```
#pragma warning( disable: 4101 )
#pragma warning( disable: 4996 )
#pragma warning( disable: 6031 )
#pragma warning( disable: 6001 )
#pragma warning( disable: 6385 )
#pragma warning( disable: 6386 )
#pragma warning( disable: 26451 )

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
using namespace std;


/* 諸定数の定義 */
#define PI 3.141592653
#define C 2.99792458e8
#define Epsilon0 8.8541878e-12
#define Mu0 PI*4.0e-7

/* 関数副プログラムの読み込み */
// 屈折率波長微分
double dndl ( double lamda, double n_lamda, int mater );
// 屈折率濃度微分
double dndc ( double lamda, int mater );
// コア中心屈折率
double ncore ( double lamda, int mater );
// クラッド屈折率
double nclad ( double lamda, int mater );
// 第1種変形Bessel関数 In(x)
double  bessi0 (double x), bessi1 (double x);
// 第2種変形Bessel関数 Kn(x)
double  bessk0 (double x),  bessk1 (double x), bessk ( int n, double x );
// 係数行列要素計算関数
void S_matrix ( double *a, double *b, double *q, int m, int n, double v,
  double w, double D );
// 改訂コレスキー分解
void mcholesky ( double *a, double *b, double *ML, double *MD, int m, int
  n );
// 改訂コレスキー分解法により方程式を解く
void mcholesky_sol ( double *ML, double *MD, double *R, int m, int n );
// 逆べき乗法の初期ベクトル計算
void R0 ( double *MD, double *R, int m, int n );
// 逆べき乗法の解ベクトル規格化
void R_norm ( double *R, int n );
// 固有値計算
double Eigen ( double *R, double *a, double *b, int n, int m );
// 群遅延計算用関数
double dbdk_bunbo ( double *R, double D, double w, int m, int n );
// 群遅延計算用関数
double dbdk_bunshi ( double *R, double *q2, double D, double w, int m, int
  n);
// メモリ領域確保（整数ベクトル用）
int *dintvector ( int i, int j );
// メモリ領域解放（整数ベクトル用）
void free_dintvector ( int *a,  int i );
// メモリ領域確保（実数ベクトル用）
double *drealvector ( int i, int j );
// メモリ領域解放（実数ベクトル用）
void free_drealvector ( double *a,  int i );
// メモリ領域確保（マトリクス用）
double **dmatrix ( int nr1, int nr2, int nl1, int nl2 );
```

```cpp
// メモリ領域解放（マトリクス用）
void free_dmatrix ( double **a, int nr1, int nr2, int nl1, int nl2 );
// 2次元実数配列初期化
void init_realmatrix ( double **a, int nr1, int nr2, int nl1, int nl2 );
// 1次元整数配列初期化
void init_intvector ( int *a, int nr1, int nr2 );
// 1次元実数配列初期化
void init_realvector ( double *a, int nr1, int nr2 );
// 畳み込み積分実行関数
double* convolution ( int n1, double* P1, int n2, double* P2 );

/* 1. パラメータの定義 */
// m： モード次数( TE&TM ~ 1，EH ~ n+1，HE ~ n-1)，l： 動径方向モード次数，n： 方位角モード次数
// N： 動径座標rのコア内分割数，Nbeta： 伝搬定数の分割数，mater： 材料ID，
// lamda： 波長，A： コア半径，g： 屈折率次数，n0： コア中心の屈折率，n1： クラッドの屈折率，dr： 動径座標刻み幅
// k： 波数，delta： 比屈折率，NA： 開口数，aa： 動径座標規格化サイズ
// v： 規格化周波数，w： 規格化伝搬定数，D： 規格化コア径
// tau： 群速度，beta： 伝搬定数β，dbeta： 伝搬定数刻み幅
// a： 係数行列Sの副対角要素格納配列，b： 係数行列Sの対角要素格納配列
// Gl： コア内屈折率，q： 規格化コア内屈折率，q2： コア内屈折率分散パラメータ
// R： 規格化横方向電場成分，Rb： 逆べき乗法用入れ子配列
// ML： LDU分解係数行列のL行列副対角要素，MD： LDU分解係数行列のD行列対角要素
// dd，ds： 逆べき乗法における収束判別パラメータ，eig： 固有値
// modem： 方位角モード次数，model： 動径モード次数，modep： 主モード次数
// Nz： 総ステップ数，Nzout： ファイル出力基準ステップ数
// Li： i番目の微小区間における相対遅延ステップ数の最大値
// kim： i番目の微小区間における各モードの相対遅延ステップ数
// beta： 伝搬定数β，tau： 群遅延，taumin： 最小群遅延，taumax： 最大群遅延
// zmax： ファイバ長，dZ： 空間座標刻み幅，Tv： 時間刻み幅
// hmn： 電力結合係数，Hmmmin： H行列対角要素最小値，Hrowsum：
// fmax： 最大評価周波数，Nf： 評価周波数範囲の分割数，df： 評価周波数分解能(=fmax/Nf)
// nmax： プロファイルループ内におけるLmaxの最大値.
// nstd： 相対時間の基準値（y=0のtauimnを基準としたn=0の補正値）
// Dc： 相関長，sigma2： microbendingの軸揺らぎの分散
// wo： 摂動設定用パラメータ（0： w/o coupling，1： w/ coupling_heterogeneity，2： w/ coupling_microbending）
// ftou： ファイル出力設定用パラメータ（0： 部分出力，1： 全出力）
// matdis： 材料分散考慮パラメータ（0:無視，1:考慮）
// nP： インパルス応答格納配列用確保領域
// A00： 入力インパルス振幅，Aw00： 入力インパルス振幅のスペクトル成分合計
// 行列 A-(+)（各節点におけるモード結合前後のモードパワー分布）
// 行列 H（各節点における伝達関数）
// 配列 kim（i番目の微小区間におけるモードmの相対遅延ステップ数）


int main ( void ) {
    FILE *fp,*fp2, *fp3, *fp4,  *fq, *fr, *fr2, *fr3, *fr4, *fr5,*fr6, *fr7, *fr8, *fr9, *fs, *fs2, *fs3, *fs4;
    int         i, j, l, m, n, x, y, nr, jmax, count = 0;
    int         mater, Nl, N, Nclad, Nbeta, NLP, NLP0, Ntotal, Nwkb, Ptotal, myu, nyu, nstd, nstdmin, nstdmax, mm;
    int         Nz, Nzout, Nf, Nfp, Ti, Li, Lmax, nmax, wo, gi, ss, fout, matdis, scc, nP, Mn, launch, Nxy;
    double lamda, lamda0, lamdamin, lamdamax, lpmin, lpmax, dlp, fp0, dl, k, omega,  A, AA, g, n0, n1, dr;
    double delta, NA, aa, v, w, D, w0, r0, dx, dy, xx, yy, rr, Rxy, Ein, Emev, Emod, Amev, Amod;
    double tau, beta, dbeta, bb, eps1, eps2, sum, sumcore, sumclad, Rinf, dd, ds, eig;
    double deps2, Dc, sigma2, Db, dbmn, E_over,  hmn;
    double zmax, zout, dz, Tv, taumax, taumin, Hmmmin, Hrowsum, tauminstd, nctaumax;
    double fmin, fmax, df, fpmin, fpmax, dfp, A00, Aw00, Hw, ReHw, ImHw, me, bw, tav, rms, Ptot, ap, spct, Cpsum;
    double *Gl, *GC, *q, *q2, *R, *R2, *Rb, *a, *b, *ML, *MD, **Rlp, *Mtau, *Mbeta;
    double **Amin, **Aplu, **H, *alpha, *P, *Pm, *Pg, *M, *Cp, *Pin, *Pout, *GIND, *Spin;
    int         *model, *modem,*modep, *pdeg, *kim, *Pnum;

     /** 1. 初期設定 **/
     /* 入力ファイルの読み込み */
    if ( (fp = fopen("BW_input.csv","r")) != NULL ) {
        char s1 [81], s2 [128];
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &mater );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &g );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &A );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &zmax );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lamda0 );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &wo );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &deps2 );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &Dc );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &sigma2 );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &Db );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &launch );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &w0 );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &r0 );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dx );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &matdis );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lamdamin );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lamdamax );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Nl );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &scc );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lpmin );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lpmax );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dlp );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Ti );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &gi );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &fout );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &AA );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dr );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &aa );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Nbeta );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &eps1 );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &eps2 );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &jmax );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dz );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &Tv );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &zout );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &fmin );
         fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &fmax );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Nf );
         fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &nP ); }
     else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE ); }

     /* 入力パラメータの単位変換 */
     N = (int) ( 1000.0*A / dr ); Nclad = (int)( 1000.0*( AA - A ) / dr ); // 面内動径方向ステップ数の換算（必ずこの位置で定義）.
     if ( Nl%2 != 0 ) { Nl = Nl+1;} dl = ( lamdamax - lamdamin ) / (double) Nl;  // 材料分散評価用波長ステップ
     fpmin = C / lpmax, fpmax = C / lpmin, Nfp = int (( lpmax - lpmin ) / dlp), dfp = (fpmax-fpmin) / (double) Nfp; // 周波数領域の光源スペクトルの定義.
     lamda0 = lamda0*1.0e-9; lamdamin = lamdamin*1.0e-9; lamdamax = lamdamax*1.0e-9; dl = dl*1.0e-9; // 単位変換 (m)
     lpmin = lpmin*1.0e-9, lpmax = lpmax*1.0e-9, dlp = dlp*1.0e-9; // 単位変換 (m)
     dfp = dfp*1.0e-3; fpmin = fpmin*1.0e-3; fpmax = fpmax*1.0e-3; // 単位変換 (THz)
     A = A*1.0e-6; dr = dr*1.0e-9; AA = AA *1.0e-6; // 単位変換 (m)
     w0 = w0*1.0e-6; r0 = r0*1.0e-6, dx = dy = dx*1.0e-9; // 単位変換 (m)
     Nxy = (int) ( 5.0*w0 / dx );//とりあえず4w0の範囲
     Dc = Dc*1.0e-9, Db = Db*1.0e-3, deps2 = deps2 *(Epsilon0)*(Epsilon0); // 単位変換 (m) , 比誘電率を誘電率に変換
     fmin = fmin*1.0e-3; fmax = fmax*1.0e-3; df = ( fmax-fmin ) / (double) Nf; // 単位変換 (THz(1/ps))
     Nz = (int) ( zmax / dz ); Nzout = (int) ( zout / dz ); // ファイバ軸方向分割数およびファイル出力間隔の換算.
     D = A / aa; // 規格化コア径の換算.
     lamda = lamda0; Aw00 = 0.0; spct = Cpsum = 0.0; // 変数初期化
     xx = yy = rr = Rxy = 0.0;
     if ( matdis == 0 ) { NI = 0; }

     fclose (fp);

     /* 入力データの格納 */
     // 入射光時間波形
     Pin = drealvector ( 0, Ti ); init_realvector ( Pin, 0, Ti );
     if ( (fr5 = fopen ("Input_pulse_waveform.csv", "r") ) != NULL ) {
         for ( i = 0; i < Ti; i++ ) { fscanf ( fr5, "%lf¥n", &Pin[i] ); }}
     else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE );  }
     fclose (fr5);
     // 入射光スペクトル
     Spin = drealvector ( 0, Nfp ); init_realvector ( Spin, 0, Nfp );
     if ( (fr8 = fopen ("Source_spectrum.csv", "r") ) != NULL ) {
         for ( i = 0; i <= Nfp; i++ ) { fscanf ( fr8, "%lf¥n", &Spin[i] ); }}
     else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE );  }
     fclose (fr8);
     // 屈折率分布 (@589nm) &ドーパント濃度分布分布
     if ( gi == 1) {
         GIND = drealvector ( 0, N ); init_realvector ( GIND, 0, N );
         if ( (fr7 = fopen ("GI_profile_NaD.csv", "r") ) != NULL ) {
             for ( i = 0; i <= N; i++ ) { fscanf ( fr7, "%lf¥n", &GIND[i] ); }}
         else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE );   }
         GC = drealvector ( 0, N ); init_realvector ( GC, 0, N );
         for ( i = 0; i <= N; i++ ) { GC[i] = ( GIND[i] - GIND[N] ) / dndc ( 589.0, mater); }
         fclose (fr7);}


     /* 出力ファイルの設定*/
     // 評価条件一覧
     if ( ( fp2 = fopen("BW_setting.csv","w")) != NULL ) {}
     else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE ); }
     // 屈折率プロファイル
     if ( ( fp3 = fopen ("BW_profile.csv", "w")) != NULL ) {
         if (gi == 0) { fprintf (fp3, "r [μm],n,q,q2¥n"); } if (gi == 1) { fprintf (fp3, "r [μm],c [wt pct],n,q,q2¥n"); }}
     else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE ); }
     // 有限要素法計算結果
     if ( ( fp4 = fopen ("FEM_result.csv", "w")) != NULL ) {}
     else { printf (" U cannot open the file !¥n"); exit ( EXIT_FAILURE ); }
     // 電力結合係数計算結果
     if ( ( fq = fopen ( "Hmn_result.csv", "w" )) != NULL ) {
         if ( wo == 1 ) {
             fprintf ( fq, "μ,m (mode μ),l (mode μ),p (mode μ),β (mode μ),ν,m (mode ν),l (mode ν),p (mode ν),β (mode ν),Δp,|Δβ|,E-field overlap,hμν¥n");}
         if ( wo == 2 ) {
             fprintf ( fq, "μ,m (mode μ),l (mode μ),p (mode μ),β (mode
                 μ),ν,m (mode ν),l (mode ν),p (mode ν),β (mode ν),Δp,|Δβ|,hμν¥n");}}
     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
     // 行列H
     if ( ( fr = fopen ( "CPE_Hmatrix.csv", "w" ) ) != NULL ) {}
     else { printf ( "The file cannot be opened !¥n" ); exit ( EXIT_FAILURE ); }
     // インパルス応答波長成分 P
     if ( ( fr2 = fopen ( "CPE_Impulse-responce.csv", "w" ) ) != NULL ) {}
     else { printf ("The file cannot be opened !¥n"); exit ( EXIT_FAILURE ); }
     // 出射波形波長成分 Pout
     if ( ( fr6 = fopen ( "CPE_Output-pulse-waveform.csv", "w" ) ) != NULL ) {}
     else { printf ("The file cannot be opened !¥n"); exit ( EXIT_FAILURE ); }
     // モードパワー分布 Pm
     if ( ( fr3 = fopen ( "CPE_Mode-power-distribution.csv", "w" ) ) != NULL ) {}
     else { printf ("The file cannot be opened !¥n"); exit ( EXIT_FAILURE ); }
     // モード群パワー分布 Pg
     if ( ( fr9 = fopen ( "CPE_Group-power-distribution.csv", "w" ) ) != NULL ) {}
     else { printf ("The file cannot be opened !¥n"); exit ( EXIT_FAILURE ); }
     // 周波数応答 M
     if ( ( fr4 = fopen ( "CPE_Frequency-respnse.csv", "w" ) ) != NULL ) {
         fprintf ( fr4, "," ); for ( j = 0; j <= Nf; j++ ) { fprintf ( fr4, "%f,", (fmin*1.0e3)+(double)j*(df*1.0e3) ); } fprintf ( fr4, "¥n" ); }
     else { printf ("The file cannot be opened !¥n"); exit ( EXIT_FAILURE ); }
     // 主要な結果
     if ( ( fs = fopen ( "BW_result.csv", "w" )) != NULL ) {
         if ( matdis == 0) { fprintf ( fs, "g value,NIp,NIp0,tstart [ns],length[m],pulse broadening [ps],-3dB bandwidth [GHz],tav [ps],Ptot,Aw00,spct¥n"); }
         if ( matdis == 1) { fprintf ( fs, "g value,length[m],pulse broadening [ps],-3dB bandwidth [GHz],tav[ps],Ptot,Aw00,spct¥n"); }}
 //         else { fprintf ( fs, "g value,length[m],-3dB bandwidth [GHz],rms width [ps],tav[ps],Ptot,Aw00,spct¥n"); }}
     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
     // インパルス応答
     if ( ( fs3 = fopen ( "BW_Impulse-responce.csv", "w" )) != NULL ) {
         for ( n = 0; n <= nP; n++ ) { fprintf ( fs3, "%f,", double(n)*Tv ); } fprintf ( fs3, "¥n" ); }
     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
     // 出射波形波長成分 Pout
     if ( ( fs4 = fopen ( "BW_Output-pulse-waveform.csv", "w" ) ) != NULL ) {}
     else { printf ("The file cannot be opened !¥n"); exit ( EXIT_FAILURE ); }
     // 光源スペクトル
     if ( ( fs2 = fopen ( "BW_source-spectrum.csv", "w" )) != NULL ) {
             for ( i = 0; i <= Nfp; i++ ) { fprintf ( fs2, "%f,%f,%f¥n", fpmin+double(i)*dfp,1.0e-3*C / ( fpmin+double(i)*dfp ), Spin[Nfp-i] ); }}
     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }

     printf ( "Total spatial steps Nf: %d¥n", Nf );


```

```cpp
    /** 2. 計算開始 **/
        /* 変数の初期化 */
        Aw00 = tauminstd = 0.0; nmax = nstd = nstdmin = nstdmax = Mn = 0;
        /* 配列の記憶領域確保および初期化 */
        if ( matdis == 1 ) { P = drealvector ( 0, nP ), init_realvector ( P, 0, nP ); }
        Pnum= dintvector ( 0, Nl ); init_intvector ( Pnum, 0, Nl );
        M = drealvector ( 0, Nf ); init_realvector ( M, 0, Nf );

        for ( y = 0; y <= Nl; y++ ) {
            /* 波長指定 */
            if ( matdis == 1 ) { lamda = lamdamax -  y*dl; }
            /* 各種パラメタ計算 */
            if ( gi == 0 ) { n1 = ncore ( lamda*1.0e9, mater ); n0 = nclad ( lamda*1.0e9, mater ); }
            if ( gi == 1 ) { n1 = dndc ( lamda*1.0e9, mater )*GC[0] + nclad ( lamda*1.0e9, mater ); n0 = nclad ( lamda*1.0e9, mater ); }

            delta = (n1*n1-n0*n0) / (2.0*n1*n1), NA = sqrt (n1*n1-n0*n0);
            k = 2.0*PI / lamda; omega = 2.0*PI*C / lamda; v = k*aa*n1*sqrt ( 2.0*delta );
            Nwkb = int( (1.0/4.0)*( g / (g+2.0) )*(k*k)*(n1*n1)*delta*(A*A) );
            /* 変数の初期化 */
            bb = -1; dbeta=  k*( n1 - n0 ) / (double) Nbeta;
            NLP= 0;  NLP0 = Ntotal = Ptotal = 0;
            /* 配列の記憶領域確保および初期化 */
            GI = drealvector ( 0, N ); init_realvector ( GI, 0, N );
            q = drealvector ( 0, N ); init_realvector ( q, 0, N );
            q2 = drealvector ( 0, N ); init_realvector ( q2, 0, N );
            R= drealvector ( 0, N ); init_realvector ( R, 0, N );
            R2 = drealvector ( 0, N ); init_realvector ( R2, 0, N );
            Rb = drealvector ( 0, N ); init_realvector ( Rb, 0, N );
            a = drealvector ( 0, N ); init_realvector ( a, 0, N );
            b = drealvector ( 0, N ); init_realvector ( b, 0, N );
            ML = drealvector ( 0, N ); init_realvector ( ML, 0, N );
            MD = drealvector ( 0, N ); init_realvector ( MD, 0, N );
            modem = dintvector ( 0, 2*Nwkb ); init_intvector ( modem, 0, 2*Nwkb );
            model = dintvector ( 0, 2*Nwkb ); init_intvector ( model, 0, 2*Nwkb );
            modep = dintvector ( 0, 2*Nwkb ); init_intvector ( model, 0, 2*Nwkb );
            pdeg = dintvector ( 0, Nwkb ); init_intvector ( pdeg, 0, Nwkb );
            Mbeta = drealvector ( 0, 2*Nwkb ); init_realvector ( Mbeta, 0, 2*Nwkb );
            Mtau = drealvector ( 0, 2*Nwkb ); init_realvector ( Mtau, 0, 2*Nwkb );
            Rlp = dmatrix ( 0, 2*Nwkb, 0, N ); init_realmatrix ( Rlp, 0, 2*Nwkb, 0, N );
            /* 屈折率分布, 比屈折率分布および波長微分分布 */
            if (gi == 0) {
                for ( j = 0; j <= N; j++ ) { GI[j] = n1*sqrt ( 1.0-2.0*delta*pow (((double) j / (double) N), g) ); }
                for ( j = 0; j <= N; j++ ) { q[j] = (GI[j]*GI[j] - n0*n0) / (n1*n1 - n0*n0); }
                for ( j = 0; j <= N; j++ ) { q2[j] = GI[j]*GI[j] - (lamda*GI[j]*dndl (lamda*1.0e9, GI[j], mater)) / (1 - (lamda/GI[j])*dndl (lamda*1.0e9, GI[j], mater)); }
                if ( y == Nl/2 || fout == 1 ) { for ( j = 0; j <= N; j++ ) { fprintf ( fp3, "%f, %f, %f, %f¥n", A*1.0e6* (double) j / (double) N, GI[j], q[j], q2[j] ); }}}
            if (gi == 1) {
                for ( j = 0; j <= N; j++ ) { GI[j] = dndc ( lamda*1.0e9,
                    mater )*GC[j] + nclad ( lamda*1.0e9, mater ); } // 評価波長
                    における屈折率分布に換算
                for ( j = 0; j <= N; j++ ) { q[j] = (GI[j]*GI[j] - GI[N]*GI
                    [N]) / (GI[0]*GI[0] - GI[N]*GI[N]); }
                for ( j = 0; j <= N; j++ ) { q2[j] = GI[j]*GI[j] - (lamda*GI
                    [j]*dndl (lamda*1.0e9, GI[j], mater)) / (1 - (lamda/GI[j])
                    *dndl (lamda*1.0e9, GI[j], mater)); }
                if ( y == Nl/2 || fout == 1 ) { for ( j = 0; j <= N; j++ )
                    { fprintf ( fp3, "%f, %f, %f, %f, %f¥n", A*1.0e6* (double)
                    j / (double) N, GC[j], GI[j], q[j], q2[j] ); }}}

            /* 評価条件の出力 */
            if (  y == 0 ) {
                fprintf ( fp2, "Material, mater, %d¥n", mater );
                fprintf ( fp2, "Central wavelength, λ0, %f nm¥n",
                    lamda0*1e9 );
                fprintf ( fp2, "Step size of wavelength, dl, %f nm¥n",
                    dl*1e9 );
                fprintf ( fp2, "Partition number of evaluated wavelength, Nl,
                     %d¥n", Nl );
                fprintf ( fp2, "Core radius, A, %f μm¥n", A*1e6 );
                fprintf ( fp2, "Analysis region in radial axis, AA,%f μm¥n",
                     AA*1e6 );
                fprintf ( fp2, "Refractive index at the core center, n1,%f
                    ¥n", n1 );
                fprintf ( fp2, "Refractive index in the cladding, n0,%f¥n",
                    n0 );
                fprintf ( fp2, "Relative refractive index, Δ,%f¥n", delta );
                fprintf ( fp2, "Numerical aperture, NA, %f¥n", NA );
                fprintf ( fp2, "Step size of the elements, dr, %f, nm¥n",
                    dr*1e9 );
                fprintf ( fp2, "Step size of propagation constant, dβ,%f¥n",
                     dbeta );
                fprintf ( fp2, "Partition number of propagation constant,
                    Nβ,%d¥n", Nbeta );
                fprintf ( fp2, "Partition number of fiber core radius, N,%d
                    ¥n", N );
                fprintf ( fp2, "Partition number of fiber cladding, Nclad,%d
                    ¥n", Nclad );
                fprintf ( fp2, "Maximum allowable error for convergence
                    solution vector, eps1,%f¥n", eps1 );
                fprintf ( fp2, "Maximum allowable error of zero eigen value,
                    eps2,%f¥n", eps2 );
                fprintf ( fp2, "Maximum number of iterations in inverse power
                     method, jmax,%d¥n", jmax );
                fprintf ( fp2, "Correlation length, Dc, %e, nm¥n", Dc*1e9 );
                fprintf ( fp2, "Mean square of dielectric constant
                    fluctuation, <dε2>, %e¥n", deps2 );
                fprintf ( fp2, "Total fiber length,zmax,%e,m¥n", zmax );
                fprintf ( fp2, "Step size of fiber length,dz,%e,m¥n", dz );
                fprintf ( fp2, "Step size of time,Tv,%e,ps¥n", Tv );
                fprintf ( fp2, "Total spatial steps,Nz,%d¥n", Nz );
                fprintf ( fp2, "File output step interval,Nzout,%d¥n",
                    Nzout );
                fprintf ( fp2, "Minimum evaluated frequency,fminx,%e,GHz¥n",
                    fmin*1.0e3 );
                fprintf ( fp2, "Maximum evaluated frequency,fmax,%e,GHz¥n",
                    fmax*1.0e3 );
                fprintf ( fp2, "Step size of evaluated frequency,df,%e,GHz
                    ¥n", df*1.0e3 );
                printf ( "Central wavelength λ0: %f nm¥n", lamda0*1e9 );
                printf ( "Step size of wavelength dl: %f nm¥n", dl*1e9 );
                printf ( "Partition number of evaluated wavelength Nl: %d¥n",
                     Nl );
                printf ( "Minimum evaluated spectral frequency, fpmin, %f THz ¥n", fpmin );
                printf ( "Maximum evaluated spectral frequency, fpmax, %f THz ¥n", fpmax );
                printf ( "Core radius A: %f μm¥n", A*1.0e6 );
                printf ( "Analysis region AA: %f μm¥n", AA*1e6 );
                printf ( " Refractive index at the core center n1: %f¥n", n1 );
                printf ( "Refractive index in the cladding n0: %f¥n", n0 );
                printf ( "Relative refractive index Δ: %f¥n", delta );
                printf ( "Numerical aperture NA: %f¥n", NA );
                printf ( "Step size of the elements dr: %f nm¥n", dr*1.0e9 );
                printf ( "Partition number of fiber core radius N: %d¥n", N );
                printf ( "Partition number of fiber cladding Nclad: %d¥n", Nclad );
                printf ( "Step size of propagation constants dβ: %f¥n", dbeta );
                printf ( "Partition number of propagation constants Nβ: %d ¥n", Nbeta );
                printf ( "Maximum allowable error for convergence solution vector eps1: %f¥n", eps1 );
                printf ( "Maximum allowable error of zero eigen value eps2: %f¥n", eps2 );
                printf ( "Maximum number of iterations in inverse power method jmax: %d¥n", jmax );
                printf ( "Correlation length Dc: %e m¥n", Dc );
                printf ( "Mean square of dielectric constant fluctuation <dε2>: %f¥n", deps2 );
                printf ( "Total fiber length zmax: %e m¥n", zmax );
                printf ( "Step size of fiber length dz: %e m¥n", dz );
                printf ( "Step size of time Tv: %e ps¥n", Tv );
                printf ( "Total spatial steps Nz: %d¥n", Nz );
                printf ( "File output step interval Nzout: %d¥n", Nzout );
                printf ( "Minimum evaluated frequency fminx: %e GHz¥n", fmin*1.0e3 );
                printf ( "Maximum evaluated frequency fmax: %e GHz¥n", fmax*1.0e3 );
                printf ( "Step size of evaluated frequency df: %e GHz¥n¥n", df*1.0e3 ); }
            printf ( "Wavelength: %f nm (%d/%d)¥n", lamda*1.0e9,y,Nl );
            if ( wo ==0 ) { printf ( "without mode coupling¥n"); }
            if ( wo ==1 ) { printf ( "with microscopic heterogeneities¥n"); }
            if ( wo ==2 ) { printf ( "with microbending¥n"); }
            fprintf ( fp2, "lamda=%f nm¥n", lamda*1.0e9 );
            fprintf ( fp4, "lamda=%f nm¥n", lamda*1.0e9 );
            if (  y == Nl/2 || fout == 1 ) {
                fprintf ( fp4, "m,l,p,tau,beta,ne,confinement,R_infinite,eig,");
                for ( j = 0; j <= N; j++ )  { fprintf ( fp4,"%f,", (double) j*dr*1.0e6 ); } fprintf ( fp4,"¥n" ); }
            fprintf ( fq, "lamda=%f nm¥n", lamda*1.0e9 );
            fprintf ( fr2, "lamda=%f nm¥n", lamda*1.0e9 );
            fprintf ( fr3, "lamda=%f nm¥n", lamda*1.0e9 );


            /** 2. モード解析 (FEM.cpp)  **/
            for ( m = 0 ; ; m++ ) {
                l = 1; beta = k*n1;
                /*********************************************************************************/
                for ( ; ; )  {
                    /* 係数行列計算および改訂コレスキー分解 */
                    w = aa*sqrt ( (beta*beta) - (k*k)*(n0*n0) );
```

```cpp
        S_matrix ( a, b, q, m, N, v, w, D );
        mcholesky ( a, b, ML, MD, m, N );
        /* 初期ベクトル R0 の付与 */
        R0 ( MD, R, m, N );
        /*  連立一次方程式 SR=(LDL)R=bR の反復評価 */
        for ( j = 0 ; ; j++ ) {
            for ( i = 0; i <= N; i++ )  { Rb[i] = R[i]; }
            mcholesky_sol ( ML, MD, R, m, N );
            R_norm ( R, N );
            /* 収束判定 */
            dd = 0, ds = 0;
            for ( i = 0; i <= N; i++ ) {
                dd = dd + (Rb[i] - R[i])*(Rb[i] - R[i]);
                ds = ds + (Rb[i] + R[i])*(Rb[i] + R[i]); }
            if ( dd < eps1 || ds < eps1 ) break;
            if ( j >= jmax ) goto next;
            // ① RとRbの成分差ddが0に漸近すれば収束（break）．
            // ② RとRbの成分和dsが0に漸近すれば中止（break）．
            // ③ 反復回数が上限値 jmax を超えたらβを変更して再計算（go to next）．
        }
        /* 固有値の計算 */
        eig = Eigen ( R, a, b, N, m );
        /* 固有値の妥当性評価 */
        // 「収束固有値 eig が前回値 bb と同値」であればβを変えて初めから再計算
        if ( eig == bb ) {
            dbeta = k*(n1 - n0) / (double) Nbeta;
            beta = beta - 1.0*dbeta;
            continue; }

        //  ①「0< 収束固有値 eig < eps2」であれば零固有値として採用
        if ( 0.0 < eig && eig < eps2 )  {
            /* 横方向電場成分Rの規格化 （パワーを1Wとする）*/

            sum = sumcore = sumclad = 0.0;
            for ( j = 0; j < N; j++ ) { sumcore = sumcore + R[j]*R[j]*(j*dr)*dr; }
            for ( j = 0; j < Nclad; j++ ) { sumclad = sumclad + R[N]*( bessk (m, w*(j*dr+A)) / bessk (m, w*A) )*R[N]*( bessk (m, w*(j*dr+A)) / bessk (m, w*A) )*(j*dr+A)*dr; }
            for ( j = 0; j <= N; j++ ) { R2[j] = R[j] * sqrt( (omega*Mu0) / (PI*beta*(sumcore + sumclad)) ); }
            tau = (1.0 / (C*1.0e-12))*(k / beta) * dbdk_bunshi( R, q2, D, w, m, N ) / dbdk_bunbo ( R, D, w, m, N ); // = (1/c)*(dβ/dk) [ps/m]
            /* 計算結果の出力 */
            Rinf = R[N]*( bessk (m, w*(Nclad*dr+A)) / bessk (m, w*A) );
            if ( y == Nl/2 || fout == 1 ) { fprintf ( fp4, "%d, %d, %d, %f, %f, %f, %f, %f, %f,",m,l,2*l+m-1,tau,beta,beta/k,sumcore/(sumcore+sumclad),Rinf,eig ); }
            modem[NLP] = m, model[NLP] = l, Mtau[NLP] = tau, Mbeta[NLP] = beta;
            modep[NLP] = 2*l + m - 1; if ( Ptotal < modep[NLP] ) { Ptotal = modep[NLP]; } // 主モード次数
            for ( j = 0; j <= N; j++ )  { if ( fout == 1 || y == Nl/2 ) { fprintf ( fp4,"%f,", R2[j] ); } Rlp[NLP][j] = R2[j]; }
            if ( y == Nl/2 || fout == 1 ) { fprintf( fp4,"¥n" ); }
            //printf ( "%d, %d, %f, %f, %f, %f, %f¥n", m, l, tau, beta, beta/k, eig, R[N] );

```

```
                        dbeta = k*(n1 - n0) / (double) Nbeta;
                        bb = eig;
                        l = l + 1;
                        NLP = NLP + 1; if ( m == 0 ) { NLP0 = NLP0 +1;}
                        count = 0; }

                    //  ②「0 < 収束固有値 eig」かつ「-1 < 前回値 bb < 0」であれば
                    if ( eig > 0.0 && bb < 0.0 && (fabs(bb) < 1.0) ) {
                        beta = beta + dbeta;
                        dbeta = dbeta / 2.0;
                        count = count + 1; }

                    // ③ その他
                    else { bb = eig; count = 0; }

                    if ( count > 1000 ) {
                        dbeta = k*(n1 - n0) / (double) Nbeta;
                        beta = beta - dbeta; }
next:
                    beta = beta - dbeta;
                    if ( beta < k*n0 )  break;
                }
                /*****************************************************************************************/
                if ( l == 1 ) { Ntotal = ( NLP0 )*2 + ( NLP - NLP0 )*4; break; } // mが最高次数に到達
            }

            free_drealvector ( GI, 0 ); free_drealvector ( q, 0 ); free_drealvector ( q2, 0 );
            free_drealvector ( R, 0 ); free_drealvector ( R2, 0 ); free_drealvector ( Rb, 0 );
            free_drealvector ( a, 0 ); free_drealvector ( b, 0 );
            free_drealvector ( ML, 0 ); free_drealvector ( MD, 0 );

            printf ( "Total numbers of LPml modes (WKB) Nwkb: %d¥n", Nwkb);
            printf ( "Total numbers of LPml modes NLP: %d¥n", NLP);
            printf ( "Total numbers of LP0l modes NLP0: %d¥n", NLP0);
            printf ( "Total numbers of all modes Ntotal: %d¥n", Ntotal);
            printf ( "Total numbers of all mode groups Ptotal: %d¥n", Ptotal);


            /** 3. LPモード特性の整理 **/
            /* 群遅延範囲 */
            // 各ループの単位長群遅延範囲 (taumin < tau < taumax)
            taumax = taumin = Mtau[0];
            for ( myu = 0; myu < NLP; myu++ ) {
                if ( taumax < Mtau[myu] ) { taumax = Mtau[myu]; }
                if ( taumin > Mtau[myu] ) { taumin = Mtau[myu]; }}
            // 各ループの最高次モード群単位長群遅延
            nctaumax = taumin;
            for ( myu = 0; myu < NLP; myu++ ) {
                if ( modep [myu] == Ptotal ) {  if ( nctaumax < Mtau[myu] ) { nctaumax = Mtau[myu]; }}}
            // 各ループの基準時間補正要素数 (nstd)
            if ( y == 0 ) { tauminstd = taumin; }
            nstd = (int) ( (taumin - tauminstd)*zmax / Tv );
            printf ("nstd:%d,nstdmin:%d¥n", nstd,nstdmin);
            fprintf ( fp2, "Minimum group delay per unit length,taumin,%e,ps/m¥n", taumin );
            fprintf ( fp2, "Maximum group delay per unit length,taumax,%e,ps/m¥n", taumax );
            // 各ループの最大郡遅延差 (Lmax)
```

```cpp
            Lmax = (int) ( (( taumax - taumin )*( (double)Nz*dz ) / Tv) + 0.5 );
            fprintf ( fp2, "Total time steps,Lmax,%d¥n", Lmax );
            if ( Lmax > ( 1.0e9 / (8.0*NLP) ) ) { printf ( "Memory over!¥n" ); exit ( EXIT_FAILURE ); }
            printf ( "Required memory for CPE analysis: %fGB¥n", (double)(Lmax*NLP*8) /1.0e9 );
            // プロファイルループの最大郡遅延差 (nmax)
            if ( Lmax > nmax ) { nmax = Lmax; }
            fprintf ( fp2, "Maximum time step,nmax,%d¥n", nmax );
            // インパルス応答格納配列数 (Pnum)
            if ( nstd == 0 ) { Pnum [y] = nmax + ( nstdmax - nstdmin ); }
            if ( nstd > 0 )  {
                if ( nstd < nstdmax ) { Pnum [y] = nmax + ( nstdmax - nstdmin ); }
                else { Pnum [y] = nmax + ( nstdmax - nstdmin ) + ( nstd - nstdmax ); }}
            if ( nstd < 0 )  {
                if ( nstd < nstdmin ) { Pnum [y] = nmax + ( nstdmax - nstdmin ) + ( nstdmin - nstd ); }
                else { Pnum [y] = nmax + ( nstdmax - nstdmin ); }}
            fprintf ( fp2, "Net total time steps,Pnum,%d¥n", Pnum [y] );

            /* 配列の記憶領域確保および初期化 */
            H = dmatrix ( 0, NLP-1, 0, NLP-1 ); init_realmatrix ( H, 0, NLP-1, 0, NLP-1 );
            Amin = dmatrix ( 0, NLP-1, 0, Lmax ); init_realmatrix ( Amin, 0, NLP-1, 0, Lmax );
            Aplu = dmatrix ( 0, NLP-1, 0, Lmax ); init_realmatrix ( Aplu, 0, NLP-1, 0, Lmax );
            alpha = drealvector  ( 0, NLP-1 ); init_realvector ( alpha, 0, NLP-1 );
            kim = dintvector  ( 0, NLP-1 ); init_intvector ( kim, 0, NLP-1 );
            Pm = drealvector ( 0, NLP-1 ); init_realvector ( Pm, 0, NLP-1 );
            Pg = drealvector ( 0, Ptotal-1 ); init_realvector ( Pg, 0, Ptotal-1 );
            if ( matdis == 0 ) { P = drealvector ( 0, Lmax ); init_realvector ( P, 0, Lmax ); }
            A00 = 0.0;


            /** 4. LPモード μ とLPモード ν 間の電力結合係数計算 **/
            /* H行列の算出 */
            /****************************************************************************************/
            if ( wo == 0 ) { for ( myu = 0; myu < NLP; myu++ ) { H[myu][myu] = 1.0; } }
            else {
            // ミクロ不均一構造
            if ( wo == 1) { dbmn = 0.0; E_over = 0.0; hmn =0.0;
#pragma omp parallel
            {
#pragma omp for
            for  ( myu = 0; myu < NLP; myu++ )  {
                for  ( nyu = 0; nyu < NLP; nyu++ )  {
                    dbmn = Mbeta [myu] - Mbeta [nyu];
                    for  ( n = 1; n <= N; n++ )  {

                        E_over = E_over + (2.0*PI)*( RIp[myu][n]*RIp[nyu][n] )*( RIp[myu][n]*RIp[nyu][n] )*((double)(n-1)*dr)*dr; }

                    if ( modep[myu] == Ptotal || modep[nyu] == Ptotal ) { hmn = 0.0; }   // 最高次モードは無視
                    else {
                        hmn = deps2*((omega*omega*(PI*sqrt(PI))*(Dc*Dc*Dc)) / 8.0)*exp(-(dbmn*dbmn)*(Dc*Dc) / 4.0)*E_over; }

                    if ( myu == nyu ) { H[myu][nyu] = 0.0; } else { H[myu][nyu] = hmn*dz; } // 仮入力

                    if ( y == NI/2 || fout == 1 ) {
                        if ( myu != nyu && myu > nyu ) {
//                          if ( myu != nyu ) {
                                fprintf ( fq, "%d,%d,%d,%d,%f,%d,%d,%d,%d,%f,%d,%f,%e,%e¥n", myu, modem [myu], model [myu], modep [myu], Mbeta [myu],
                                    nyu, modem [nyu], model [nyu], modep [nyu], Mbeta [nyu], abs(modep[myu]-modep[nyu]), fabs(dbmn), E_over, hmn ); }}
                    E_over = 0.0; }}
            }

            }

            // マイクロベンディング
            if ( wo == 2) { dbmn = 0.0; hmn =0.0;
            for ( myu = 0; myu < NLP; myu++ ) {
                for ( nyu = 0; nyu < NLP; nyu++ ) {
                    dbmn = Mbeta [myu] - Mbeta [nyu];

                    if ( modep[myu] == Ptotal || modep[nyu] == Ptotal ) { hmn = 0.0; }   // 最高次モードは無視
                    else if ( abs (modep [myu] - modep [nyu]) == 1 ) {
                        if ( modep[myu] > modep [nyu] ) { mm = modep[myu] ; } else { mm = modep[nyu]; }
                        hmn = (1.0/8.0)*(n1*k*A)*(n1*k*A)*pow( ((double)mm/(double)Ptotal), 4.0/(g+2.0) )
                                *sigma2*sqrt(PI)*Db*exp(-(dbmn*dbmn)*(Db*Db) / 4.0); }
                    else { hmn = 0.0; }

                    if ( myu == nyu ) { H[myu][nyu] = 0.0; } else { H[myu][nyu] = hmn*dz; } // 仮入力
                    if ( y == NI/2 || fout == 1 ) {
                        if ( myu != nyu && myu > nyu ) {
//                          if ( myu != nyu ) {
                                fprintf ( fq, "%d,%d,%d,%d,%f,%d,%d, %d,%d,%f,%d,%f,%e¥n", myu, modem [myu], model [myu], modep [myu], Mbeta [myu],
                                    nyu, modem [nyu], model [nyu], modep [nyu], Mbeta [nyu], abs(modep[myu]-modep[nyu]), fabs(dbmn), hmn ); }}
                         hmn = 0.0; }}
            }


            /* H行列の算出 */
            /*****************************************************************************************/
            hmn = 0.0; myu = 0; nyu = 0;
            // 対格要素
            for ( myu = 0; myu < NLP; myu++ ) { Hrowsum = 0.0;
            for ( nyu = 0; nyu < NLP; nyu++ ) {
                if ( modem[nyu] == 0 ) { Hrowsum = Hrowsum + H[myu][nyu]; } else { Hrowsum = Hrowsum + 2.0*H[myu][nyu]; }}
            H[myu][myu] = 1.0 - ( 2.0*alpha[myu]*dz + Hrowsum ); }
            // 非対格要素
            for ( myu = 0; myu < NLP; myu++ ) {
            for ( nyu = 0; nyu < NLP; nyu++ ) {
                if ( myu != nyu ) { if ( modem[myu] == 0 ) { H[myu][nyu] =
                      1.0*H[myu][nyu]; } else { H[myu][nyu] = 2.0*H[myu][nyu]; }}}}
            // 安定条件の確認
            Hmmmin = H[0][0];
            for ( m = 1; m < NLP; m++ ) { if ( Hmmmin > H[m][m] ) { Hmmmin = H[m][m]; } }
            if ( Hmmmin < 0) { printf ( "Too large Δz value! Change the value appropriately!¥n" ); exit ( EXIT_FAILURE ); }}
            /* H行列の出力*/
            if ( y == Nl/2 && fout == 1 ) {
                for ( myu = 0; myu < NLP; myu++ ) {
                for ( nyu = 0; nyu < NLP; nyu++ ) { fprintf ( fr, "%f,", H[myu][nyu] ); } fprintf ( fr, "¥n" ); }}



            /* 励振条件設定 (A+行列の算出) */
            // OFL condition
            if ( launch == 0 ) {
                for ( m = 0; m < NLP; m++ ) {
                    if ( matdis == 0 ) { if (modem[m] == 0) { Aplu[m][0] = 100.0; } else { Aplu[m][0] = 200.0; }} // 縮退数の考慮
                    if ( matdis == 1 ) {
                        if (modem[m] == 0) { Aplu[m][0] = 100.0*Spin[ (int)((lamda -lpmin)/dlp) ]; }
                        else { Aplu[m][0] = 200.0*Spin[ (int)((lamda -lpmin)/dlp) ]; }}}}


            // RML condition
            if ( launch == 1 ) {
#pragma omp parallel
                {
#pragma omp for
                for ( m = 0; m < NLP; m++ ) { Amev = Amod = 0.0;
                    for ( i = 0; i < Nxy; i++ ) { xx = ( r0 - ( (double)Nxy*dx / 2.0 ) ) + (double)i*dx;
                        for ( j = 0; j < Nxy; j++ ) { yy = - ( (double)Nxy*dy / 2.0 ) + (double)j*dy;
                        rr = sqrt ( xx*xx + yy*yy ); nr = (int) (rr / dr); // 切り捨て？
                        if ( nr == 0 ) { Rxy =  Rlp [m][0] + ( Rlp [m][1] - Rlp [m][0]) * (rr /dr); ;}
//          else { Rxy = Rlp [m][nr] + ( Rlp [m][nr+1] - Rlp [m][nr]) * ((rr - (double)nr*dr) /dr); } // interpolation
                        else if ( nr <= N ) { Rxy = Rlp [m][nr] + ( Rlp [m][nr+1] - Rlp [m][nr]) * ((rr - (double)nr*dr) /dr); } // core (interpolation)
                        else if ( nr > N ) { Rxy = Rlp [m][N]*( bessk (m, w*rr) / bessk (m, w*A) ); } // cladding
                        Emev = Rxy*cos( (double)(modem[m])*atan2(yy, xx) );
                        Emod = Rxy*sin( (double)(modem[m])*atan2(yy, xx) );
                        Ein = exp ( - ((xx - r0)*(xx - r0) + yy*yy) / (w0*w0) ); // input field
                        Amev = Amev + Emev*Ein*dx*dy; Amod = Amod + Emod*Ein*dx*dy; }}
                        Amev = Amev*Amev / (((2.0*omega*Mu0)/Mbeta[m])*(PI*w0*w0/2.0));
                        Amod =  Amod*Amod / (((2.0*omega*Mu0)/Mbeta[m])*(PI*w0*w0/2.0));
                    if ( matdis == 0 ) { Aplu[m][0] = 100.0* (Amod + Amev); }
                    if ( matdis == 1 ) { Aplu[m][0] = 100.0*( Amod + Amev )*Spin[ (int)((lamda -lpmin)/dlp) ]; }
                    //printf ("Aplu [%d][0] =%f¥n", modem[m],Aplu [m][0] );
                    }}}
            // Correction for the highest mode group ( elimination of power )
```

```cpp
        for ( m = 0; m < NLP; m++ ) { if ( modep[m] == Ptotal ) { Aplu[m][0] = 0.0; }}
        // Total power of input impulse
        for ( m = 0; m < NLP; m++ ) { A00 = A00 + Aplu[m][0]; } Aw00 = Aw00 + A00;

        fprintf ( fp2, "Amplitude of input impulse,A00,%e,¥n", A00 );
        fprintf ( fp2, "Cumulative amplitude of input impulse,Aw00,%e,¥n¥n", Aw00 );
        free_dmatrix ( Rlp, 0, 2*Nwkb, 0, N-1 );


        /* 出力ファイル */
        if ( y == Nl/2 || fout == 1 ) {
            // 時間波形 P
            fprintf ( fr2, "nstc=%d,", nstd );
            for ( n = 0; n <= Lmax; n++ ) { fprintf ( fr2, "%f,", (double)n*Tv );} fprintf ( fr2, "pct.¥n" );
            fprintf ( fr2, "%f,%f¥n", 0.0, A00 );
            // モードパワー分布 Pm
            fprintf ( fr3, "myu," ); for ( m = 0; m < NLP; m++ ) { fprintf ( fr3, "%d,", m ); } fprintf ( fr3, "¥n" );
            fprintf ( fr3, "LP," ); for ( m = 0; m < NLP; m++ ) { fprintf ( fr3, "LP%d_%d,", modem[m], model[m] ); } fprintf ( fr3, "¥n" );
            fprintf ( fr3, "m," ); for ( m = 0; m < NLP; m++ ) { fprintf ( fr3, "%d,", modem[m] ); } fprintf ( fr3, "¥n" );
            fprintf ( fr3, "l," ); for ( m = 0; m < NLP; m++ ) { fprintf ( fr3, "%d,", model[m] ); } fprintf ( fr3, "¥n" );
            fprintf ( fr3, "p," ); for ( m = 0; m < NLP; m++ ) { fprintf ( fr3, "%d,", modep[m] ); } fprintf ( fr3, "¥n" );
            fprintf ( fr3, "%f,", 0.0 );
            for ( m = 0; m < NLP; m++ ) { for ( n = 0; n <= Lmax; n++ ) { Pm[m] = Pm[m] + Aplu[m][n]; }
            fprintf ( fr3, "%f,", Pm[m] ); } fprintf ( fr3,"¥n");
            // モード群パワー分布 Pg
            fprintf ( fr9, "," ); for ( j = 1; j <= Ptotal; j++ ) { fprintf ( fr9, "%d,", j ); } fprintf ( fr9, "¥n" ); // モード群番号
            fprintf ( fr9, "," ); for ( i = 1; i <= Ptotal; i++ ) { count =0;
            for ( myu = 0; myu < NLP; myu++ ) { if ( modep [myu] == i ) { count = count +1; }} // 縮退数（モード数）
            fprintf ( fr9, "%d,", count ); pdeg[i-1] =count; } fprintf ( fr9,"¥n");
            fprintf ( fr9, "%f,", 0.0 );
            for ( m = 0; m < NLP; m++ ) { Pg[modep[m]-1] = Pg[modep[m]-1] + Pm[m]; } // モード群パワー
            for ( j = 0; j < Ptotal; j++ ) { fprintf ( fr9, "%f,", Pg[j] / (double)pdeg[j] ); Pg[j] = 0.0; }  fprintf ( fr9,"¥n");
        }


        printf("start!¥n");
        /** 5. 光波伝搬解析 **/
        /****************************************************************************************/
        for ( i = 1; i <= Nz; i++ ) // z(i-1) ~ zi
        {
            /* 相対遅延ステップ数の最大値算出 */
            Li =0;
            Li = (int) ( (( taumax - taumin )*( (double)i*dz ) / Tv) + 0.5 );

```

```
                /* 相対遅延ステップ数のモード分布算出 */
                for ( m = 0; m < NLP; m++ ) {
                    kim[m] = (int) ( (( Mtau[m] - taumin )*( (double)i*dz ) / Tv ) + 0.5 )
                                  - (int) ( (( Mtau[m] - taumin )*( (double)(i-1)*dz ) / Tv ) + 0.5 ); }

#pragma omp parallel
{
#pragma omp for
                /* タイムシフト演算 */
                for ( m = 0; m < NLP; m++ ) {
                    for ( n = 0; n < kim[m]; n++ ) { Amin[m][n] = 0.0; }
                    for ( n = kim[m]; n <= Li; n++ ) { Amin[m][n] = Aplu[m][n - kim[m]]; }}
#pragma omp for
                /* カップリング演算 */
                for ( m = 0; m < NLP; m++ ) {
                    for ( n = 0; n <= Li; n++ ) { me=0.0;
                    for ( l = 0; l < NLP; l++ ) { me = me + H[m][l]*Amin[l][n]; }
                    Aplu[m][n] = me; }}
}

                // ① 波長分散を考慮する場合
                /* 時間波形の重ね合わせ */
                if ( matdis == 1 && i == Nz ) {
                    // 各波長成分のインパルス応答（基準時間は非考慮）
                    if ( y == Nl/2 || fout == 1) { fprintf ( fr2, "%f,", (double)i*dz ); }
                    for ( n = 0; n <= Li; n++ ) { ap = 0.0;
                        for ( m = 0; m < NLP; m++ ) { ap = ap + Aplu[m][n]; }
                        if ( y == Nl/2 || fout == 1) { fprintf ( fr2, "%f,", ap ); }}
                    if ( y == Nl/2 || fout == 1) { fprintf ( fr2,"¥n" ); }
                    // 最小群遅延の波長依存性を考慮した足し合わせ（基準時間を考慮）
                    if ( y == 0 || nstd == 0 ) { for ( n = 0; n <= Li; n++ ) { for ( m = 0; m < NLP; m++ ) { P[n] = P[n] + Aplu[m][n]; }}; }
                    if ( y != 0 && nstd > 0 ) { for ( n = 0; n <= Li; n++ ) { for ( m = 0; m < NLP; m++ ) { P[n+nstd] = P[n+nstd] + Aplu[m][n]; }}; }
                    if ( y != 0 && nstd < 0 ) {
                        if ( nstd < nstdmin ) {
                            for ( n = 0; n <= Pnum[y-1] ; n++ ) { P[(Pnum[y-1]-n)+(nstdmin-nstd)] = P[(Pnum[y-1]-n)]; }
                            for ( n = 0; n < nstdmin-nstd; n++ ) { P[n] = 0.0; }
                            for ( n = 0; n <= Li; n++ ) { for ( m = 0; m < NLP; m++ ) { P[n] = P[n] + Aplu[m][n]; }}}
                        else {
                            for ( n = 0; n <= Li; n++ ) { for ( m = 0; m < NLP; m++ ) {
                                P[n+(nstd-nstdmin)] = P[n+(nstd-nstdmin)] + Aplu[m][n]; }}; }};
                    // for ( n = 0; n <= Pnum[y]; n++ ) { fprintf ( fs3,"%f,", P[n] ); } fprintf ( fs3,"¥n" );
                }


                // ② 波長分散を考慮しない場合
                /* 計算結果の出力 */
                if ( matdis == 0 && (i%Nzout == 0 || i == Nz) ) {
                    /* モードパワー分布 Pm */
                    if ( fout == 1 || y == Nl/2 ) { fprintf ( fr3, "%f,"
```

```cpp
(double)i*dz ); }
for ( m = 0; m < NLP; m++ ) {
    Pm[m] = 0.0;
    for ( n = 0; n <= Li; n++ ) { Pm[m] = Pm[m] + Aplu[m][n]; }
    if ( fout == 1 || y == Nl/2 ) { fprintf ( fr3, "%f,", Pm[m] ); }
    Pg[modep[m]-1] = Pg[modep[m]-1] + Pm[m]; }
if ( fout == 1 || y == Nl/2 ) { fprintf ( fr3,"¥n"); }
/* モード群パワー分布 Pg */
if ( fout == 1 || y == Nl/2 ) { fprintf ( fr9, "%f,", (double)i*dz );
for ( j = 0; j < Ptotal; j++ ) { fprintf ( fr9, "%f,", Pg[j] / (double)pdeg[j] ); Pg[j] =0.0;}  fprintf ( fr9,"¥n"); }
/* インパルス応答 P */
if ( fout == 1 || y == Nl/2) { fprintf ( fr2, "%f,", (double)i*dz ); }
for ( n = 0; n <= Li; n++ ) {
    P[n] = 0.0;
    for ( m = 0; m < NLP; m++ ) { P[n] = P[n] + Aplu[m][n]; }

    if ( fout == 1 || y == Nl/2 ) { fprintf ( fr2, "%f,", P[n] ); } }
if ( fout == 1 || y == Nl/2 ) { fprintf ( fr2,"¥n" ); }

/* 出力波形 */
Pout = drealvector ( 0, Ti+Li ); init_realvector ( Pout, 0, Ti+Li );
Pout = convolution ( Ti, Pin, Li, P );
if ( fout == 1 || y == Nl/2) {
    for ( n = 0; n < Ti+Li; n++ ) { fprintf ( fr6, "%f,", Pout[n] );} fprintf ( fr6,"¥n" );}
/* 周波数応答 M */
if ( fout == 1 || y == Nl/2 ) { fprintf ( fr4, "%f,", (double)i*dz ); }
for ( j = 0; j <= Nf; j++ ) {
    Hw = ReHw = ImHw = 0.0;
    for ( n = 0; n <= Li; n++ ) {
        ReHw = ReHw + P[n]*cos ( (2.0*PI*(fmin+(double)j*df))*(double)n*Tv );
        ImHw = ImHw - P[n]*sin ( (2.0*PI*(fmin+(double)j*df))*(double)n*Tv ); }
    M[j] = sqrt ( ReHw*ReHw + ImHw*ImHw ) / A00; if ( y == Nl/2 || fout == 1 ) { fprintf ( fr4, "%f,", -10.0*log10(1.0 / M[j]) ); }}
/* -3dB帯域幅 bw */
bw = tav = Ptot = rms = 0.0;
for ( j = 0; j <= Nf-1; j++ )  { if ( M[j] >0.5 && M[j+1] < 0.5 ) { bw = (fmin*1.0e3) + (df *1.0e3)*(j + (M[j] - 0.5) / (M[j] - M[j+1])); break; } }
if ( fout == 1 || y == Nl/2 ) { fprintf ( fr4, "%f,", bw );}
/* インパルス応答RMS幅 rms */
for ( n = 0; n <= Li; n++ ) { tav = tav + ( taumin*(double)(i-1)*dz + (double)n*Tv)*P[n]*Tv; Ptot = Ptot + P[n]*Tv; } tav = tav / Ptot;
for ( n = 0; n <= Li; n++ ) { rms = rms + ( taumin*(double)(i-1)*dz + (double)n*Tv)*( taumin*(double)(i-1)*dz + (double)n*Tv)*( P[n] / Ptot )*Tv; }
rms = sqrt ( rms - (tav*tav) );

if ( fout == 1 || y == Nl/2 ) { fprintf ( fr4, "%f,%f",
                rms, tav - ( taumin*(double)(i-1)*dz) ); fprintf ( fr4, "¥n" ); }

                 fprintf ( fs, "%f,%d,%d,%f,%f,%f,%f,%f,%f,%f¥n", g, NLP, NLP0, taumin*(double)(i-1)*dz*1.0e-3, (double)i*dz, rms, bw, tav, Ptot, Aw00 );
                 printf ( "length:%f m ", (double)i*dz ); printf ( "-3db bandwidth:%f GHz ", bw ); printf ( "pulse broadening:%f ps¥n", rms );
              }

          }
          /*****************************************************************************************/

           if ( nstd < nstdmin ) { nstdmin = nstd; }
           if ( nstd > nstdmax ) { nstdmax = nstd; }

           free_dmatrix ( H, 0, NLP-1, 0, NLP-1 );
           free_dmatrix ( Amin, 0, NLP-1, 0, Lmax );
           free_dmatrix ( Aplu, 0, NLP-1, 0, Lmax );
           free_drealvector ( alpha, 0 );
           free_dintvector ( kim, 0 );
           free_drealvector ( Pm, 0 );
           free_drealvector ( Pg, 0 );
           free_dintvector ( modem, 0 );
           free_dintvector ( model, 0 );
           free_dintvector ( modep, 0 );
           free_dintvector ( pdeg, 0 );
           free_drealvector ( Mbeta, 0 );
           free_drealvector ( Mtau, 0 );
           printf ( "A00=%f¥n", A00 );
           printf ( "End¥n¥n");



} // 波長ループ終了
printf ( "Aw00=%f¥n", Aw00 );



/* 計算結果の出力 */
if ( matdis == 1 ) {
    /* インパルス応答波形（光源スペクトル考慮） Pw */
    for ( n = 0; n <= Pnum[NI]; n++ ) { fprintf ( fs3,"%f,", P[n] ); } fprintf ( fs3,"¥n" );
    /*  出力波形 */
    Pout = drealvector ( 0, Ti+Pnum[NI] ); init_realvector ( Pout, 0, Ti+Pnum[NI] );
    Pout = convolution ( Ti, Pin, Pnum[NI], P );
    if ( fout == 1 || y == NI/2) {
        for ( n = 0; n < Ti+Pnum[NI]; n++ ) { fprintf ( fs4, "%f,", Pout[n] );} fprintf ( fs4,"¥n" );}
    /* 周波数応答 M */
    fprintf ( fr4, "%f,", (double)Nz*dz );
    for ( j = 0; j <= Nf; j++ ) {
        ReHw = ImHw = 0.0;
        for ( n = 0; n <= Pnum[NI]; n++ ) {
            ReHw = ReHw + P[n]*cos ( (2.0*PI*(fmin+(double)j*df))*(double)n*Tv );
            ImHw = ImHw - P[n]*sin ( (2.0*PI*(fmin+(double)j*df))*(double)n*Tv ); }
        M[j] = sqrt ( ReHw*ReHw + ImHw*ImHw ) / Aw00; fprintf ( fr4, "%f,", -10.0*log10(1.0 / M[j]) ); }

    /* -3dB帯域幅 bw */
    bw = tav = Ptot = rms = 0.0;
    for ( j = 0; j <= Nf-1; j++ )  { if ( M[j] >0.5 && M[j+1] < 0.5 ) { bw = (fmin*1.0e3) + (df *1.0e3)*(j + (M[j] - 0.5) / (M[j] - M[j+1])); break; }}
    fprintf ( fr4, "%f,", bw );

    /* インパルス応答RMS幅 rms */
    for ( n = 0; n <= Pnum[Nl]; n++ ) { tav = tav + ( taumin*(double)(i-1)*dz + (double)n*Tv)*P[n]*Tv; Ptot = Ptot + P[n]*Tv; } tav = tav / Ptot;

    for ( n = 0; n <= Pnum[Nl]; n++ ) { rms = rms + ( taumin*(double)(i-1)*dz + (double)n*Tv)*( taumin*(double)(i-1)*dz + (double)n*Tv )*( P[n] / Ptot )*Tv; }
    rms = sqrt ( rms - (tav*tav) );
    fprintf ( fr4, "%f,", rms); fprintf ( fr4, "¥n" );

    fprintf ( fs, "%f,%f,%f,%f,%f,%f,%f,", g, (double)Nz*dz, rms, bw, tav, Ptot, Aw00 );
    printf ( "length:%f m¥n", (double)Nz*dz ); printf ( "-3db bandwidth:%f GHz¥n", bw ); printf ( "rms width:%f ps¥n", rms ); }



/* スペックルコントラストの計算 */
if ( scc == 1 ) {
    /* 光源スペクトル自己相関関数 Cp */
    Cp = drealvector ( 0, Nf ); init_realvector ( Cp, 0, Nf );
    for ( j = 0; j <= Nf; j++ ) {
        for ( i = 0; i <= Nfp; i++ ) {
            int jj = (int) ( (double)j*df / dfp );
            Cp[j] = Cp[j] + Spin[Nfp-i]*Spin[ Nfp-i+jj ]; }
        Cpsum = Cpsum + df*Cp[j]; }
        // ガウス型スペクトル形状関数
//      Cp[j] = Cp[j] + exp( -pow( (fpmin + dfp*double(i) - fp0) / fpgw, 2.0 ))
//                              *exp( -pow( (fpmin + dfp*double(i) - df*double(j) - fp0 ) / fpgw, 2.0 ))*dfp; }

    /* スペックルコントラスト spct */
    fprintf ( fr4, ","  );
    for ( j = 0; j <= Nf; j++ ) {
        Cp[j] = Cp[j] / (2.0*Cpsum); fprintf ( fr4, "%f,", Cp[j] ); // Cpが偶関数であることを考慮
        spct = spct + Cp[j]*M[j]*M[j]*df; } spct = sqrt ( 2.0*spct );
        fprintf ( fs, "%f¥n", spct );   printf ( "speckle contrast:%f¥n", spct );
        free_drealvector (Cp, 0); }


free_dintvector ( Pnum, 0 );
free_drealvector ( P, 0 );
free_drealvector ( Pin, 0 );
free_drealvector ( Pout, 0 );
free_drealvector ( M, 0 );


fclose ( fp2 ); // BW_setting.csv
fclose ( fp3 ); // BW_profile.csv
fclose ( fp4 ); // FEM_result.csv
fclose (fq);    // Hmn_result.csv
fclose ( fr );  // CPE_Hmatrix.csv
fclose ( fr2 ); // CPE_Impulse-responce.csv
fclose ( fr3 ); // CPE_Mode-power-distribution.csv
fclose ( fr4 ); // CPE_Frequency-response.csv
fclose ( fr6 ); // CPE_Output-pulse-waveform.csv
fclose (fs);    // BW_result.csv
fclose (fs2);   // BW_Source-spectrum.csv
fclose (fs3);   // BW_Impulse-responce.csv
fclose (fs4);   // BW_Output-pulse-waveform.csv

system("pause");
return 0;
}



/*A.1. 第2種変形ベッセル関数 bessk (n, x) */
// 第1種変形Bessel関数 (n=0) I0(x)
double bessi0 (double x)
{
    double ax, ans;
    double y;
    // Polynomial fit
    if ( ( ax = fabs(x) ) < 3.75 ) {
        y = x / 3.75;
        y*= y;
        ans = 1.0 + y*(3.5156229 + y*(3.0899424 + y*(1.2067492
            + y*(0.2659732 + y*(0.360768e-1+y*0.45813e-2)))));}
    else {
        y = 3.75 / ax;
        ans=(exp(ax) / sqrt(ax))*(0.39894228 + y*(0.1328592e-1
            +y*(0.225319e-2 + y*(-0.157565e-2 + y*(0.916281e-2
            +y*(-0.2057706e-1 + y*(0.2635537e-1 + y*(-0.1647633e-1
            +y*0.392377e-2))))))));}
    return ans;
}
// 第2種変形Bessel関数 (n=0)  K0(x)
double bessk0 (double x)
{
    double bessi0 (double x);
    double y, ans;
    //polynomial fit
    if ( x <= 2 ) {
        y = x*x / 4.0;
        ans = (-log(x/2.0)*bessi0(x)) + (-0.57721566 + y*(0.42278420
            + y*(0.23069756 + y*(0.3488590e-1 + y*(0.262698e-2
            + y*(0.10750e-3 + y*0.74e-5))))));
    }
    else {
        y=2.0/x;
        ans = (exp(-x)/sqrt(x))*(1.25331414 + y*(-0.7832358e-1
            + y*(0.2189568e-1 + y*(-0.1062446e-1 + y*(0.587872e-2
            + y*(-0.251540e-2 + y*0.53208e-3))))));
    }
    return ans;
}
// 第1種変形Bessel関数 (n=1)  I1(x)
double bessi1 (double x)
{
    double ax, ans;
    double y;
    if ( ( ax = fabs(x) ) < 3.75 ) {
        y = x / 3.75;
        y*=y;
        ans = ax*(0.5 + y*(0.87890594 + y*(0.51498869 + y*(0.15084934
            + y*(0.2658733e-1 + y*(0.301532e-2 + y*0.32411e-3))))));}
    else {
        y = 3.75 / ax;
        ans = 0.2282967e-1 + y*(-0.2895312e-1 + y*(0.1787654e-1
            - y*0.420059e-2));
        ans = 0.39894228 + y*(-0.3988024e-1 + y*(-0.362018e-2
            + y*(0.163801e-2 + y*(-0.1031555e-1 + y*ans))));
        ans*=(exp(ax) / sqrt(ax));
    }
    return x < 0.0 ? - ans : ans;
}
// 第2種変形Bessel関数 (n=1)  K1 (x)
double bessk1 (double x) {
    double bessi1 (double x);
    double y, ans;
    if ( x <= 2.0 ) {
        y = x*x/4.0;
        ans = (log(x/2.0)*bessi1(x)) + (1.0/x)*(1.0 + y*(0.15443144
            + y*(-0.67278579 + y*(-0.18156897 + y*(-0.1919402e-1
            + y*(-0.110404e-2 + y*(-0.4686e-4)))))));}
    else{
        y = 2.0/x;
        ans = (exp(-x)/sqrt(x))*(1.25331414 + y*(0.23498619
            + y*(-0.3655620e-1 + y*(0.1504268e-1 + y*(-0.780353e-2
            + y*(0.325614e-2 + y*(-0.68245e-3)))))));
    }
    return ans;
}
// 第2種変形Bessel関数 Kn(x)
double bessk (int n,double x)
{
    double bessnorm = 1.0e7 ;
    double bessk0 (double x);
    double bessk1 (double x);
    int j;
    double bk,bkm,bkp,tox;
    if ( n == 0 ) return bessk0 (x) / bessnorm;
    if ( n == 1 ) return bessk1 (x) / bessnorm;
    if ( n >= 2 ) {
        tox = 2.0/x;
        bkm = bessk0(x) / bessnorm;
        bk = bessk1(x) / bessnorm;
        for ( j=1; j<n; j++ ) {
            bkp = bkm + j*tox*bk;
            bkm = bk;
            bk = bkp;
        }
        return bk;
    }
    else return 0;
}

/*A.2. 屈折率波長微分関数dndl_var (lamda, n_lamda, mater) */
double dndl ( double lamda, double n_lamda, int mater )
//{ return ( ( 0.01925e-4*lamda - 16.31619e-4 )*n_lamda + ( -0.02743e-4*lamda + 23.16674e-4 ) )*1.0e9; }
{ return ( ( 0.02173e-4*lamda - 18.79107e-4 )*n_lamda + ( -0.03109e-4*lamda + 26.85035e-4 ) )*1.0e9; } // 640 ~ 690 nm

/*A.3. 屈折率濃度微分関数dndl_var (lamda, mater) */
double dndc ( double lamda, int mater )
{ return 2.03716e-9*lamda*lamda - 3.27125e-6*lamda + 2.98314e-3; } // 589 ~ 690 nm

/*A.3. コア中心屈折率 ncore (lamda, mater) */
double ncore ( double lamda, int mater ) {
    double sell; sell = 1.0;
    /* DPS-doped PMMA ( 7.8 wt.%, 1.506@589nm ) */
    sell = sqrt ( 1.0 + ( 0.41241 / (1.0 - ( 22500 / (lamda*lamda) )) )
        + ( 0.81215 / (1.0 - ( 6400 / (lamda*lamda) )) )
        + ( 0.01117 / (1.0 - ( 11560000 / (lamda*lamda) )) ) );
    /* DPS-doped PMMA ( 9.0 wt.%, 1.506@655nm ) */
//  sell = sqrt ( 1.0 + ( 0.61249 / (1.0 - ( 8467.04111 / (lamda*lamda) )) )
//      + ( 0.61872 / (1.0 - ( 16525.45233 / (lamda*lamda) )) )
//      + ( 0.08889 / (1.0 - ( 129601870.30589 / (lamda*lamda) )) ) );
```

```cpp
    return sell; }

/*A.4. クラッド屈折率 nclad (lamda, mater) */
double nclad ( double lamda, int mater ) {
    double sell; sell =1.0;
    /* PMMA ( 1.492@589nm ) */
    sell = sqrt ( 1.0 + ( 0.496284 / (1.0 - ( 5154.872 / (lamda*lamda) )) )
        + ( 0.6964977 / (1.0 - ( 13802.53 / (lamda*lamda) )) )
        + ( 0.3223 / (1.0 - ( 85527690 / (lamda*lamda) )) ) );
    /* DPS-doped PMMA ( 1.079427062 wt. % ) */
//  sell = sqrt ( 1.0 + ( 0.5954 / (1.0 - ( 6200.94518 / (lamda*lamda) )) )
//      + ( 0.602 / (1.0 - ( 14730.91236 / (lamda*lamda) )) )
//      + ( 0.29126 / (1.0 - ( 85685787.87303 / (lamda*lamda) )) ) );
    return sell; }

/*A.5. 係数行列要素計算関数 S_matrix (a, b, q, m, n, v, w, D)*/
// b[0]=S00, b[1]=S11, ...................... b[j]=Sjj ......  b[n]=Snn
// a[0]=___, a[1]=S01, a[2]=S12, ... a[j]=Sj-1,j ... a[n]=Sn-1,n
// a[0]=___, a[1]=S10, a[2]=S21, ... a[j]=Sj,j-1 ... a[n]=Sn,n-1
void S_matrix ( double *a, double *b, double *q, int m, int n, double v, double w, double D )
{
    int j;
    if ( m == 0 ) {
            b[0] = - (1.0/2.0) + (3.0*q[0]+2.0*q[1])*(v*v/60.0)*((D*D)/(n*n)) - (1.0/12.0)*(w*w)*((D*D)/(n*n));
            b[n] = - ((2.0*n-1.0)/2.0) + ((5.0*n-2.0)*q[n-1]+3.0*(5.0*n-1.0)*q[n])*(v*v/60.0)*((D*D)/(n*n)) - ((4.0*n-1.0)/12.0)*(w*w)*((D*D)/(n*n))
                      - w*D*bessk(1,w*D)/bessk(0,w*D);}
    else {
            b[0] = 0.0;
            b[n] = - (1.0-m*m)*((2.0*n-1.0)/2.0) + ((5.0*n-2.0)*q[n-1]+3.0*(5.0*n-1.0)*q[n])*(v*v/60.0)*((D*D)/(n*n)) - ((4.0*n-1.0)/12.0)*(w*w)*((D*D)/(n*n))
                      - (m*m)*((n-1.0)*(n-1.0))*log((double)n/((double)n-1.0)) - m*m - w*D*bessk(m-1,w*D)/bessk(m,w*D) - m ; }

            b[1] = - 2.0*(1.0-m*m) + (3.0*q[0]+30*q[1]+7.0*q[2])*(v*v/60.0)*((D*D)/(n*n)) - (2.0/3.0)*(w*w)*((D*D)/(n*n)) - (m*m)*4.0*log(2.0);
        for ( j = 2; j < n; j++ )
        {   b[j] = - 2.0*j*(1.0-m*m) + ((5.0*j-2.0)*q[j-1]+30.0*j*q[j]+(5.0*j+2.0)*q[j+1])*(v*v/60.0)*((D*D)/(n*n)) - (2.0/3.0)*j*(w*w)*((D*D)/(n*n))
                     - (m*m)*( ((j-1.0)*(j-1.0))*log((double)j/((double)j-1.0)) + ((j+1.0)*(j+1.0))*log( ((double)j+1.0) / (double)j) ); }

            a[0] = 0.0;//未使用要素につき0を格納
            a[1] = (1.0/2.0) + (2*q[0]+3*q[1])*(v*v/60.0)*((D*D)/(n*n)) - (1.0/12.0)*(w*w)*((D*D)/(n*n)) - (m*m)/2;
        for ( j = 2; j <= n; j++ )
        {   a[j] = ((2.0*j-1.0)/2.0)*(1.0-m*m) + ((5.0*j-3.0)*q[j-1] + (5.0*j-2.0)*q[j])*(v*v/60.0)*((D*D)/(n*n)) - ((2.0*j-1.0)/12.0)*(w*w)*((D*D)/(n*n))
        + (m*m)*(j-1.0)*j*log((double)j / ((double)j-1.0)); }
}

/*A.4. 改訂コレスキー分解  mcholesky ( a, b, ML, MD, m, n )*/
void mcholesky ( double *a, double *b, double *ML, double *MD, int m, int n )
{
    int i;
    if ( m == 0 ) {
        ML[0] = 0.0;//未使用要素につき0を格納
        MD[0] = b[0];
        for ( i = 1; i <= n; i++ )
        {   MD[i] = b[i] - a[i]*a[i] / MD[i-1];
            ML[i] = a[i] / MD[i-1]; }
    }
    else {
        ML[0] = 0.0;//未使用要素につき0を格納
        ML[1] = 0.0;//発散するから別扱い. 0で良いのか?
        MD[0] = 0.0;
        MD[1] = b[1];
        for ( i = 2; i <= n; i++ )
        {   MD[i] = b[i] - a[i]*a[i] / MD[i-1];
            ML[i] = a[i] / MD[i-1]; }
    }
}

/*A.5. 改訂コレスキー分解法により方程式を解く   mcholesky_sol ( a, b, ML, MD, 
   m, n )*/
void mcholesky_sol ( double *ML, double *MD, double *R, int m, int n )
{
    int i;
    //「Ly=R0」を解く
    if ( m==0 ) {
        for ( i=1; i <= n; i++ ) {
            R[i] = R[i] - R[i-1]*ML[i]; }
    //「(D(LT))R1=y」を「(LT)R1=(D-1)y=y'」に変える
        for ( i=1; i <= n; i++ ) {
            R[i] = R[i] / MD[i]; }
    //「 (LT)R1=y'」を解く
        for ( i=n-1; i >= 0; i-- ) {
            R[i] = R[i] - ML[i+1]*R[i+1]; }
    }
    else{
        for ( i=2; i <= n; i++ ) {
            R[i] = R[i] - R[i-1]*ML[i]; }
    //「(D(LT))R1=y」を「(LT)R1=(D-1)y=y'」に変える
        for ( i=2; i <= n; i++ ) {
            R[i] = R[i] / MD[i]; }
    //「 (LT)R1=y'」を解く
        for ( i=n-1; i >= 1; i-- ) {
            R[i] = R[i] - ML[i+1]*R[i+1]; }
    }

}

/*A.6. 逆べき乗法の初期ベクトル計算 R0 ( MD, R, m, n )*/
void R0 ( double *MD, double *R, int m, int n )
/* 対角行列Dの成分が最大となる要素だけ1であるようなベクトルを選定*/
{
    int i, j = 1;
    if ( m == 0 ) {
        for ( i = 0; i <= n-1; i++ ) {
            if ( fabs(MD[i+1]) < fabs(MD[j]) ) { j = i + 1; } }
        }
    else {
        for ( i = 1; i <= n-1; i++ ) {
            if ( fabs(MD[i+1]) < fabs(MD[j]) ) { j = i + 1; } }
    }
    //R0の初期値の代入
    for ( i = 0; i <= n; i++ ) {
            if ( i == j ) { R[i] = 1.0; }
            else { R[i] = 0.0; }
        }
}

/*A.7. 逆べき乗法の解ベクトル規格化 R_norm ( R, n )*/
void R_norm ( double *R, int n )
{
    int i;
```

```cpp
    double s = 0;
    //  行列要素の２乗和
    for ( i = 0; i <= n; i++ )  {   s = s + R[i]*R[i];  }
    if ( s  != 0 )
    {   for ( i = 0; i <= n; i++ )  {   R[i] = R[i] / sqrt(s);  }   }
}

/*A.8. 固有値計算 (Rayleigh quotient) Eigen ( R, a, b, m, n )*/
double Eigen ( double *R, double *a, double *b, int n, int m )
{
    // Rベクトルを規格化しているため内積は1
    int i;
    double s=0;
    if ( m == 0 ) {
        s = ( R[0]*b[0] + R[1]*a[1] )*R[0];
        for ( i = 1; i < n; i++ ) {
            s += ( R[i-1]*a[i] + R[i]*b[i] + R[i+1]*a[i+1] )*R[i];  }
        s += ( R[n-1]*a[n] + R[n]*b[n] )*R[n];
        return s;
    }
    else {
        s = ( R[1]*b[1] + R[2]*a[2] )*R[1];
        for ( i = 2; i < n; i++ ) {
            s += ( R[i-1]*a[i] + R[i]*b[i] + R[i+1]*a[i+1] )*R[i];  }
        s += ( R[n-1]*a[n] + R[n]*b[n] )*R[n];
        return s;
    }
/*
    s = b[0]*R[0]*R[0];
    for ( i = 1; i <= n; i++ ) {
        s = s + (  b[i]*R[i]*R[i] + 2.0*a[i]*R[i-1]*R[i] ); }
*/
}

/*A.9. 群遅延計算用関数 dbdk_bunbo ( R, D, w, m, n )*/
/* 入力パラメータ (横方向電場分布, コア径, 伝搬定数, 要素分割数) に対するm次モード群遅延計算式の分母分子を計算する */

// 横方向電場成分 R[0]~R[n], 規格化伝搬定数 w, 規格化コア径 D, 方位角モード次数 m, 分割数 n
double dbdk_bunbo ( double *R, double D, double w, int m, int n )
{
    int i;
    double s=0;
    for ( i = 0; i <= n-1; i++ )
    { s = s + (1.0/12.0) * ((D*D)/(n*n)) * ( (double)(4*i+1)*R[i]*R[i] + 2.0*(double)(2*i+1)*R[i]*R[i+1] + (double)(4*i+3)*R[i+1]*R[i+1] ); }
    if ( m == 0) {
        return s + (( bessk(1,w*D)*bessk(1,w*D) / (bessk(0,w*D)*bessk(0,w*D))) - 1.0) * ((D*D)*(R[n]*R[n]) / 2.0); }
    else {
        return s + (( bessk(m-1,w*D)*bessk(m+1,w*D) / (bessk(m,w*D)*bessk(m,w*D))) - 1.0) * ((D*D)*(R[n]*R[n]) / 2.0); }
}

/*A.10. 群遅延計算用関数    dbdk_bunshi ( R,q2, D, w, m, n )*/
/* 入力パラメータ (横方向電場分布, コア径, 伝搬定数, 要素分割数) に対するm次モード群遅延計算式の分母分子を計算する */

// 横方向電場成分 R[0]~R[n], 規格化伝搬定数 w, 規格化コア径 D, 方位角モード次数 m, 分割数 n
// 屈折率分散パラメータ q2[0]~q2[n] ( = n*(d(kn)/dk) )
double dbdk_bunshi ( double *R, double *q2, double D, double w, int m, int n)
{
    int i;
    double s=0;
    for ( i=0; i<=n-1; i++ )
    { s = s + (1.0/12.0) * ((D*D)/(n*n))* ( ((double)(3*i)+3.0/5.0)*q2[i]*R[i]*R[i] + ((double)i+2.0/5.0)*(2.0*q2[i]*R[i+1]+q2[i+1]*R[i])*R[i] + ((double)i+3.0/5.0)*(q2[i]*R[i+1]+2.0*q2[i+1]*R[i])*R[i+1] + ((double)(3*i)+12.0/5.0)*q2[i+1]*R[i+1]*R[i+1]); }
    if ( m == 0) {
        return s + q2[n] * (( bessk(1,w*D)*bessk(1,w*D) / (bessk(0,w*D)*bessk(0,w*D))) - 1.0) * ((D*D)*(R[n]*R[n]) / 2.0); }
    else {
        return s + q2[n] * (( bessk(m-1,w*D)*bessk(m+1,w*D) / (bessk(m,w*D)*bessk(m,w*D))) - 1.0) * ((D*D)*(R[n]*R[n]) / 2.0); }
}

/* A.13. 整数ベクトル領域確保用関数 dvector (i, j) */
int *dintvector ( int i, int j ) {
    int *a;
    if ( ( a = (int *) malloc ( (j -i+1)*sizeof (int) ) ) == NULL )
        { printf ("Memory cannot be allocated !¥n"); exit (1); }
    return (a-i); }

/* A.14. 整数ベクトル領域解放用関数 free_dvector (a, i) */
void free_dintvector ( int *a,  int i ) {
    free ( (void*) (a+i) ); }

/* A.15. 実数ベクトル領域確保用関数 dvector (i, j) */
double *drealvector ( int i, int j ) {
    double *a;
    if ( ( a = (double *) malloc ( (j -i+1)*sizeof (double) ) ) == NULL )
        { printf ("Memory cannot be allocated !¥n"); exit (1); }
    return (a-i); }

/* A.16. 実数ベクトル領域解放用関数 free_dvector (a, i) */
void free_drealvector ( double *a,  int i ) {
    free ( (void*) (a+i) ); }

/* A.17. 実数行列領域確保用関数 dmatrix ( nr1, nr2, nl1, nl2 ) */
double **dmatrix ( int nr1, int nr2, int nl1, int nl2 ) {
    // nrow: 行の数, ncol: 列の数
    double **a;
    int i, nrow, ncol;
    nrow = nr2 - nr1 +1;
    ncol  = nl2 - nl1 +1;
    /* 行の確保 */
    if ( ( a = (double **) malloc ( nrow*sizeof (double*) ) ) == NULL )
        { printf ("Memory cannot be allocated !¥n"); exit (1); }
    a = a - nr1; // 行をずらす
    /* 列の確保 */
    for ( i = nr1; i <= nr2; i++ ) a[i] = (double *) malloc (ncol*sizeof (double) );
    for ( i = nr1; i <= nr2; i++ ) a[i] = a[i] - nl1;   // 列をずらす
    return (a); }

/* A.18.実数行列領域解放用関数 free_dmatrix ( a, nr1, nr2, nl1, nl2 ) */
void free_dmatrix ( double **a, int nr1, int nr2, int nl1, int nl2 ) {
    int i;
    for ( i = nr1; i <= nr2; i++ ) free ( (void*) (a[i] + nl1) );
    free ( (void*) (a+nr1) );
}

/* A.19. 整数ベクトル初期化関数 init_vector ( a, nr1, nr2 ) */
void init_intvector ( int *a, int nr1, int nr2 ) {
    int i;
    for ( i = nr1; i <= nr2; i++ ) {    a[i] = 0; }
}

/* A.20. 整数ベクトル初期化関数 init_vector ( a, nr1, nr2 ) */
void init_realvector ( double *a, int nr1, int nr2 ) {
    int i;
    for ( i = nr1; i <= nr2; i++ ) { a[i] = 0.0; }
}

/* A.21. 実数行列初期化関数 init_vector ( a, nr1, nr2, nl1, nl2 ) */
void init_realmatrix (double **a, int nr1, int nr2, int nl1, int nl2 ) {
    for ( int i = nr1; i <= nr2; i++ ) {
        for ( int j = nl1; j <= nl2; j++ ) { a[i][j] = 0.0; } }
}

/* 畳み込み積分 convolution (n1, P1, n2, P2) */
double* convolution ( int n1, double* P1, int n2, double* P2 ) {
    int i, j;
    double* R;
//  R = (double*) malloc ( sizeof (double) *(n1+n2+1));
    if ( ( R = (double *) malloc ( (n1+n2+1)*sizeof (double) ) ) == NULL )
    { printf ("Memory cannot be allocated !¥n"); exit (1); }
    for ( i=0; i<=n1+n2; i++ ) { R[i] = 0.0; }
    for ( i=0; i<n1; i++ ) {
        for( j=0; j<=n2; j++ ) { R[i+j] = R[i+j] + P1[i]*P2[j]; }}
    //for( j=0; j<=n2; j++ ) { R[i+j]+=P1[i]*P2[j]; }}
    return R; }

```