	21.30	21.40	21.30	21.35	21.75	21.90	21.67	0.789	0.738
x3Pe	32.93	30.43	8.22	82.60	21.10	21.15	21.15	21.13	21.35	21.35	21.35	21.35	0.884	0.817
v1Pe	26.35	24.29	8.48	82.95	20.70	20.65	20.55	20.63	20.15	20.70	20.60	20.48	0.752	0.693
v2Pe	32.72	30.23	8.24	83.25	21.00	20.95	21.00	20.98	21.45	21.20	20.85	21.17	0.885	0.818
v3Pe	33.06	30.49	8.43	82.95	22.15	21.95	21.30	21.80	22.90	21.80	21.65	22.12	0.827	0.762

Table E.8 Results from compressive test perpendicular to grain direction

Sample	Height (mm)	Width (mm)	Breadth (mm)	Area (mm^2)	Max Load (N)	Deflection at Max Load (mm)	Stiffness (MN/m)	Young's Modulus (MPa)	Load at Break (N)	Stress at Break (MPa)	Stress at Maximum Load (MPa)
XV1-Pe	81.85	21.57	22.05	475.62	8.530	3.511	5.312	914.209	4.2652	8.9677	17.935
XV2-Pe	82.75	22.03	21.58	475.41	5.151	3.750	2.733	475.662	2.5757	5.4180	10.836
XV3-Pe	78.55	21.90	21.00	459.90	5.017	3.975	2.536	433.077	3.5122	7.6369	10.910
X1-Pe	82.80	21.18	21.22	449.44	4.022	3.209	3.029	558.110	2.8151	6.2635	8.948
X2-Pe	82.70	21.30	21.67	461.57	3.803	2.809	2.136	382.763	-	-	8.239
X3-Pe	82.60	21.13	21.35	451.13	5.771	4.386	3.120	571.229	4.0396	8.9545	12.792
V1-Pe	82.95	20.63	20.48	422.50	3.045	2.677	2.101	412.428	-	-	7.207
V2-Pe	83.25	21.17	20.98	444.15	5.629	3.675	3.369	631.440	3.9401	8.8712	12.673
V3-Pe	82.95	21.80	22.12	482.22	5.364	3.934	3.116	536.094	3.7546	7.7862	11.123

Bending Test

Table E.9 Specific gravity determination of specimen in bending test

Sample	W_1 (mm)	W_2 (mm)	W_{ave} (mm)	L_1 (mm)	L_2 (mm)	L_{ave} (mm)	t_1 (mm)	t_2 (mm)	t_{ave} (mm)	m_1 (g)	m_0 (g)	Density (g/mm^3)	MC (%)	SG
xv1	50.35	49.80	50.08	50.80	50.75	50.78	22.40	22.25	22.33	46.81	43.2	824.66	8.356481	0.761064
xv2	50.40	50.10	50.25	50.80	50.85	50.83	22.10	22.25	22.18	41.10	37.9	725.71	8.443272	0.66921
xv3	50.35	50.05	50.20	51.10	51.20	51.15	22.55	22.70	22.63	45.32	41.8	780.10	8.421053	0.719512
x1	50.70	50.85	50.78	49.85	51.40	50.63	22.30	22.25	22.28	42.25	38.98	737.89	8.388917	0.680784
x2	50.45	50.15	50.30	50.80	51.00	50.90	22.55	21.90	22.23	44.84	41.39	788.02	8.335347	0.727391
x3	50.30	50.55	50.43	50.95	50.60	50.78	22.15	22.05	22.10	42.48	39.23	750.75	8.284476	0.693314
v1	50.45	50.35	50.40	51.05	50.95	51.00	22.00	22.00	22.00	47.63	43.98	842.28	8.299227	0.777735
v2	50.40	50.45	50.43	50.70	50.80	50.75	22.60	22.45	22.53	49.31	45.54	855.44	8.278437	0.790035
v3	50.40	50.05	50.23	50.60	50.75	50.68	22.15	22.25	22.20	44.48	40.97	787.22	8.567244	0.725102
xv20L1	50.65	50.45	50.55	50.80	50.95	50.88	20.40	20.35	20.38	48.19	44.91	919.67	7.303496	0.857077
xv20L2	50.00	50.00	50.00	50.30	50.80	50.55	20.35	20.30	20.33	45.93	42.71	894.08	7.539218	0.831396
xv20L3	50.00	50.60	50.30	50.55	50.50	50.53	20.35	20.20	20.28	45.23	42.14	877.79	7.332701	0.817823
x20L1	49.80	49.75	49.78	50.60	50.60	50.60	21.20	21.20	21.20	45.80	42.56	857.76	7.612782	0.797084
x20L2	49.80	50.25	50.03	50.40	50.15	50.28	20.30	20.35	20.33	49.50	46.05	968.36	7.491857	0.900865
x20L3	48.80	49.40	49.10	49.55	50.25	49.90	20.30	20.35	20.33	41.29	38.4	829.15	7.526042	0.771114
v20L1	49.60	50.00	49.80	50.25	50.40	50.33	20.30	20.30	20.30	41.36	38.71	812.96	6.845776	0.760876
v20L2	49.90	50.20	50.05	50.65	50.40	50.53	20.65	20.40	20.53	40.65	37.59	783.19	8.140463	0.724234
v20L3	50.45	50.50	50.48	50.40	49.75	50.08	20.15	20.45	20.30	42.07	39.01	819.93	7.844143	0.760296
xv10L1	49.70	49.60	49.65	50.45	50.50	50.48	20.25	20.45	20.35	42.18	39.24	827.08	7.492355	0.76943
xv10L2	50.60	50.35	50.48	50.25	49.95	50.10	20.40	20.25	20.33	41.66	38.75	810.54	7.509677	0.753923
xv10L3	49.55	50.00	49.78	50.20	50.75	50.48	20.25	20.15	20.20	43.40	40.38	855.17	7.47895	0.79566
x10L1	49.80	50.25	50.03	50.35	50.35	50.35	20.25	20.30	20.28	43.71	40.64	855.92	7.554134	0.795804
x10L2	50.10	49.85	49.98	50.45	50.35	50.40	20.25	20.30	20.28	45.57	42.31	892.35	7.705034	0.828512
x10L3	49.25	49.25	49.25	50.55	50.40	50.48	20.25	20.30	20.28	45.18	42.04	896.40	7.469077	0.834102
v10L1	50.15	50.20	50.18	50.60	49.65	50.13	20.45	20.35	20.40	38.82	35.91	756.63	8.103592	0.699912
v10L2	49.45	50.20	49.83	50.50	50.55	50.53	20.20	20.20	20.20	40.16	37.17	789.75	8.044122	0.73095
v10L3	50.25	50.10	50.18	50.60	50.45	50.53	20.20	20.25	20.23	41.38	38.46	807.06	7.592304	0.750114

Table E.10 Results from bending test

Sample	Deep (mm)	Width (mm)	F_{MAX} (N)	Strain@ F_{MAX}	MOR (MPa)	0.1(F_{MAX})			0.4(F_{MAX})			Chord Modulus (MPa)	Comment
						Cal.	F_1	d_1	Cal.	F_2	d_2		
XV1	22.40	50.60	6089.26	0.0137	71.95	608.926	607.06	0.4257	2435.705	2435.2	1.3817	6725.129	Tension failure
XV2	22.35	50.82	5269.26	0.0142	62.27	526.926	527.27	0.3833	2107.703	2106.2	1.2764	6232.042	Tension failure
XV3	22.38	50.55	6685.19	0.0132	79.21	668.519	665.38	0.3974	2674.076	2672.3	1.3631	7334.896	Shear failure
X1	22.07	50.92	4968.71	0.0136	60.10	496.871	498.31	0.3609	1987.483	1984.4	1.3109	5715.585	Tension failure
X2	22.42	50.90	4997.07	0.0124	58.59	499.707	500.14	0.3332	1998.828	1999	1.1952	6062.463	Tension failure
X3	22.08	50.68	5928.16	0.0144	71.98	592.816	596.68	0.3463	2371.262	2374.1	1.3244	6662.043	Shear failure
V1	22.20	50.98	5325.07	0.0124	63.58	532.507	532.44	0.3608	2130.026	2129.2	1.2711	6289.717	Shear failure
V2	22.65	50.70	5724.62	0.0137	66.03	572.462	570.72	0.4017	2289.847	2289.9	1.3705	6023.962	Shear failure
V3	22.30	50.55	5373.18	0.0132	64.12	537.318	540.05	0.3589	2149.273	2147.3	1.3404	5842.169	Shear failure
XV20L1	20.72	49.83	5844.74	0.0182	81.96	584.474	581.04	0.3945	2337.897	2340.4	1.6942	6107.847	Tension failure
XV20L2	20.67	49.78	5356.83	0.0166	75.56	535.683	532.16	0.3448	2142.731	2141.5	1.4836	6429.100	Tension failure
XV20L3	20.32	50.02	5118.76	0.0185	74.35	511.876	508.83	0.4228	2047.505	2051.4	1.716	5684.572	Tension failure
X20L1	21.14	49.73	4291.51	0.0163	57.93	429.151	425.31	0.3564	1716.602	1713.6	1.4451	5037.302	Tension failure
X20L2	21.52	49.65	5344.29	0.0154	69.73	534.429	536.91	0.4849	2137.714	2134.2	1.6697	5449.258	Shear failure
X20L3	20.30	49.32	3985.69	0.0138	58.83	398.569	398.02	0.3544	1594.274	1595.6	1.5992	4663.439	Tension failure
V20L1	20.43	49.83	3788.16	0.0154	54.64	378.816	380.11	0.3553	1515.262	1518.9	1.5437	4510.551	Tension failure
V20L2	20.47	49.85	3385.55	0.0165	48.62	338.555	339.79	0.3192	1354.220	1357.9	1.4846	4086.272	Tension failure
V20L3	20.25	49.77	3608.17	0.0147	53.04	360.817	361.6	0.3135	1443.267	1442.3	1.4416	4635.933	Tension failure
XV10L1	20.28	49.85	4501.39	0.0142	65.87	450.139	447.91	0.3621	1800.556	1798.6	1.4411	6021.313	Tension failure
XV10L2	20.32	49.88	5223.65	0.0164	76.09	522.365	523.64	0.4279	2089.459	2083.9	1.7442	5664.54	Tension failure
XV10L3	20.20	50.00	4375.72	0.0157	64.34	437.572	433.5	0.3284	1750.289	1747	1.4605	5630.753	Tension failure
X10L1	20.32	50.02	5279.15	0.0173	76.68	527.915	529.68	0.4265	2111.662	2111.3	1.7522	5685.631	Tension failure
X10L2	20.30	50.08	3802.35	0.0132	55.27	380.235	380.86	0.3542	1520.942	1517.5	1.4379	5007.021	Shear failure
X10L3	20.42	49.37	5007.69	0.0172	72.98	500.769	500.64	0.4380	2003.075	2001.3	1.6608	5839.03	Tension failure
V10L1	20.25	49.78	2768.59	0.0132	40.69	276.859	276.95	0.2536	1107.434	1106.6	1.2476	4038.515	Tension failure
V10L2	20.53	49.90	3405.85	0.0128	48.58	340.585	340.73	0.3055	1362.341	1365.6	1.352	4536.113	Tension failure
V10L3	20.32	50.22	3524.83	0.0138	51.00	352.483	350.03	0.2892	1409.934	1410.5	1.3813	4609.121	Tension failure

ภาคผนวก ฉ

สมบัติเชิงกลของโอเอสแอล

สมบัติเชิงกลเหล่านี้ได้มาจากการทดสอบโอเอสแอลแบบ 0 ยกเว้นความแข็งแรงเฉือนที่ใช้ค่าจากไม้ยางพาราจริงโดยแสดงค่าในรูปของความแข็งแรงและความยืดหยุ่นจำเพาะ สัญลักษณ์ต่าง ๆที่ปรากฏในตารางมีความหมายคือ สัญลักษณ์ s ข้างหน้าหมายถึงสมบัติเชิงกลจำเพาะ และตัวห้อย @ หมายถึงที่ความยาวแถบไม้ต่าง ๆ (XV = 15 cm., X = 10 cm., V = 5 cm.) ตัวห้อย 1 และ 2 หมายถึงสมบัติเชิงกลในทิศทาง 1 และ 2 ตามลำดับ จากตารางจะเห็นได้ว่าสมบัติเชิงกลในทิศทาง 1 มีค่าต่างกันทั้ง 3 ความยาวแถบไม้ แต่สมบัติเชิงกลในทิศทาง 2 ไม่แตกต่างกัน

Table F.1 Mechanical properties of OSL and solids wood used for initial value in programming calculation

OSL Type 0 Tension	OSL Type 0 Compression
$sE^{T}_{1@XV}$ = 1305.50 MPa	$sE^{C}_{1@XV}$ = 6193.23 MPa
$sE^{T}_{1@X}$ = 936.56 MPa	$sE^{C}_{1@X}$ = 6036.01 MPa
$sE^{T}_{1@V}$ = 917.55 MPa	$sE^{C}_{1@V}$ = 5692.26 MPa
sE^{T}_{2} = 140.08 MPa	sE^{C}_{2} = 326.35 MPa
$sT_{1@XV}$ = 76.82 MPa	$sC_{1@XV}$ = 58.36 MPa
$sT_{1@X}$ = 55.98 MPa	$sC_{1@X}$ = 52.74 MPa
$sT_{1@V}$ = 49.58 MPa	$sC_{1@V}$ = 49.65 MPa
sT_{2} = 1.76 MPa	sC_{2} = 14.10 MPa
Solid Wood Shear sS_{12} = 24.60 MPa	

ผลงานตีพิมพ์เผยแพร่จากวิทยานิพนธ์

Chirasatitsin, S. ,Prasertsan. S., Wisutmethangoon. W., Kyokong. B. 2001. "Effect of Strand Configurations and Grain Directions on the Strength of Oriented Strand Lumber (OSL)", In Proc. The 15th Mechanical Engineering Network Academic Conference (ME-NETT). Vol.2, MM43-MM49. Sri Nakarin Wirot Univ. : Bangkok, Thailand.(in Thai)

Chirasatitsin, S. ,Prasertsan. S., Wisutmethangoon. W., Kyokong. B. 2005. "Mechanical Properties of Rubberwood Oriented Strand Lumber (OSL): The Effect of Strand Length". Songklanakarin J. Sci. Technol. 27(6). (*In Press, Corrected Proof*)

ผลกระทบของขนาด รูปร่าง และทิศทางของเสี้ยนของชิ้นไม้ย่อยต่อความแข็งแรงของโอเอสแอล

Effect of Strand Configuration and Grain Directions on the Strength of Oriented Strand Lumber (OSL)

รศ.ดร.สุธีระ ประเสริฐสรรพ์
ดร.วรวุธ วิสุทธิ์เมธางกูร
สมยศ จิรสถิตสิน
ภาควิชาวิศวกรรมเครื่องกล คณะวิศวกรมศาสตร์ มหาวิทยาลัยสงขลานครินทร์
หาดใหญ่ สงขลา 90112
โทร 074-212893,222250,222251,211030 ต่อ. 2250-1 โทรสาร 074-212893

ผศ.ดร.บุญนำ เกี่ยวข้อง
สำนักวิชาเทคโนโลยีอุตสาหกรรมและทรัพยากร มหาวิทยาลัยวลัยลักษณ์
222 ต.ไทยบุรี อ.ท่าศาลา จ.นครศรีธรรมราช 80160
โทร. 66-75-672301, 66-75-672303 โทรสาร. 66-75-672302

Assoc.Prof.Dr.Suteera Prasertsan
Dr.Worawut Wisutmethangoon
Mr.Somjot Chirasatitsin
Mechanical Engineering Department, Faculty of Engineering, Prince of Songkla University
Haad Yai, Songkla 90112, Thailand
Tel. 66-74-212893,222250,222251,211030 Ext. 2250-1 FAX 66-74-212893

Asst.Prof.Dr.Buhnnum Kyokong
Institute of Industrial and Resources Technology, Walailak University.
222 Thaiburi, Thasala, Nakhon Si-Thammarat 80160, Thailand
Tel. 66-75-672301, 66-75-672303 FAX. 66-75-672302

บทคัดย่อ

โครงการวิจัยนี้ศึกษาเกี่ยวกับการนำเศษไม้ยางที่มีเหลือจำนวนมากจากอุตสาหกรรมภายในประเทศมาทำเป็นไม้ประกอบโครงสร้าง โดยเทคนิคของไม้ประกอบที่จะศึกษาคือไม้ประกอบประเภทโอเอสแอล คือการนำแถบไม้หนา 1 มม. กว้าง 10 มม.และยาว 75 มม. มาเรียงซ้อนทับกัน และอัดติดด้วยกาว ความแข็งแรงของโอเอสแอลจะขึ้นอยู่กับทิศทางการวางตัวของแถบไม้ ลักษณะทางเรขาคณิตของแถบไม้ กาว และกระบวนการผลิต โครงงานนี้ต้องการศึกษาในประเด็นของขนาด ทิศทางการวางแถบไม้ ที่มีต่อความแข็งแรงของ โอเอสแอล เพื่อใช้เป็นแนวทางในการผลิตระดับอุตสาหกรรม การวิเคราะห์ความแข็งแรงของโอเอสแอลในโครงการวิจัยนี้ใช้ทฤษฎีความเค้นสูงสุดและทฤษฎีความเสียหายของไซ-ฮิลล์เป็นเกณฑ์โดยการคำนวณหาความเค้นที่เกิดขึ้นในโอเอสแอลเมื่อรับภาระใช้ทฤษฎีวัสดุที่เรียงเป็นชั้นมีสมมติฐานเบื้องต้นว่าความเสียหายที่เกิดขึ้นเกิดขึ้นที่เนื้อไม้มิใช่ที่กาวหรือช่วงรอยต่อ การเรียงตัวของแถบไม้ในโอเอสแอลเป็นแบบสมมาตร ออกแบบคานขนาด $2'' \times 6'' \times 4m$ ผลจากการวิเคราะห์หาการเรียงที่สามารถรับน้ำหนักบรรทุกได้มากที่สุดที่การเรียงมุมเป็นดังนี้คือ $[23^\circ,20^\circ,15^\circ,-42^\circ,-1^\circ]s$ โมเมนต์ดัดสูงสุดที่รับได้คือ 9,322.8 N·m ตามทฤษฎีความเค้นสูงสุด และตามทฤษฎีความเสียหายของไซ-ฮิลล์การเรียงมุมที่ได้คือ $[19^\circ,13^\circ,2^\circ,-11^\circ,-10^\circ]s$ โมเมนต์ดัดสูงสุดคือ 8,231.63 N·m เปรียบเทียบกับโอเอสแอลที่ไม่ได้เรียงมุม (แถบไม้เรียงตัวขนานกับความยาวของคาน) มีความแข็งแรงขึ้นประมาณ 33% ตาม

ทฤษฎีความเค้นสูงสุดและ 17% ตามทฤษฎีความเสียหายของไซ-ฮิลล์ ผลที่ได้จากการวิเคราะห์ยังต้องนำไปใช้ในการสร้างต่อไป

Abstract

This research studied the use of waste wood from rubber wood industry in the country to make wood-based structural composite. Oriented strand lumber or OSL was considered. It is composed of strands, roughly 1 mm. thick, 10 mm. wide, 75 mm. long. The strands are smeared with an adhesive or resin, formed a fluffy mat, pressed and heated, and consolidated into an OSL. Strength properties of OSL are affected by strand orientation, strand geometry, adhesive, and manufacturing. This paper present an effect of size and alignment of strands on the strength of OSL in order to guide for an industry. Failure theories, such as Maximum Stresses Theory and Tsai-Hill Theory, were applied for predicting the strength. Stresses in OSL were determined by laminate theory. Assuming that the failure occurs in wood, not in an adhesive or interface, was established. The results of these were that OSL 2″× 6″× 4m optimized with symmetrical lamination was oriented [23°,20°,15°,-42°,-13°]s , maximum moment of 9,322.8 N·m, by Maximum Stresses Theory and [19°,13°,2°,-11°,-10°]s , maximum moment of 8,231.63 N·m, by Tsai-Hill Theory. Comparing with parallel strand alignment of OSL, the strength of OSL optimized by these theories was increased by 33% and 17%, respectively.

1.บทนำ

ไม้เป็นวัสดุอย่างหนึ่งที่ใช้เป็นโครงสร้างมาช้านานตั้งแต่อดีตจนถึงปัจจุบัน ตัวอย่างการใช้งานก็ได้แก่ การนำไปใช้เป็นฝาบ้าน คาน ตง พื้นบ้าน เสา โครงหลังคา เป็นต้น ในอดีตที่ผ่านมา การนำไม้มาใช้นั้นจะแปรรูปจากต้นไม้เป็นไม้ท่อน แผ่นไม้แปรรูป แล้วจึงนำไปใช้งานต่อไป แต่ในช่วงสงครามโลกครั้งที่สอง ทรัพยากรไม้ถูกใช้ไปมากจนไม่สามารถปลูกทดแทนได้ทันใช้งาน จึงมีการพัฒนาแผ่นไม้อัด (plywood) ซึ่งมีลักษณะเป็นไม้แผ่นประกอบ เพื่อใช้ในการทำพื้น ผนังและหลังคาแทนแผ่นไม้แปรรูป และได้มีการพัฒนาเรื่อยมาเป็นแผ่นไม้ประกอบ (COM-PLY® panel) แผ่นชิ้นไม้อัด (particleboard) และแผ่นแถบไม้อัดเรียงเส้ียน (oriented strand board : OSB) ที่ใช้กันอยู่ในปัจจุบัน

ไม้ประกอบโครงสร้างเช่น คานหรือเสา ได้มีการพัฒนาตั้งแต่ต้นทศวรรษที่ 1970 โดยมีความพยายามที่จะใช้เศษเหลือจากการทำอุตสาหกรรมไม้เช่น เศษไม้บาง ไม้ท่อนที่มีขนาดเล็ก เป็นต้น ยกตัวอย่างไม้ประกอบที่มีใช้กันอยู่ในปัจจุบันได้แก่ แผ่นแปรรูปไม้บางประกับ (laminated veneer lumber : LVL) , แผ่นแปรรูปไม้อัดประกอบ (COM-PLY® lumber) ,แผ่นแปรรูปแถบไม้อัดขนาน (parallel strand lumber : PSL) , แผ่นแปรรูปแถบไม้อัดเรียงเส้ียน หรือโอเอสแอล (oriented strand lumber : OSL) (Malony 1996)

ขั้นตอนการผลิตโอเอสแอลโดยทั่วไปคือการนำแถบไม้ (strand) อาบกาวแล้วเรียงซ้อนทับกัน จากนั้นจึงอัดให้ได้ขนาดตามต้องการ มีโอเอสแอลที่ออกจำหน่ายสู่ตลาดบ้างแล้วโดยใช้ชื่อทางการค้าว่า TimberStrand™ LSL ผลิตโดย MacMillan Bloedel, Ltd. ประเทศแคนาดา ซึ่งปัจจุบันเปลี่ยนเป็น Trus Joist MacMillan เป็นโอเอสแอลทำจากแถบไม้ที่เป็นเศษเหลือจากไม้บาง (veneer) ยาวประมาณ 12 นิ้ว เป็นการนำเศษไม้บางมาเรียงเป็นแนวตามยาว นอกจากนั้นแล้วยังมีโอเอสแอลที่ใช้ชื่อทางการค้าอื่น ๆ อีกเช่น Scrimber™ ของประเทศออสเตรเลีย และ Zephyrwood™ ของประเทศญี่ปุ่น แต่สองชนิดหลังนี้ยังไม่ประสบความสำเร็จด้านการตลาดเท่าที่ควร

โอเอสแอลสามารถผลิตได้จากขบวนการผลิตแผ่นแถบไม้อัดเรียงเส้ียน (oriented strand board : OSB) แต่ต้องปรับปรุงกระบวนการเรียงแถบไม้เพื่อเพิ่มความแข็งแรงให้เหมาะสมกับการใช้งาน โอเอสแอลจึงกำลังอยู่ในระหว่างการพัฒนาเนื่องจากคุณสมบัติความแข็งแรงยังต้องทำการทดสอบให้ได้ค่าตามมาตรฐาน แต่น่าจะมีแนวโน้มไปในทางที่ดีในอนาคต

ในประเทศไทยมีอุตสาหกรรมไม้ยางพารา ซึ่งมีเศษเหลือจำนวนมาก เป็นที่ประมาณการว่าไม้ยางพารา 1 ไร่ จะประกอบด้วยเศษเหลือเป็นไม้ท่อนขนาดเล็ก (เส้นผ่านศูนย์กลางน้อยกว่า 15 ซม.) 24.3 ลบ.ม. และปีกไม้จากการแปรรูปในโรงเรื่อย 10.3 ลบ.ม. จากพื้นที่เพาะปลูกกว่า 12 ล้านไร่ และอายุการตัดโค่น 25 - 30 ปี จึงประมาณการได้ว่า จะมีเศษไม้ประมาณ 10.4 ล้าน ลบ.ม. ต่อปี (Prasertsan and Vanapruk, 1998)

แนวคิดเบื้องต้นของโครงการวิจัยนี้จึงต้องการนำเศษไม้ยางพารามาทำเป็นไม้ประกอบขนาดใหญ่ เพื่อทดแทนไม้เนื้อแข็งในการใช้งานเป็นไม้โครงสร้าง เทคนิคของไม้ประกอบที่จะศึกษาคือโอเอสแอล (Oriented Strand Lumber : OSL) และต้องการศึกษาในประเด็นของขนาด การเรียงตัวแถบไม้ และชนิดของกาวที่มีต่อความแข็งแรงของ โอเอสแอล เพื่อใช้เป็นแนวทางในการผลิตระดับอุตสาหกรรมต่อไป

2.ทฤษฎีความเสียหาย

ไม้ถือเป็นวัสดุออร์โทโทรปิคคือวัสดุที่มีคุณสมบัติขึ้นอยู่กับทิศทางที่ตั้งฉากกัน 3 ทิศทาง ซึ่งจะเป็นแกนหลักในการคำนวณหาความเค้นและใช้ในการพิจารณาความเสียหายด้วย แกนหลักทั้ง 3 คือ

(1) แกนตามเสี้ยน ใช้สัญลักษณ์ L หรือ 1

(2) แกนตามแนวเส้นสัมผัส ใช้สัญลักษณ์ T หรือ 2

(3) แกนตามแนวรัศมี ใช้สัญลักษณ์ R หรือ 3

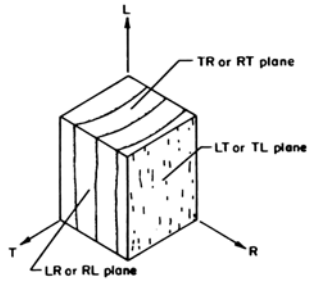
แกนหลักทั้งสามจะก่อให้เกิดระนาบที่ตั้งฉากกัน 3 ระนาบคือ

(1) ระนาบตามเสี้ยน-ตามเส้นสัมผัส ใช้สัญลักษณ์ LT หรือ 12

(2) ระนาบตามเสี้ยน-ตามรัศมี ใช้สัญลักษณ์ LR หรือ 13

(3) ระนาบตามเส้นสัมผัส -ตามรัศมี ใช้สัญลักษณ์ TR หรือ 23

ดังรูปที่ 1



รูปที่ 1 ทิศทางและระนาบต่าง ๆ ของไม้

2.1ทฤษฎีความเค้นสูงสุด (Maximum Stress Theory)

สำหรับความเค้นในระนาบ การเสียหายจะเกิดขึ้นเมื่อความเค้นในแกนหลักในแนวใดแนวหนึ่งของวัสดุเกินความต้านทานแรง (strength) ในแกนนั้นๆ

$$X > \sigma_1 \quad \text{.....(1ก)}$$

$$Y > \sigma_2 \quad \text{.....(1ข)}$$

$$S > \tau_{12} \quad \text{.....(1ค)}$$

เมื่อ σ_1 , σ_2 , τ_{12} เป็นความเค้นในระนาบในแกนหลักของวัสดุ

X, Y, S เป็นความต้านทานแรงในแกนหลักของวัสดุ

2.2ทฤษฎีความเสียหายของไซ-ฮิลล์ (Tsai-Hill Theory)

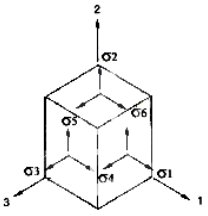
สำหรับความเค้นในระนาบแล้วเกณฑ์ที่ใช้ในการพิจารณาจะเป็นไปตามสมการ

$$\frac{\sigma_1^2}{X^2} - \frac{\sigma_1\sigma_2}{X^2} + \frac{\sigma_2^2}{Y^2} + \frac{\tau_{12}^2}{S^2} = 1 \quad \text{.....(2)}$$

ความเสียหายจะเกิดขึ้นเมื่อผลลัพธ์ทางด้านซ้ายของสมการมีค่ามากกว่า 1

3.การคำนวณหาความเค้นที่เกิดขึ้นในโอเอสแอล

โอเอสแอลที่ออกแบบเป็นคานที่มีการรองรับอย่างง่าย (simply supported beam) มีน้ำหนักบรรทุกกระทำตามขวาง (transverse load) ทำให้เกิดโมเมนต์ดัดสูงสุดที่ตำแหน่งกึ่งกลางคาน โอเอสแอลมีลักษณะเป็นชั้นที่มีการเรียงมุมต่าง ๆ กัน เพื่อให้มีความแข็งแรงสูงสุด การคำนวณความเค้นจึงอาศัยทฤษฎีวัสดุที่เรียงซ้อนเป็นชั้น (Laminate Theory) โดยความเค้นที่เกิดขึ้นในองค์ประกอบเล็ก ๆ มีสัญลักษณ์เป็นไปตามรูปที่ 2



รูปที่ 2 ความเค้นต่าง ๆ ที่เกิดขึ้นในองค์ประกอบเล็ก ๆ

σ_1 σ_2 σ_3 คือ ความเค้นตั้งฉากที่เกิดขึ้นในทิศทาง 1 2 และ 3 ตามลำดับ σ_4 σ_5 σ_6 คือ ความเค้นเฉือนที่เกิดขึ้นในระนาบ12 23 และ 13 ตามลำดับ อาจเขียนเป็น τ_{12} τ_{23} τ_{13}

3.1โอเอสแอลที่มีลักษณะเป็นวัสดุออร์โทโทรปิค

ความสัมพันธ์ระหว่างความเค้นและความเครียดในระนาบของไม้ซึ่งเป็นวัสดุออร์โทโทรปิคนั้นสามารถเขียนให้อยู่ในรูปของเมทริกซ์ได้

$$\begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix} = \begin{bmatrix}Q_{11} & Q_{12} & 0\\ Q_{12} & Q_{22} & 0\\ 0 & 0 & 2Q_{66}\end{bmatrix}\begin{Bmatrix}\varepsilon_1\\ \varepsilon_2\\ \frac{1}{2}\gamma_{12}\end{Bmatrix} \quad \text{.....(3)}$$

เมื่อ

$$Q_{11} = \frac{E_1}{1-\nu_{12}\nu_{21}}$$

$$Q_{12} = \frac{\nu_{12}E_2}{1-\nu_{12}\nu_{21}} = \frac{\nu_{21}E_1}{1-\nu_{12}\nu_{21}}$$

$$Q_{22} = \frac{E_2}{1-\nu_{12}\nu_{21}}$$

$$Q_{66} = G_{12}$$

โดยที่

E_1 คือโมดูลัสยืดหยุ่นในทิศทางตามเสี้ยน

E_2 คือโมดูลัสยืดหยุ่นในทิศทางตั้งฉากเสี้ยน

G_{12} คือโมดูลัสเฉือน

ν_{12} และ ν_{21} คืออัตราส่วนพัวส์ซอง

โดยที่ $\frac{\nu_{12}}{E_1} = \frac{\nu_{21}}{E_2}$

หรืออาจเขียนได้เป็น

$$\{\sigma\}_{12} = [Q]\{\varepsilon\}_{12} \quad \text{.....(4)}$$

สำหรับความเค้นที่เกิดขึ้นในทิศทางขนานและตั้งฉากเสี้ยนสามารถหาได้จากความเค้นที่เกิดขึ้นในทิศทางใด ๆ จากสมการ

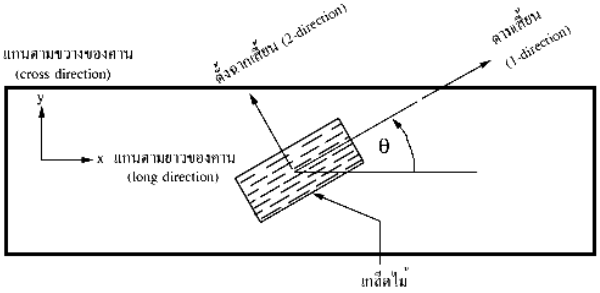
$$\begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix} = \begin{bmatrix}\cos^2\theta & \sin^2\theta & 2\sin\theta\cos\theta\\ \sin^2\theta & \cos^2\theta & -2\sin\theta\cos\theta\\ -\sin\theta\cos\theta & \sin\theta\cos\theta & -(\cos^2\theta-\sin^2\theta)\end{bmatrix}\begin{Bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{Bmatrix}$$

หรือเขียนได้เป็น

$$\{\sigma\}_{12} = [T]\{\sigma\}_{xy}$$

เมื่อ [T] คือเมทริกซ์ตัวแปลงโดยที่

$$[T] = \begin{bmatrix}\cos^2\theta & \sin^2\theta & 2\sin\theta\cos\theta\\ \sin^2\theta & \cos^2\theta & -2\sin\theta\cos\theta\\ -\sin\theta\cos\theta & \sin\theta\cos\theta & -(\cos^2\theta-\sin^2\theta)\end{bmatrix}$$



รูปที่ 3 ทิศทางตามยาวและตั้งฉากของคาน (x-direction, y-direction) และทิศทางขนาน (1-direction) และตั้งฉากเสี้ยน (2-direction)

3.2การคำนวณความเค้น

การคำนวณความเค้นในงานวิจัยนี้อาศัยทฤษฎีวัสดุที่เรียงซ้อนเป็นชั้น (Laminate Theory) ซึ่งอ้างอิงสมมติฐานของเคอร์ชอฟฟ์ (Kirchhoff Hypothesis) และสมบัติของวัสดุออร์โทโทรปิคเป็นหลัก

ความสัมพันธ์ระหว่างภาระที่ได้รับและความเครียดที่เกิดขึ้นเป็นไปตามสมการ (Jones 1975)

$$\{N\} = [A]\{\varepsilon^\circ\} + [B]\{\kappa^\circ\} \quad \text{.....(5)}$$

$$\{M\} = [B]\{\varepsilon^\circ\} + [D]\{\kappa^\circ\} \quad \text{.....(6)}$$

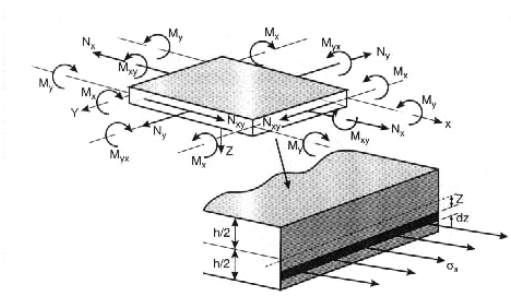
เมื่อ

$\{N\}$ คือ เวกเตอร์ของน้ำหนักบรรทุกที่กระทำตามแนวแกน ประกอบด้วย N_x, N_y และ N_{xy}

$\{M\}$ คือ เวกเตอร์ของโมเมนต์ดัดที่กระทำตามแนวแกน ประกอบด้วย M_x, M_y และ M_{xy}

$\{\varepsilon^\circ\}$ คือ เวกเตอร์ของความเครียดที่แนวกึ่งกลางหรือแนวอ้างอิง ประกอบด้วย ε_x°, ε_y° และ γ_{xy}°

$\{\kappa^\circ\}$ คือ เวกเตอร์ของความโค้งที่แนวกึ่งกลางหรือแนวอ้างอิง ประกอบด้วย κ_x°, κ_x° และ κ_x°



รูปที่ 4 ลักษณะภาระต่าง ๆ ที่กระทำต่อวัสดุที่เรียงตัวเป็นชั้น

$$A_{ij} = \sum_{k=1}^{n} \bar{Q}_{ijk}(z_k - z_{k-1})$$

$$B_{ij} = \frac{1}{2}\sum_{k=1}^{n} \bar{Q}_{ijk}(z_k^2 - z_{k-1}^2)$$

$$D_{ij} = \frac{1}{3}\sum_{k=1}^{n} \bar{Q}_{ijk}(z_k^3 - z_{k-1}^3)$$

[A] คือเมทริกซ์ความแข็งแกร่งยืดหด (Extensional Stiffness Matrix)

[B] คือเมทริกซ์ความแข็งแกร่งควบคู่ (Coupling Stiffness Matrix)

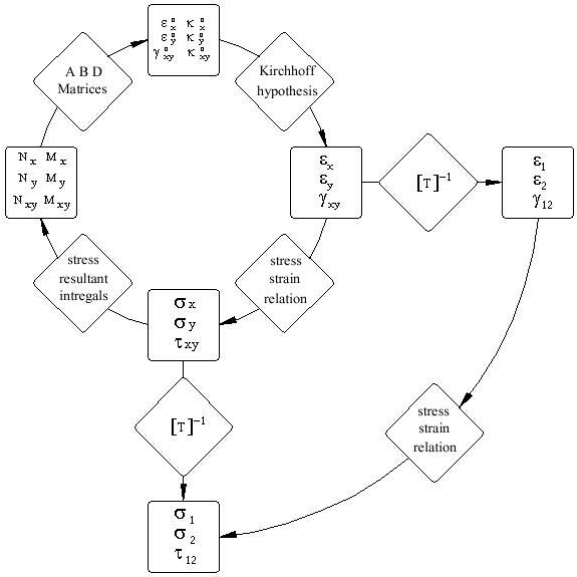
[D] คือเมทริกซ์ความแข็งแกร่งดัดโค้ง (Bending Stiffness Matrix)

z_k คือระยะจากแนวกึ่งกลางหรือแนวอ้างอิงถึงชั้นที่ k

$\bar{Q}_{ijk}$ คือสมาชิกแถวที่ i หลักที่ j ของเมทริกซ์ความแข็งแกร่ง $[\bar{Q}]$ ในชั้นที่ k

$[\bar{Q}]$ คือเมทริกซ์ความแข็งแกร่งเป็นฟังก์ชันของมุมระหว่างแกนใดๆ และแกนหลักของวัสดุ

ตัวห้อย x, y, 1, 2 แสดงถึงทิศทางของความเค้นตามแกนต่าง ๆ ตัวห้อย xy แสดงระนาบตามแกน xy ใด ๆ ตัวห้อย ij แทนสมาชิกแถวที่ i หลักที่ j ของเมทริกซ์ และ[A] [B] [D] เป็นเมทริกซ์ขนาด 3×3



รูปที่ 5 แผนภาพแสดงการคำนวณหาความเค้นในแนวขนานและตั้งฉากเสี้ยน (Hyer 1998)

วัฏจักรการหาความเค้นตามแกนหลักของวัสดุเป็นไปตามรูปที่ 5 เมื่อทราบสมบัติของวัสดุ [A] [B] [D] และทราบน้ำหนักบรรทุก (load) N_x N_y N_{xy} M_x M_y M_{xy} ก็จะหาความเครียดที่แนวอ้างอิงได้ และใช้สมมติฐานของเคอร์ชอฟฟ์เพื่อหาความเครียดของชั้นต่าง ๆ จากนั้นใช้ความสัมพันธ์ระหว่างความเค้น-ความเครียดและเมทริกซ์ตัวแปลงเพื่อคำนวณความเค้นตามแกนหลักของวัสดุ ความเค้นที่ได้จะนำไปคำนวณหาน้ำหนักบรรทุกสูงสุดต่อไป

ในกรณีที่คานรองรับอย่างง่าย และรับโมเมนต์ดัด $M_x = m_x$เพียงอย่างเดียว $N_x = N_y = N_{xy} = M_y = M_{xy} = 0$

ความเค้นที่เกิดขึ้นในแต่ละชั้นเป็นฟังก์ชันเชิงเส้นตรงกับโมเมนต์ดัด m_x (Hyer 1998)

$$\sigma_1 = A \cdot m_x$$

$$\sigma_2 = B \cdot m_x$$

$$\tau_{12} = C \cdot m_x$$

เมื่อ A B C เป็นค่าคงที่ได้จากการคำนวณตามสมการที่ (5) และ (6) เป็นความเค้นในแต่ละชั้น

แทนค่าในทฤษฎีความเค้นสูงสุดตามสมการที่ (1) เพื่อหาโมเมนต์ดัดสูงสุด

$$X = Am_x \quad \text{.....(7ก)}$$

$$Y = Bm_x \quad \text{.....(7ข)}$$

$$S = Cm_x \quad \text{.....(7ค)}$$

และแทนค่าลงสมการที่ (2) ตามทฤษฎีความเสียหายของไซ-ฮิลล์

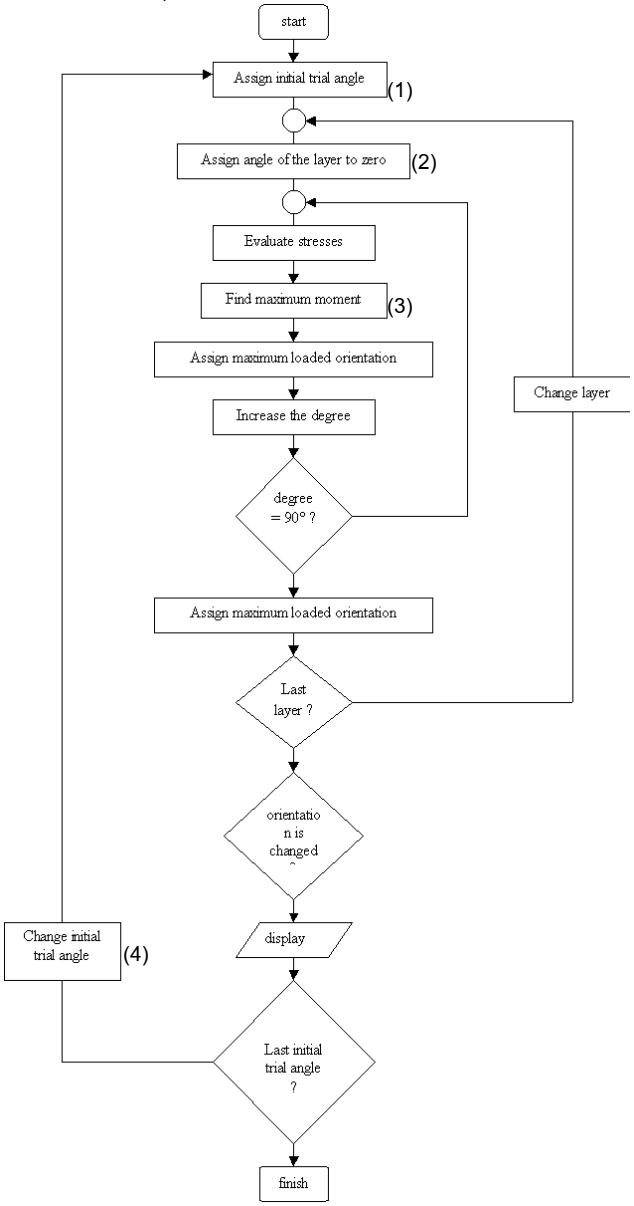
$$\frac{(A \cdot m_x)^2}{X^2} - \frac{(A \cdot m_x)(B \cdot m_x)}{X^2} + \frac{(B \cdot m_x)^2}{Y^2} + \frac{(C \cdot m_x)^2}{S^2} = 1$$

จะได้

$$m_x = \frac{1}{\sqrt{\frac{A^2}{X^2} - \frac{A \cdot B}{X^2} + \frac{B^2}{Y^2} + \frac{C^2}{S^2}}} \quad \text{.....(8)}$$

4.การเรียงตัวที่เหมาะสมของชิ้นไม้ย่อย

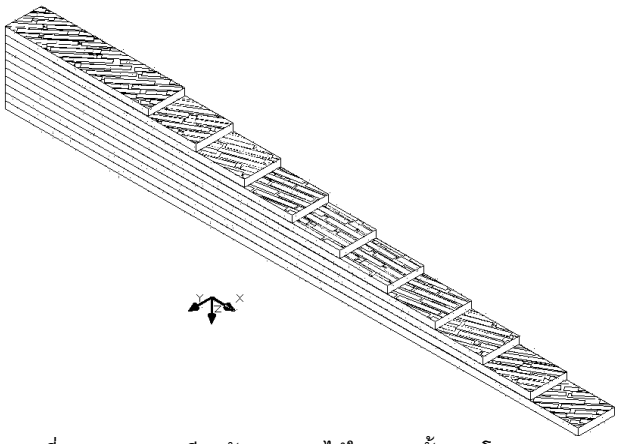
แผนภาพที่ใช้สำหรับหาการเรียงตัวที่เหมาะสมที่สุดหรือในโครงงานนี้ก็คือหาการเรียงตัวที่สามารถรับน้ำหนักบรรทุกได้มากที่สุด เป็นไปดังรูปที่ 6 แผนภาพเริ่มด้วย (1) สุ่มมุมที่จะทำการเริ่มทำการค้นหาตามวิธีค้นหาแบบสุ่มครั้งละหนึ่งตัวแปร (univariate search method) หาความเค้นในขั้นตอนที่ (2) และน้ำหนักบรรทุกที่สามารถรับได้สูงสุดในแต่ละชั้นในขั้นตอนที่ (3) ชั้นที่รับภาระได้ต่ำที่สุดจะเป็นชั้นที่กำหนดความแข็งแรงของโอเอสแอลที่การเรียงตัวนั้น ๆ เมื่อได้ภาระสูงสุดที่รับได้ในแต่ละแบบของการเรียงตัวแล้วก็จะนำมาเปรียบเทียบกันในแต่ละแบบของการเรียงตัวแล้วหาแบบการเรียงตัวที่ให้ภาระสูงสุดเป็นแบบที่ดีที่สุด จากนั้นก็สุ่มมุมเริ่มต้นใหม่(4) และทำซ้ำกระบวนการข้างต้นอีกจนครบมุมเริ่มต้นการค้นหา



รูปที่ 6 แผนภาพการหาการเรียงตัวที่เหมะสมที่สุดของโอเอสแอลโดย
หมายเลข (2) คำนวณความเค้นตามสมการที่ (5) และ (6)
หมายเลข (3) คำนวณน้ำหนักบรรทุกสูงสุดตามสมการที่ (7) และ (8)

5.ผลและวิจารณ์ผล

โอเอสแอลขนาด 2 นิ้ว × 6 นิ้ว × 4 เมตร รองรับแบบง่ายที่ปลายทั้งสองข้างของคาน รับน้ำหนักบรรทุกแบบกระจายสม่ำเสมอ และมีการเรียงชั้นแบบสมมาตร แบ่งเป็น 10 ชั้น ดังนั้นแต่ละชั้นจะมีความหนา 0.6 นิ้ว (0.0152m) การเรียงตัวที่รับน้ำหนักบรรทุกสูงสุดตามทฤษฎีความเค้นสูงสุดที่ [-23°,-20°,-15°,43°,-1°]s (สัญลักษณ์ s หมายถึงสมมาตรกับแนวกึ่งกลาง) น้ำหนักบรรทุกที่รับได้สูงสุด (maximum load) คือ 1.8373×10^{5} N·m/m (หน่วยเป็นโมเมนต์ดัดต่อหนึ่งหน่วยความกว้างของคาน) หรือโมเมนต์ดัดสูงสุดที่รับได้คือ 9,333.48 N·m คิดเป็นน้ำหนักบรรทุกแบบกระจายสม่ำเสมอได้ 4,666.74 N/m และคำนวณตามทฤษฎีความเสียหายของไซ-ฮิลล์การเรียงมุมที่ได้คือ [19°,13°,2°,-11°,-10°]s น้ำหนักบรรทุกที่รับได้สูงสุดคือ 1.6204×10^{5} N·m/m หรือโมเมนต์ดัดสูงสุดที่รับได้คือ 8,231.63 N·m คิดเป็นน้ำหนักบรรทุกแบบกระจายสม่ำเสมอได้ 4,115.82 N/m เปรียบเทียบกับโอเอสแอลที่แถบไม้เรียงตัวขนานกับแนวตามยาวของคานซึ่งรับน้ำหนักสูงสุดได้ 1.38×10^{5} N·m/m น้ำหนักบรรทุกสูงสุดตามทฤษฎีความเค้นสูงสุดและตามทฤษฎีความเสียหายของไซ-ฮิลล์มีค่ามากกว่า 33% และ 17% ตามลำดับ



รูปที่ 7 แสดงการเรียงตัวของแถบไม้ในแต่ละชั้นของโอเอสแอล

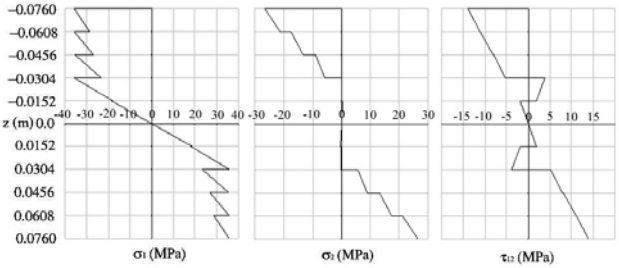
ลักษณะการเรียงตัวของแถบไม้ในแต่ละชั้นเป็นดังรูปที่ 7 ชั้นบนสุดและชั้นล่างสุดจะทำมุมกับแนวตามยาวของคานมากที่สุด -23° ตามทฤษฎีความเค้นสูงสุดและ19° ตามทฤษฎีความเสียหายของไซ-ฮิลล์ ชั้นที่อยู่ต่ำลงมาก็จะทำมุมกับแนวตามยาวของคานน้อยลง ความเค้นที่เกิดขึ้นในแนวตามและตั้งฉากเสี้ยนเป็นดังตารางที่ 1 และ 2

ตารางที่ 1 ความเค้นต่าง ๆ ที่เกิดขึ้นในแนวขนานและตั้งฉากเสี้ยนของโอเอสแอลที่เรียงมุมตามทฤษฎีความเค้นสูงสุด (MPa)

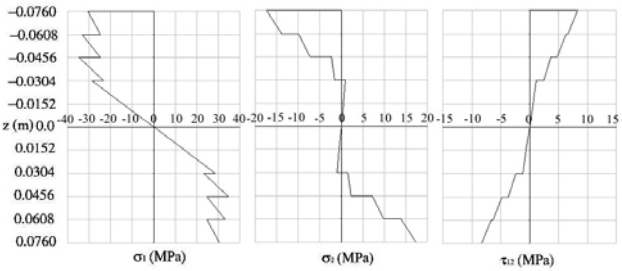
มุม	σ_1	σ_2	τ_{12}
-23° (z = −0.0760 m)	−35.89	−2.65	−13.90
-20° (z = −0.0608 m)	−35.80	−1.77	−11.00
-15° (z = −0.0456 m)	−35.50	−0.90	−7.86
43° (z = −0.0304 m)	−35.89	0.004	4.00
-1° (z = −0.0152 m)	−18.70	0.04	−1.87

ตารางที่ 2 ความเค้นต่างๆ ที่เกิดขึ้นในแนวขนานและตั้งฉากเสี้ยนของโอเอสแอลที่เรียงมุมตามทฤษฎีความเสียหายของไซ-ฮิลล์ (MPa)

มุม	σ_1	σ_2	τ_{12}
19° (z = −0.0760 m)	−30.95	−1.75	8.41
13° (z = −0.0608 m)	−33.20	−0.98	6.34
2° (z = −0.0456 m)	−35.01	−0.24	3.71
-11° (z = −0.0304 m)	−28.54	0.10	1.19
-10° (z = −0.0152 m)	−14.27	0.05	6.52



รูปที่ 8 ความเค้น ณ ตำแหน่งต่าง ๆ บนโอเอสแอลที่การเรียงตัวแบบสมมาตรตามทฤษฎีความเค้นสูงสุดที่สามารถรับน้ำหนักบรรทุกได้สูงสุด



รูปที่ 9 ความเค้น ณ ตำแหน่งต่าง ๆ บนโอเอสแอลที่การเรียงตัวแบบสมมาตรตามทฤษฎีความเสียหายของไซ-ฮิลล์เค้นสูงสุดที่สามารถรับน้ำหนักบรรทุกได้สูงสุด

โอเอสแอลที่ออกแบบตามทฤษฎีความเค้นสูงสุด ความเค้นอัดสูงสุดในแนวตามเสี้ยนที่เกิดขึ้นคือ 35.89 MPa ที่ z = −0.0760 m ในชั้นที่ 1 ซึ่งเท่ากับค่าความต้านทานแรงอัดสูงสุดของไม้ ที่ชั้นที่ 2 3 และ 4 ก็ให้ค่าใกล้เคียงกับค่าความต้านทานแรงอัดเช่นกัน คือ 35.80 MPa 35.50 MPa และ 35.89MPa ตามลำดับ ความเค้นดึงสูงสุดตามแนวเสี้ยนคือ 35.89 MPa ที่ z = 0.0760 m ซึ่งมีขนาดเท่ากับความเค้นอัดสูงสุดเนื่องจากโอเอสแอลมีความสมมาตรเปรียบเทียบกับค่าความต้านทานแรงดึงสูงสุดตามแนวเสี้ยนคือ 56.68 MPa ความเค้นอัดและความเค้นดึงสูงสุดในแนวตั้งฉากเสี้ยนมีขนาดเท่ากับ 2.65 MPa เปรียบเทียบกับค่าความต้านทานแรงอัดและแรงดึงตั้งฉากเสี้ยนซึ่งมีค่า 11.87 MPa และ 2.80 MPa ตามลำดับความเค้นเฉือนที่เกิดขึ้นสูงสุดมีขนาด 13.90 MPa โดยที่ค่าความต้านทานแรงเฉือนสูงสุดมีค่า 15.99 MPa

สำหรับโอเอสแอลที่ออกแบบตามทฤษฎีความเสียหายของไซ-ฮิลล์ ความเค้นตามแนวเสี้ยนสูงสุดมีขนาด 35.01 MPa ในชั้นที่ 3 ความเค้นตั้งฉากเสี้ยนสูงสุดมีขนาด 1.75 MPa ในชั้นที่ 1 และความเค้นเฉือนสูงสุดมีขนาด 8.41 MPa

แถบไม้ทีเบี่ยงมุมไปจากแนวตามยาวของคาน มีผลทำให้ (1) ความเค้นตามแนวเสี้ยนลดลง แต่ความเค้นตั้งฉากเสี้ยนและความเค้นเฉือนเพิ่มขึ้น และ (2) ความแข็งแกร่ง (stiffness) ของคานลดลง ส่งผลให้ความเค้นตามแนวยาวของคานเพิ่มขึ้นอีก (น้ำหนักบรรทุกยังคงเท่าเดิม) จากผลทั้งสองประการนี้ ถ้าอัตราการเพิ่มขึ้นของความเค้นที่มีผลเนื่องมาจากความแข็งแกร่งของคานที่ลดลงนั้น ทำให้ความเค้นตามแนวเสี้ยนและตั้งฉากเสี้ยนไม่เกินควาความต้านทานแรง ผลลัพธ์ก็คือคานจะสามารถรับน้ำหนักบรรทุกได้เพิ่มขึ้นอีก

ความเสียหายที่เกิดขึ้นของโอเอสแอลนั้นเกิดจากความเสียหายเนื่องจากแรงอัดตั้งฉากเสี้ยนเป็นสำคัญ ตามปกติแล้วความเค้นอัดที่เกิดขึ้นสูงสุดที่เท่ากับความต้านทานแรงอัดจะเกิดที่ชั้นบนสุดของโอเอสแอลเมื่อน้ำหนักบรรทุกถึงค่า ๆ หนึ่ง ถ้าน้ำหนักเกินกว่านั้นแล้ว จะเกิดความเสียหายที่ชั้นบนสุดก่อน เพื่อมิให้เกิดความเสียหายจึงต้องเรียงแถบไม้ในชั้นที่ถัดลงมาเพื่อลดความเค้นที่เกิดขึ้นกับชั้นบนสุดแล้วกระจายความเค้นมายังชั้นอื่น ๆ ซึ่งจะสังเกตได้จากที่การเรียงตัวที่เหมาะสมที่สุดนั้น ความเค้นจะกระจายไปยังชั้นที่สองและที่สามมากขึ้นจนมีค่าเข้าใกล้ค่าความต้านทานแรงแสดงดังรูปที่ 8

6.สรุป

การหาการเรียงตัวที่เหมาะสมโดยวิเคราะห์ความเค้นที่เกิดขึ้นจากทฤษฎีการเรียงตัวเป็นชั้นจะคำนึงถึงคุณสมบัติของแต่ละชั้น เมื่อเกิดการเปลี่ยนแปลงในชั้นใดชั้นหนึ่งความเค้นก็จะเปลี่ยนแปลงไปมีผลให้การคำนวณหาการเรียงตัวที่เหมาะสมมีความซับซ้อนมากขึ้น ผลที่ได้จากการหาการเรียงตัวที่เหมาะสมที่สุดหรือสำหรับโครงงานนี้คือการเรียงตัวที่สามารถรับน้ำหนักบรรทุกได้สูงสุดให้ผลดังนี้คือ การเรียงมุมจะเรียงจากมากไปหาน้อยจากชั้นบนสุดลงมายังชั้นที่อยู่กึ่งกลางมีผลให้เกิดการกระจายความเค้นและทำให้ความเค้นที่เกิดขึ้นในแต่ละชั้นมีค่าใกล้เคียงกันกับค่าความต้านทานแรงของไม้ แนวโน้มซึ่งเป็นไปได้สำหรับการเรียงตัวของแถบไม้ในโอเอสแอลแบบสมมาตรจึงต้องเรียงแถบไม้ในชั้นบนสุดหรือยิ่งไกลออกไปจากแนวกึ่งกลางคานให้มีมุมมากกว่าชั้นที่อยู่ใกล้กับแนวกึ่งกลางคาน ซึ่งการเรียงตัวในลักษณะนี้พบได้ในงานวิจัยของ Sharma and Sharon (1993) ด้วย

ค่าที่ได้จากการคำนวณนี้สามารถเปลี่ยนแปลงได้ขึ้นอยู่กับสมบัติของวัสดุผสมระหว่างแถบไม้และกาวโดยในโครงงานนี้ตั้งสมมติฐานว่าสมบัติทางกลอันได้แก่ โมดูลัสความยืดหยุ่นและความต้านทานแรงของโอเอสแอลเท่ากับไม้ การคำนวณหาควรใช้สมบัติของวัสดุผสมเพื่อความถูกต้องแม่นยำมากขึ้น ซึ่งอาจจะใช้ทฤษฎีวัสดุผสมเพื่อช่วยในการคำนวณ

เอกสารอ้างอิง

[1] Forest Products Laboratory. 1999. Wood Handbook-Wood as an Engineering Material. Gen. Tech. Rep. FPL–GTR–113. Madison, WI: U.S. Department of Agriculture, Forest Service, Forest Products Laboratory.

[2] Hyer, M.W. 1998. Stress Analysis of Fiber-Reinforced Composite Materials. International edition. Singapore : McGraw-Hill Book Co.

[3] Jones, R.M., 1975. Mechanics of Composite Materials. International Student Edition, Tokyo. Japan : Kosaido Printing Co., Ltd.

[4] Malony,T.M., 1996. "The Family of Wood Composite Materials",Forest Product Journal. Vol 46 no 2. , 19-26.

[5] Prasertsan, S. and Vanapruk P., 1998. "Rubber Plantation : an Overlooked Dendropower" , The 5th ASEAN Science and Technology Week., 103-111. Hanoi : ASEAN Committee on Science and Technology.

[6] Sharma V. and Sharon A., 1993. "Optimal Orientation of Flakes in Oriented Strand Board (OSB)", Experimental Mechanics. Vol 33 no 2. , 91-98.

Mechanical Properties of Rubberwood Oriented Strand Lumber (OSL): The Effect of Strand Length

S. Chirasatitsin, S. Prasertsan and W. Wisutmethangoon
Department of Mechanical Engineering, Prince of Songkla University,
Hat Yai, Thailand, 90112.
B.Kyokong
Institute of Engineering and resources Management, Walailak University,
Nakhonsithammarat, Thailand, 80160.

Abstract

Effect of strand length on mechanical properties (tension, compression and bending) of oriented strand lumber (OSL) made of rubberwood *(Hevea brasiliensis* Muell. Arg.) was reported. Three strand lengths of 50 mm, 100 mm, and 150 mm with 1 mm thickness and 15 mm width were used. The strands were mixed with 5% pMDI glue (weight basis) in a tumble mixer. The OSL specimens were formed by hot pressing process of unidirectionally aligned strands. Average specific gravity and moisture content were 0.76 and 8.34%, respectively. Tension and compression tests were carried out for directions both parallel and perpendicular to grain while bending test was performed only in parallel direction. Ultimate stresses and moduli of elasticity were examined from the stress-strain curves. It was found that for the parallel-to-grain direction, the longer strand OSL gave higher strength. The role of the strand length did not appear for the direction normal to the grain. The relationship between the mechanical properties of OSL and strand length was well described by the modified Hankinson formula.

Keywords: Rubberwood, oriented strand lumber, composite wood, strand length, mechanical properties.

สมบัติเชิงกลของแถบไม้อัดเรียงเสี้ยนจากไม้ยางพารา : ผลกระทบของความยาวแถบไม้

สมยศ จิรสถิตสิน สุธีระ ประเสริฐสรรพ์ วรวุธ วิสุทธิ์เมธางกูร

ภาควิชาวิศวกรรมเครื่องกล มหาวิทยาลัยสงขลานครินทร์

อำเภอหาดใหญ่ จังหวัดสงขลา 90110

บุญนำ เกี่ยวข้อง

สำนักวิชาวิศวกรรมศาสตร์และทรัพยากร มหาวิทยาลัยวลัยลักษณ์

จังหวัดนครศรีธรรมราช 80160

บทคัดย่อ

งานวิจัยได้ศึกษาผลของความยาวแถบไม้ที่มีต่อสมบัติเชิงกล (การดึง การอัด และการดัด) ของแถบไม้อัดเรียงเสี้ยนจากไม้ยางพารา ชิ้นทดสอบทำจากแถบไม้ยาว 50 มิลลิเมตร 100 มิลลิเมตร และ 150 มิลลิเมตร กว้าง 15 มิลลิเมตร และหนา 1 มิลลิเมตร ผสมด้วยกาว pMDI ในปริมาณ 5%โดยน้ำหนัก ค่าเฉลี่ยความถ่วงจำเพาะและปริมาณความชื้นของชิ้นทดสอบคือ 0.76 และ 8.34% ตามลำดับ การทดสอบการดึงและการอัดทำทั้งในทิศทางตามเสี้ยนและตั้งฉากเสี้ยนแต่การทดสอบการดัดทำเฉพาะทิศทางตามเสี้ยนเท่านั้น ผลการศึกษาพบว่า เมื่อความยาวแถบไม้มากขึ้นสมบัติเชิงกลในทิศทางตามเสี้ยนจะมีค่าสูงขึ้น ในขณะที่สมบัติเชิงกลในทิศทางตั้งฉากเสี้ยนไม่มีอิทธิพลจากความยาวแถบไม้ ความสัมพันธ์ระหว่างสมบัติเชิงกลของแถบไม้อัดเรียงเสี้ยนและความยาวแถบไม้สามารถอธิบายได้ด้วยสมการดัดแปลงจากสูตรแฮนคินสัน

Introduction

The quantity of discarded rubberwood (*Hevea brasilliensis* Muell. Arg.) from the replantation scheme was estimated to a figure of 11,875 tons annually (Prasertsan and Krukanont, 2003). Thailand importation of timber and wood product was estimated at around 15,000 million Baht a year (Royal Forest Department of Thailand, 2 There is an opportunity for rubberwood residue to fill the demand-supply gap, if it was properly engineered. Engineered wood commonly appears in the form of wood composite. Wood composite products are manufactured by gluing small pieces of wood together. Typical commercial products are plywood, oriented strand board (OSB) and the likes.

Oriented strand lumber (OSL) is similar to OSB, but, instead of board formation, the technology gives structural lumber from strands. MacMillan Bloedel, Ltd. developed an OSL product marketed as TimberStrand® LSL in North America and Intrallam LSL in Europe. As being used as structural timber, the required mechanical properties are substantially different from those of the boards. It can be anticipated that the strand length plays an important role in governing the mechanical properties of the lumber. In general, the OSL is manufactured from longer strands in comparison to the OSB and orientation has significant effect on the strength. Many research works revealed that increasing of the strand length has resulted in the increase of the strength and modulus of composite wood products (Post, 1958; Brumbaugh, 1960; Badejo, 1988; Barnes, 2000). But none has reported product from rubberwood. This article reports the study leading to the understanding of the effect of the strand length on the properties of rubberwood OSL.

Materials and Methods

Strands used in this study were cut from rubberwood veneer of 1 mm thickness. The width was 15 mm and the lengths were 50 mm, 100 mm and 150 mm (in grain direction). The strands were dried and conditioned in an oven for 24 hours, then mixed with 5% pMDI (by weight) and a predetermined quantity water to attain 12 % moisture content. Mat of uni-directional orientated strands was formed manually. The amount of the laid strands was targeted for a lumber having density of 800 kg/m^3 (rubberwood density is 680 kg/m^3) at 20 mm thickness. Hot pressing was

performed in 3 steps following well-known references (Moslemi, 1974; Barnes, 1979; Smith, 1980; Maloney, 1993). The mat was pressed to 9 MPa within 90 seconds and kept at this maximum pressure for 8 minutes. The press then released pressure to 6 MPa and 3.5 MPa, which again the holding time at each pressure was 8 minutes. The working temperature was 150 °C.

Specimens were cut from the hot-pressed panels (400 mm x 400 mm) Altogether, 27 specimens of dimension 38 mm x 7 mm x 254 mm (width x thickness x length) were cut in dog bone shape for tension test. The gauge length was 51 mm for tension. The dimensions of the compression specimen were20 mm x 20 mm x 80 mm (width x thickness x length). The corresponding dimensions for the bending test were 50 mm (width) x 20 mm (depth) x 350 mm (length). The experiments were conducted on a Lloyd Universal Testing Machine (150kN) at 25°C and 67%RH. Three replications of specimens with three different strand lengths were performed for each test. Prior to the tests the specimens were conditioned at 20±1°C, 65%RH for at least 24 hours. ASTM D 1037-99 procedure was applied for tension and compression tests, and JIS A5908-1994 was used for bending tests. The mechanical properties were normalized by quantifying in specific values (per specific gravity).

The modified Hankinson formula (Barnes, 2001) commonly used to describe properties of OSL, POSL, as a function of slenderness ratio (strand length per *in situ* strand thickness, ζ) was represented by

$$P_{OSL} = \frac{P_{SW}}{20 \cdot \sin^n(\arctan(2/\zeta)) + \cos^n(\arctan(2/\zeta))} \quad (1)$$

where P_{SW} is the parallel-to-grain property (strength) of solid wood and n is the experimentally determined exponent. For this study, *in situ* strand thickness was 1 mm; thus, the slenderness ratio is the strand length (in mm). Mathematically, P_{SW} was the possible maximum value Of P_{OSL}.

Results and Discussion

Specimen Physical Properties

The specific gravity and moisture content of the specimens are shown in Table 1, which gives the average values of 0.76 and 8.34%, respectively.

Tensile properties

Typical stress-strain curves of specimens tested in tension parallel and perpendicular to grain are shown in Figure 1. The ultimate tensile stress is the maximum stress appeared on the stress-strain curve and the modulus of elasticity is the slope of the secant drawn from the original to the maximum stress (ASTM E111, 1978). For the parallel-to-grain test, it was found that the OSL failed like brittle materials. The maximum stress and strain tested in the direction parallel to grain was substantially higher than that of perpendicular to grain. The OSL is relatively weak in the direction perpendicular to the grain. In some cases, a slip occurred (as appeared in Figure 1b) before the self-tightening grips firmly hold the specimens. This did not have effect on the mechanical property evaluation since the modulus was determined from the slope after the slip.

Figure 2 illustrates the data fitting by the modified Hankinson formula, which subsequently gives equation (2)

$$UTS_{OSL} = UTS_{SW}\left(\frac{1}{20\cdot\sin^{1.25}(\arctan(2/\zeta))+\cos^{1.25}(\arctan(2/\zeta))}\right) \quad (2)$$

where UTS_{SW} parallel to grain of solid wood was 50.91 MPa interpolated to SG = 0.7 (Puajindanetr and Wisuttipaet, 2003). The coefficient of determination, R^2, was 0.83. The longer strand length (or larger slenderness ratio) has resulted in the decreasing of the stress transfer angle and, hence, increase the strength. UTS_{OSL} is about 90% of UTS_{SW} as the slenderness ratio is 360.

Compressive properties

Typical stress-strain curves of compression tests were shown in Figure 3. The failure was accompanied by specimen delamination. The specimens both in direction parallel and perpendicular to grain split out in the direction normal to load direction and failed.

Similar to Figure 2, the modified Hankinson formula describing the compressive strength parallel to grain is obtained as shown in equation (3) with R^2 = 0.96 (Figure 4)

$$UCS_{OSL} = UCS_{SW}\left(\frac{1}{20\cdot\sin(\arctan(2/\zeta))+\cos(\arctan(2/\zeta))}\right) \quad (3)$$

where UCS_{SW} = 52.67 interpolated to SG = 0.7 (Puajindanetr and Wisuttipaet, 2003). Longer strand length withstands higher compressive load due to smaller stress transfer angle. UCS_{OSL} is about 90% of UCS_{SW} as the slenderness ratio is 360.

Bending Properties

Typical load-deflection curves of OSL tested in static bending were plotted at 3 different strand lengths and given in Figure 5. The bending modulus of elasticity is the slope of a chord in the linear region, which is the slope of a line drawn from 10% to 40% of the maximum stress (BS EN 310, 1993). Specimens failed under tension at the bottom surface.

The modified Hankinson formula, represented by equation (4), is used to describe the bending strength.

$$UBS_{OSL} = UBS_{SW}\left(\frac{1}{20\cdot\sin(\arctan(2/\zeta))+\cos(\arctan(2/\zeta))}\right) \quad (4)$$

where UBS_{SW} 95.45MPa interpolated to SG 0.7 (Dhonanon and Cheuwichitchan, 1980). Equation (4) is plotted in Figure 6 with R^2 of 0.89. UBS_{OSL} at 150 mm strand length was about 70% of UBS_{SW}. According to this equation, UBS_{OSL} would approach 90% of the maximum value (UBS_{SW}) for the strand length of 360 mm (1 mm thickness).

BMOE at any given slenderness ratio, was represented by

$$BMOE_{OSL} = BMOE_{SW}\left(\frac{1}{20\cdot\sin^{0.95}(\arctan(2/\zeta))+\cos^{0.95}(\arctan(2/\zeta))}\right) \quad (5)$$

where $BMOE_{SW}$ = 9.42 GPa interpolated at SG = 0.7. The data fit well with the equation as indicated by R^2 = 0.92 (Figure 7). Predicting by the equation, OSL with 360 mm of strand length and 1 mm of thickness yields modulus of elasticity in bending of 90% of $BMOE_{SW}$.

Effects of Strand Length

Specific mean values and coefficients of variation of mechanical properties tested in tension, compression and bending were given in Table 2. The coefficients of variation, although seems relatively high (0% – 35%), is in the same range of solid wood reported at 14%-34% (Wood Handbook, 1999).

For tension parallel to grain, the OSL made of longer strands exhibited higher sUTS. Statistical analysis showed that the results are highly significant ($Pr < 0.01$). However, for the perpendicular to grain direction there was no obvious effect of the strand length on the sUTS. This is explained by the very weak cohesive force (of wood itself) in this direction. As a result, the sUTS parallel to grain is about 35 times higher than that of perpendicular to grain. The sTMOE parallel to grain is about 6.5 times higher than that of the other direction. There is no significant evidence of the role of the strand length in both directions.

It was found that sUCS and sCMOE parallel to grain increased as the strand length increased ($Pr < 0.05$), but statistically insignificant for the perpendicular-to-grain direction. The sCMOE parallel to grain was about 18 times of that perpendicular to grain. The OSL with longer strand was significantly stronger than the shorter one in term of sUBS and sBMOE.

The mechanical properties of rubberwood (solid wood) and the average properties of OSL were compared at SG = 0.7 and given in Table 3. UTS parallel to grain of OSL with strand length 150 mm was slightly higher than that of the solid wood, while UTS perpendicular to grain was only 50% of the solid wood. Voids have weakened the tensile properties of the OSL, especially for direction perpendicular to grain in which the glue lines were relatively short. UCSs of OSL were less than that of the solid wood because of the delamination. Bending properties, UBS and BMOE of the OSL were not as strong as the solid wood. However, the properties follow the

modified Hankinson equation. Therefore, it is anticipated that the mechanical properties of OSL could be improved if manufactured front longer and thinner strands. The slenderness ratio (ζ) of 360 could give OSL strength appoach 90% of the solid wood. Industrial stranding machine could produce strands as thin as 0.6 mm (Lowood, 1997), which will increase the slenderness ratio and, hence, the properties of the OSL.

Conclusions

1. The longer strands increase UTS UCS UBS and BMOE parallel to grain of OSL. Mathematically, OSL properties approach the solid wood properties as the strand length increase. OSL made from longer strand will be beneficially stronger.
2. Hankinson formula can be used to determine the mechanical properties of OSL provided that the mechanical properties of the solid wood and the slenderness ratio are known.

Acknowledgments

The authors are grateful to Thailand Research Fund for financial support of this research, Department of Forest Products, Kasetsart University for sample preparation, the Wood Science and Engineering Research Unit, Walailak University for testing specimens. The authors also thank Miss Panipa Malanit for her guiding with OSL manufacturing process.

References

American Society for Testing and Materials. 1999. ASTM D1037, Philadelphia, PA. USA.

American Society for Testing and Materials. 1978. ASTM E111, Philadelphia, PA. USA.

Badejo, S.O.O. 1988. Effect of Flake geometry on Properties of cement-bonded particleboard from mixed tropical hardwoods, Wood Sci. and Tech., 22(4), 357-370.

Barnes, D. 1979. Products of Converted Lignocellulosic Materials. U.S. Patent No.4,061,819

Barnes, D. 2000. An Integrated model of the Effect of Processing Parameters on Strength Properties of Oriented Stand Wood Products, Forest Prod. J., 50(11/12), 33-42.

Barnes, D. 2001. A Model of the Effect of Strand Length and Strand Thickness on the Strength Properties on Oriented Wood Composites, Forest Prod. J., 51(2), 36-46.

Brumbaugh, J. 1960. Effect of Flake Dimension on Properties of Particle Boards, Forest Prod. J., 10(5), 243-246.

Chunwarin, W. 1980. Wood and rubberwood. Kasetsart Univ., Bangkok, Thailand. (Unpublished) (In Thai)

Dhonano, N. and Cheuwichitchan, S. 1980. The Thai Hardwoods. Royal Forest Department of Thailand. (In Thai)

European Standard. 1993. BS EN 310, UK.

Japanese Industrial Standard. 1994. JIS A5908. Japan.

Kasemset, J., Maneeted, H., Pansri, B. and Pungsuwan, D. 2000. Wood I-Joist from Rubber Wood. Final report submitted to Thailand Research Fund, Bangkok, Thailand. (In Thai)

Kollmann, F.F.P. and W.A. C 1968. Principles of Wood Science and Technology. Book 1. Springer-Verlag., New York, USA.

Lowood, J. 1997. Chapter 5 Oriented Strand Board and Waferboard, In Engineering Wood Products : A Guide for Specifiers, Designer and User. Smuiski, S. eds. PFS Research Foundation, Madison, Wis, USA.

Maloney,T.M. 1993. Modem Particleboard and Dry Process Fiberboard Manufacture. Miller Freeman Pub., San Francisco, Calif, USA.

Maloney,T.M. 1996. The Family of Wood Composite Materials, Forest Prod. J., 46(2), 19-26.

Meyers, K.L. 2001. Impact of Strand Geometry and Orientation on Mechanical Properties of Strand Composites", Master Thesis, Washington State Univ., USA.

Moslemi, A.A. 1974. Particleboard. Vol.2 Technology. Southern Illinois Univ. Press., Illinois, USA.

Nelson, S. 1997. Chapter 6 Structural Composite Lumber, In Engineering Wood Products A Guide for Specifiers, Designer and User. Smu1ski, S. eds. PFS Research Foundation, Madison, Wis, USA.

Post, P.W. 1958. Effect of Particle Geometry and Resin Content on Bending Strength of Oak Flake Board, Forest Prod. J., 8(10), 3 17-322.

Prasertsan, S. and Krukanont, P. 2003. Implication of Fuel Moisture Content and Distribution on the Fuel Purchasing Strategy of Biomass Cogeneration Power Plants. Biomass &Bioenergy. 24, 13-25.

Puajindanetr, S and Wisuttipaet, S. 2003. Cutter Usage management of Machining Process for Parawood Final report submitted to Thailand Research Fund, Bangkok, Thailand. (In Thai)

Royal Forest Department of Thailand 2002. Forestry Statistic of Thailand.

Smith, D. 1980. Consideration in Press Design for Structural Boards. *In*:Proc. 14th International Particleboard/Composite Materials Symp., Washington, USA : 95-140.

USDA Forest Service. 1999. Wood Handb., Gen. Tech. Rep. FPL–GTR–113. USDA Forest Serv., Forest Prod. Lab., Madison, Wis, USA.

Wang, S., Winistorfer, P.M., Moschler, W.M., Helton, C. 2000. Hot Pressing of Oriented Strandboard by Step-Closure, Forest Prod. J., 50(3), 28-34.

Nomenclature

n	: The experimentally determined exponent in equation (1)
ζ	: Slenderness ratio in equation (1)
%MC	: Moisture content
OSB	: Oriented strand board
OSL	: Oriented strand lumber
R^2	: Coefficient of determination
SG	: Specific gravity
SCL	: Structural composite lumber
SW	: Solid wood
sUTS	: Specific ultimate tensile stress
sTMOE	: Specific tensile modulus of elasticity

sUCS : Specific ultimate compressive stress
sCMOE : Specific compressive modulus of elasticity
sUBS : Specific ultimate bending stress
sBMOE : Specific bending modulus of elasticity

TABLE CAPTIONS

Table 1 Specific gravity (SO) based on oven-dry condition and moisture content (%MC) of specimens

Table 2 Mean and coefficient of variation (CV) of OSL mechanical prdperties tested in tension, compression and bending.

Table 3 Mechanical properties of Rubberwood (solid wood) and the average properties of OSL made from Rubberwood compared at SG = 0.7

FIGURE CAPTIONS

Figure 1. Typical tensile stress-strain curves of OSL specimens (sample of 150 mm strand length) (a) parallel to grain, (b) perpendicular to grain

Figure 2. Ultimate tensile stress (UTS) parallel to grain fitted by the modified Hankinson formula.

Figure 3. Typical compressive stress-strain curves of OSL specimens. (sample of 150 mm strand length) (a) parallel to grain, (b) perpendicular to grain

Figure 4. Ultimate compressive stress (UCS) parallel to grain fitted by the modified Hankinson formula with n = 1.25, SG = 0.7 and UCS_{SW} = 52.67 MPa.

Figure 5. Typical load-deflection curves of specimens tested in bending parallel to grain of various strand lengths.

Figure 6. Ultimate bending stress (UBS) parallel to grain fitted by the modified Hankinson formula with n = 1.00, SG = 0.7 and UBS_{SW} = 95.45 MPa.

Figure 7. Bending modulus of elasticity (BMOE) parallel to grain fitted by the modified Hankinson formula with n = 0.95, SG = 0.7 and $BMOE_{SW}$ 9.42 GPa.

Table 1 Specific gravity (SG) and moisture content (%MC) of specimens (oven-dried weight basis)

Strand length of OSL (mm)	Tension		Compression		Bending	
	Parallel to grain					
	SG [%CV]	%MC [%CV]	SG [%CV]	%MC [%CV]	SG [%CV]	%MC [%CV]
50	0.78 [2.46]	8.41 [0.49]	0.76 [1.46]	8.36 [1.41]	0.76 [4.51]	8.34 [1.92]
100	0.74 [7.12]	8.53 [0.41]	0.76 [5.73]	8.31 [2.73]	0.70 [3.44]	8.34 [0.63]
150	0.73 [11.21]	8.65 [2.94]	0.78 [6.61]	8.14 [3.74]	0.72 [6.42]	8.41 [0.54]
	Perpendicular to grain					
50	0.84 [7.22]	8.38 [1.77]	0.76 [8.25]	8.38 [1.53]	n/a n/a	n/a n/a
100	0.72 [6.48]	8.41 [3.80]	0.76 [6.27]	8.27 [1.87]	n/a n/a	n/a n/a
150	0.74 [14.50]	8.32 [2.66]	0.83 [10.68]	7.87 [10.92]	n/a n/a	n/a n/a

n/a : data not available

Table 2 Specific mean value and coefficients of variation (CV) of OSL mechanical properties

Strand length of OSL (mm)	Tension		Compression		Bending	
	Parallel to grain					
	sUTS (MPa)	sTMOE (MPa)	sUCS (MPa)	sCMOE (MPa)	sUBS (MPa)	sBMOE (GPa)
50	49.58 [13.56]	2073 [4.39]	49.65 [0.51]	5692 [7.82]	84.59 [4.08]	7.92 [3.26]
100	55.98 [4.01]	2116 [11.76]	52.74 [3.31]	6036 [8.68]	90.88 [13.04]	8.78 [8.19]
150	76.82 [2.37]	2340 [16.27]	58.36 [98.00]	6193 [12.41]	99.23 [9.51]	9.42 [7.29]
	Perpendicular to grain					
50	2.65 [13.06]	385.8 [15.18]	13.50 [20.15]	324.6 [6.99]	n/a n/a	n/a n/a
100	1.58 [25.54]	270.8 [32.52]	13.03 [18.03]	315.1 [6.21]	n/a n/a	n/a n/a
150	1.50 [14.10]	352.2 [35.36]	15.78 [19.57]	340.3 [28.59]	n/a n/a	n/a n/a

n/a : data not available ; Coefficients of variation (CV) are given in brackets [], unit in %

Table 3 Mechanical properties of rubberwood (solid wood) and the average properties of OSL made from rubberwood compared at SG = 0.7

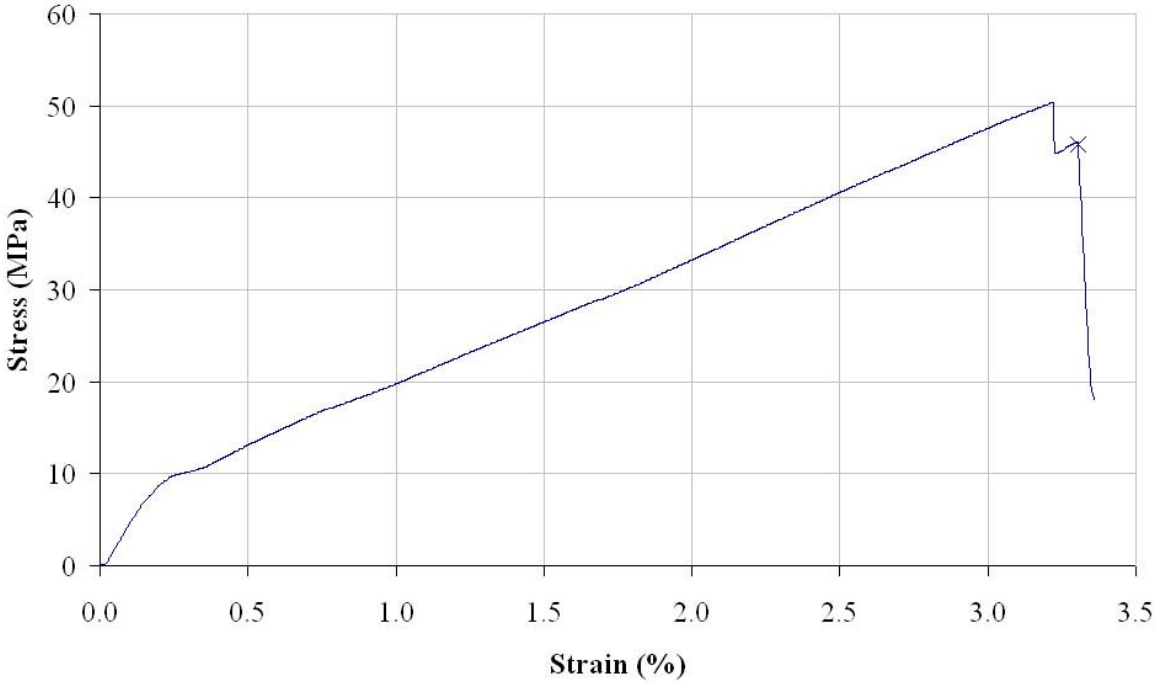
Mechanical property	Solid wood	OSL @ 50 mm	OSL @ 100 mm	OSL @ 150 mm
UTS parallel to grain (MPa)	50.91[1]	34.07	39.19	53.77
UTS perpendicular to grain (MPa)	2.80[2]	1.86	0.78	1.05
UCS parallel to grain (MPa)	52.67[1]	34.76	36.92	40.82
UCS perpendicular to grain (MPa)	11.87[3]	9.45	9.21	11.05
UBS	95.45[4]	59.21	63.62	69.46
BMOE	9.42[4]	7.76	8.39	9.23

[1] Puajindanetr and Wisuttipeat, 2003

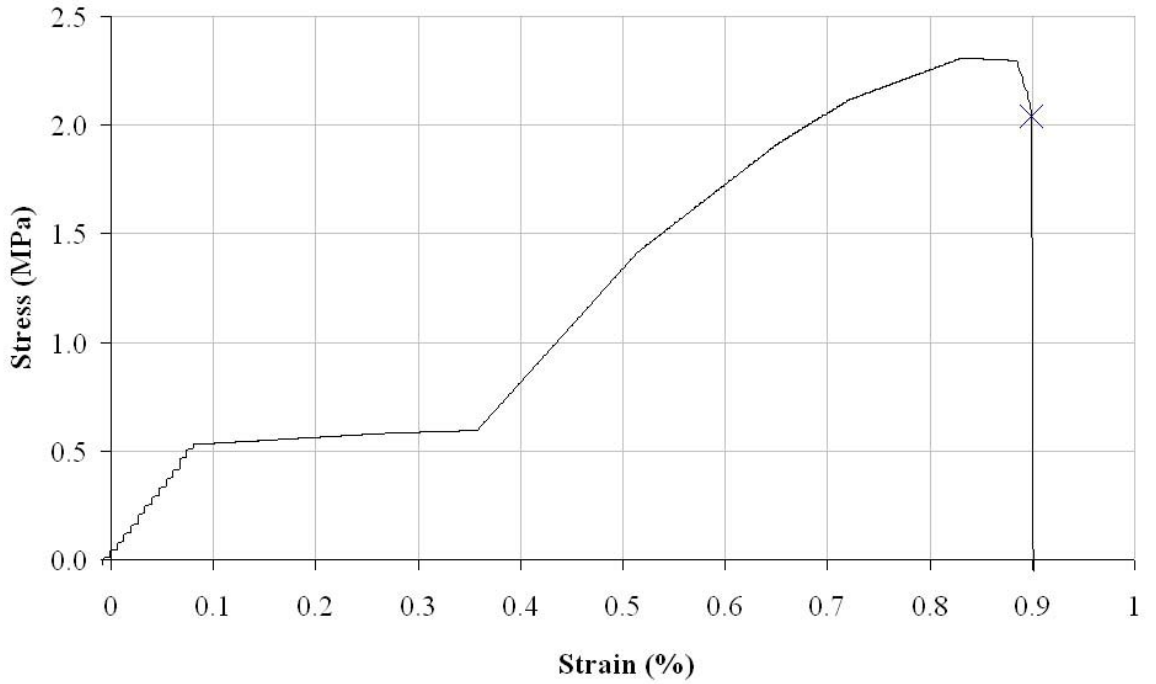
[2] Chunwarin, 1980

[3] Kasemset et al., 2000

[4] Dhonanon and Cheuwichitchan, 1980.



(a)



(b)

Figure 1. Typical tensile stress-strain curves of OSL specimens (sample of 150 mm strand length) (a) parallel to grain, (b) perpendicular to grain

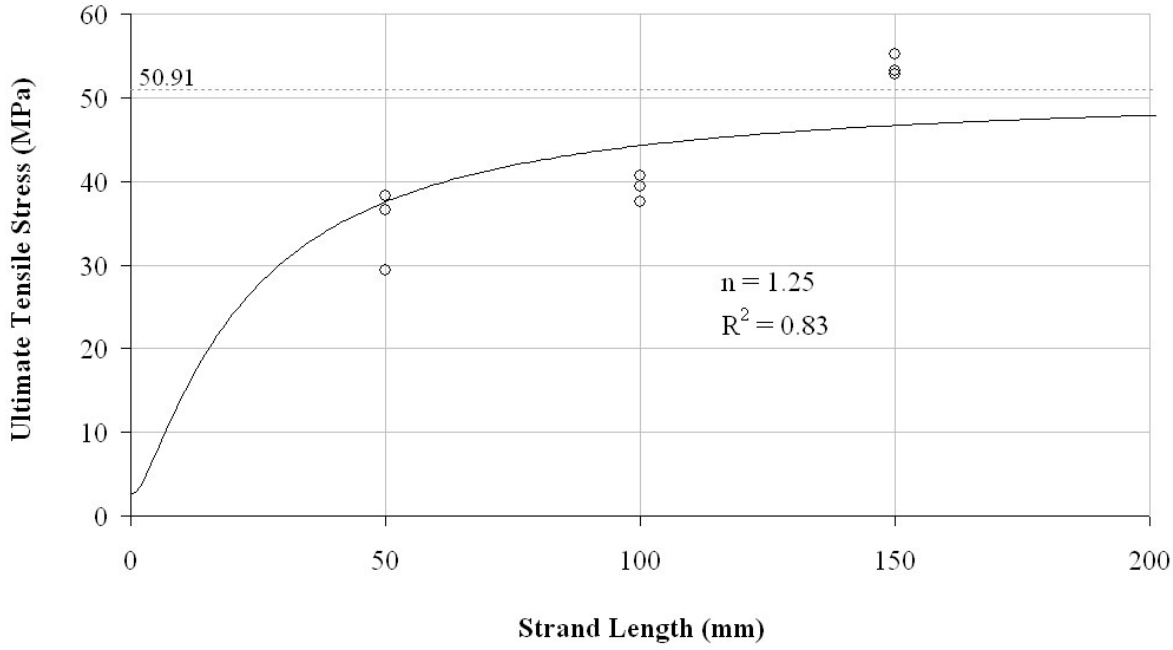
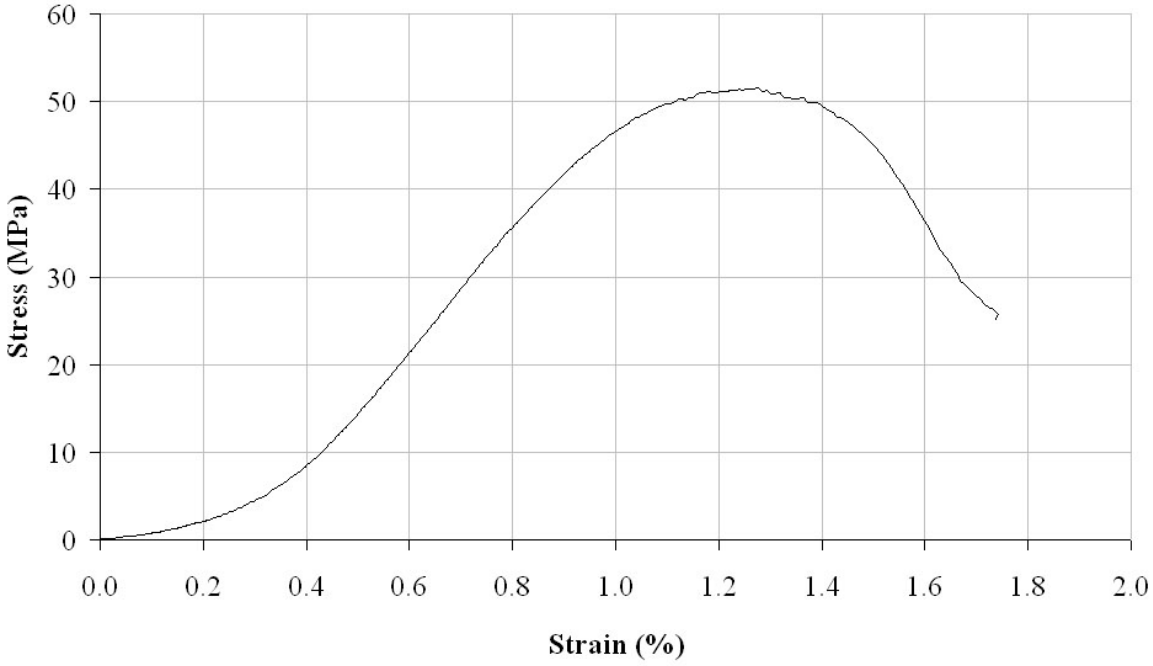
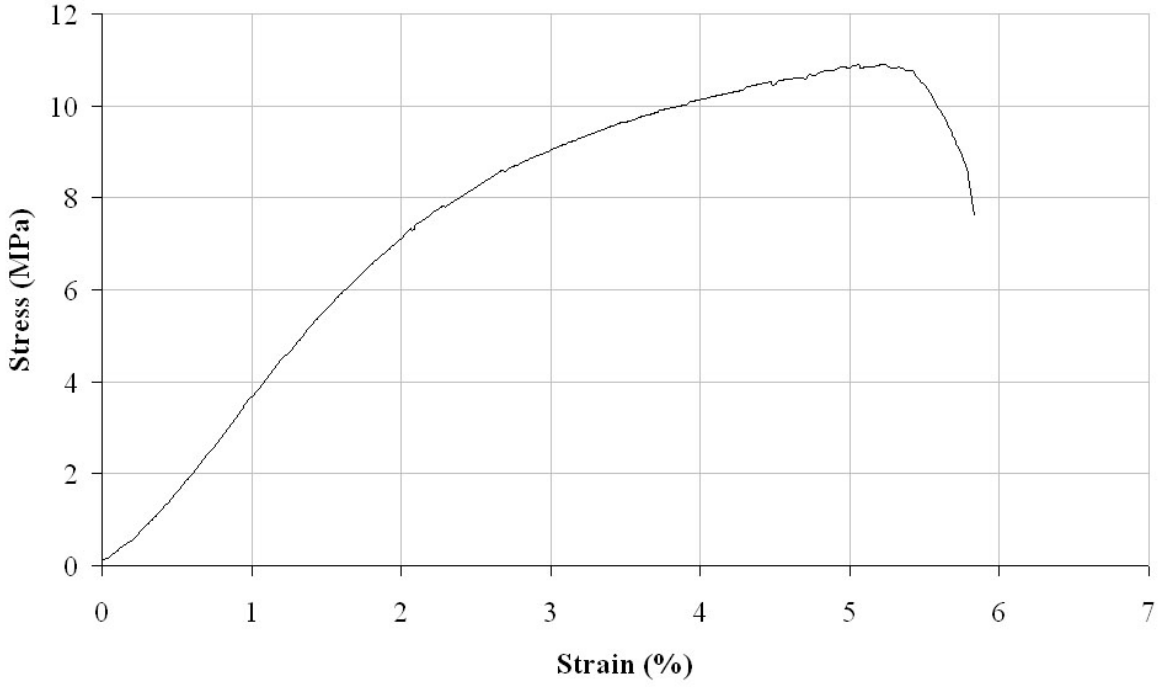


Figure 2. Ultimate tensile stress (UTS) parallel to grain fitted by the modified Hankinson formula.



(a)



(b)

Figure 3. Typical compressive stress-strain curves of OSL specimens. (sample of 150 mm strand length) (a) parallel to grain, (b) perpendicular to grain

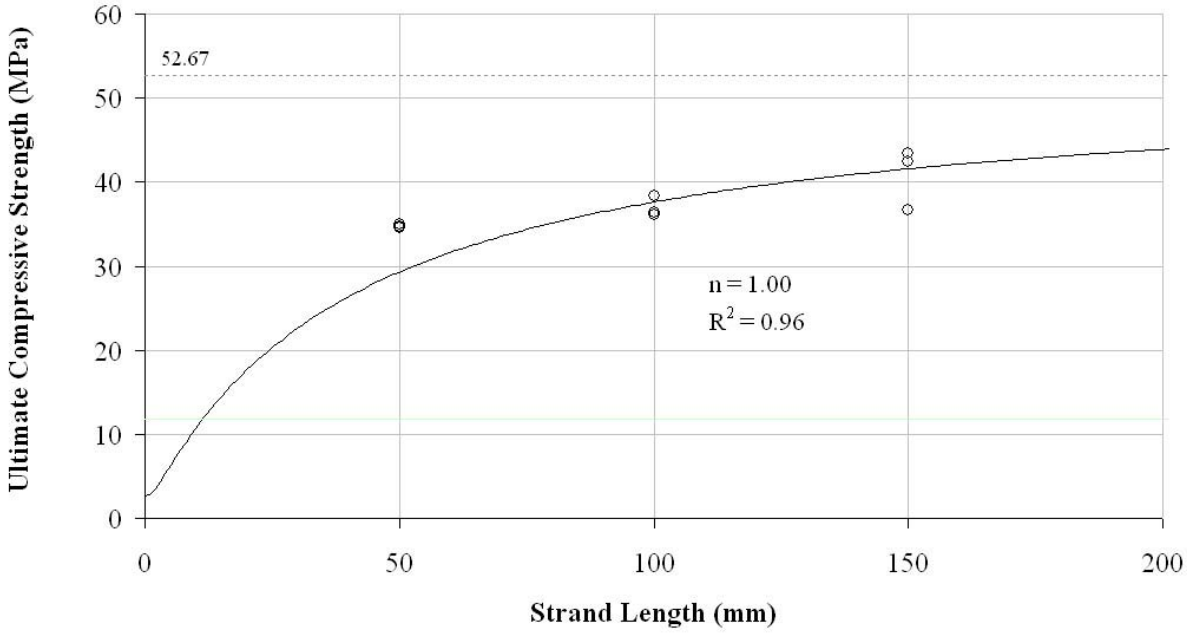


Figure 4. Ultimate compressive stress (UCS) parallel to grain fitted by the modified Hankinson formula with n = 1.25, SG = 0.7 and UCS_{SW} = 52.67 MPa.

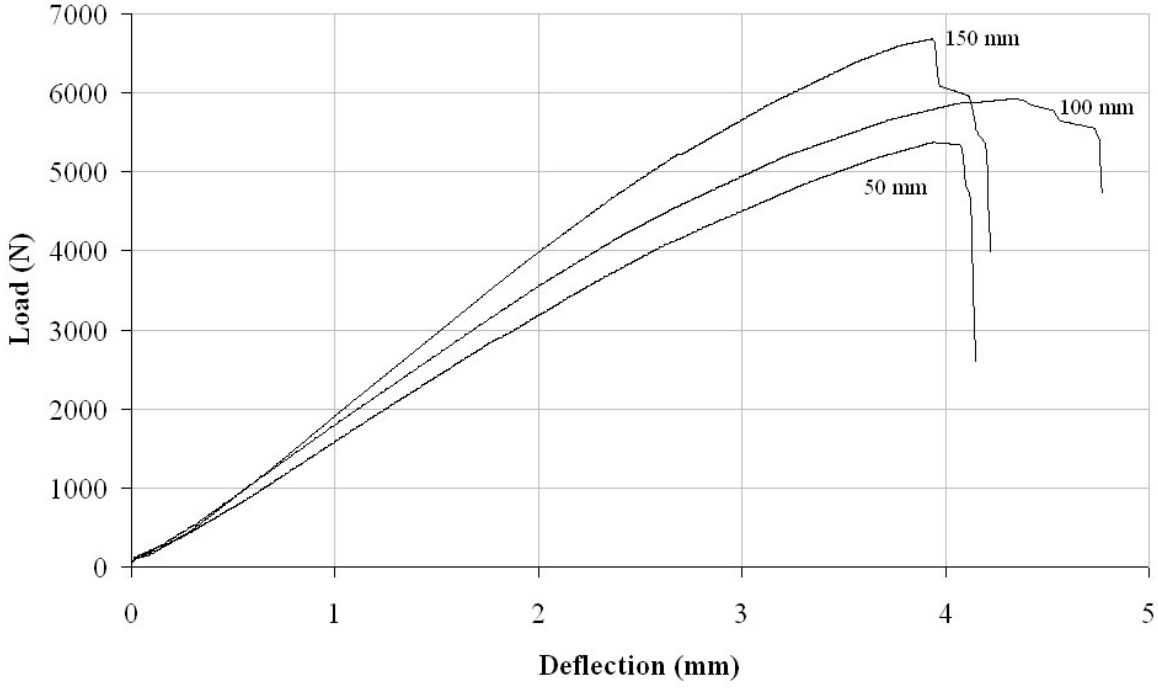


Figure 5. Typical load-deflection curves of specimens tested in bending parallel to grain of various strand lengths.

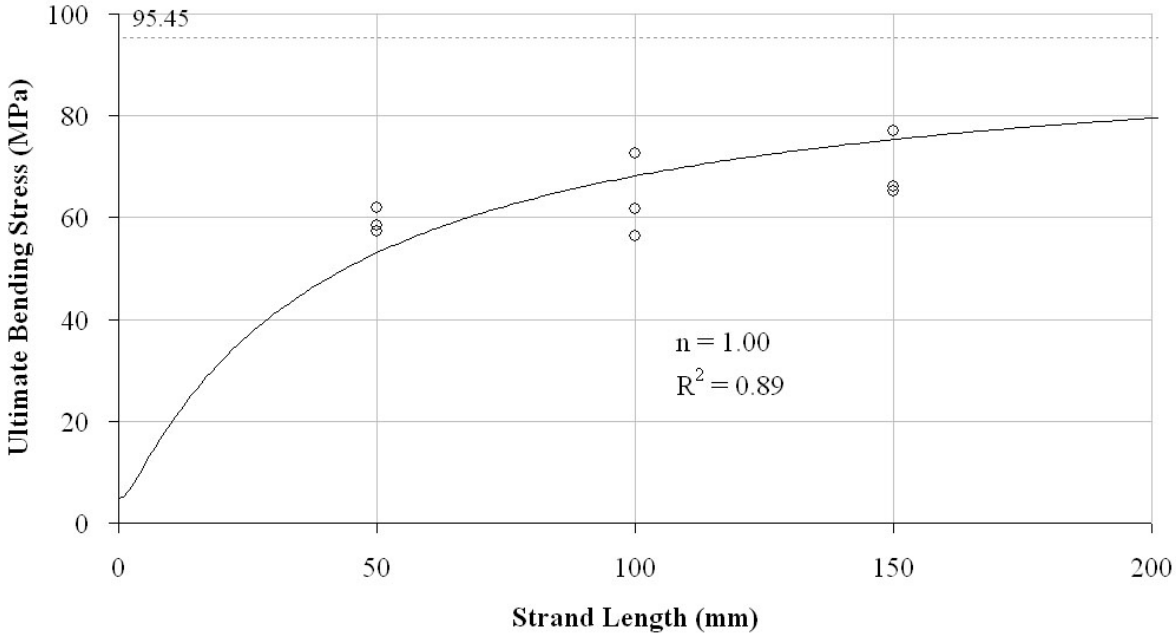


Figure 6. Ultimate bending stress (UBS) parallel to grain fitted by the modified Hankinson formula with n = 1.00, SG = 0.7 and UBS_{SW} = 95.45 MPa.

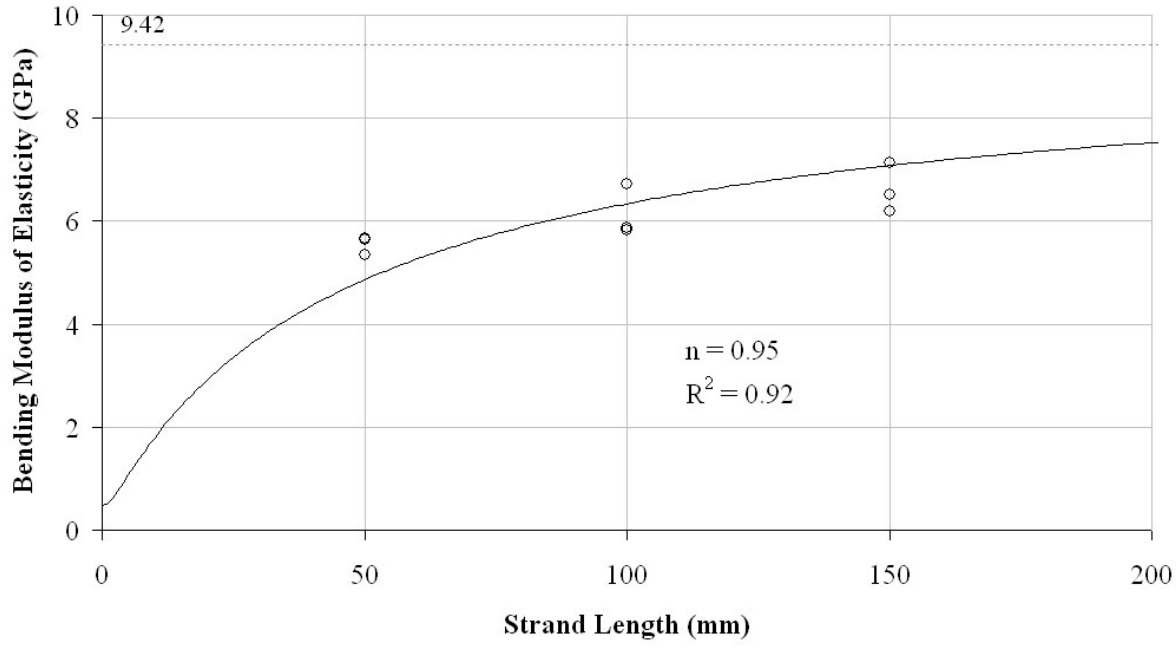


Figure 7. Bending modulus of elasticity (BMOE) parallel to grain fitted by the modified Hankinson formula with n = 0.95, SG = 0.7 and $BMOE_{SW}$ 9.42 GPa.