

```
1 #pragma warning( disable: 4101 )
2 #pragma warning( disable: 4996 )
3 #pragma warning( disable: 6031 )
4 #pragma warning( disable: 6001 )
5 #pragma warning( disable: 6385 )
6 #pragma warning( disable: 6386 )
7 #pragma warning( disable: 26451 )
8
9 #include <iostream>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <math.h>
13 #include <string.h>
14 #include <time.h>
15 using namespace std;
16
17
18 /* 諸定数の定義 */
19 #define PI 3.141592653
20 #define C 2.99792458e8
21 #define Epsilon0 8.8541878e-12
22 #define Mu0 PI*4.0e-7
23
24 /* 関数副プログラムの読み込み */
25 // 屈折率波長微分
26 double dndl ( double lamda, double n_lamda, int mater );
27 // 屈折率濃度微分
28 double dndc ( double lamda, int mater );
29 // コア中心屈折率
30 double ncore ( double lamda, int mater );
31 // クラッド屈折率
32 double nclad ( double lamda, int mater );
33 // 第1種変形Bessel関数 In(x)
34 double bessi0 (double x), bessi1 (double x);
35 // 第2種変形Bessel関数 Kn(x)
36 double bessk0 (double x), bessk1 (double x), bessk ( int n, double x );
37 // 係数行列要素計算関数
38 void S_matrix ( double *a, double *b, double *q, int m, int n, double v,
39 double w, double D );
40 // 改訂コレスキー分解
41 void mcholesky ( double *a, double *b, double *ML, double *MD, int m, int
42 n );
43 // 改訂コレスキー分解法により方程式を解く
44 void mcholesky_sol ( double *ML, double *MD, double *R, int m, int n );
45 // 逆べき乗法の初期ベクトル計算
46 void R0 ( double *MD, double *R, int m, int n );
47 // 逆べき乗法の解ベクトル規格化
48 void R_norm ( double *R, int n );
49 // 固有値計算
50 double Eigen ( double *R, double *a, double *b, int n, int m );
51 // 群遅延計算用関数
52 double dbdk_bunbo ( double *R, double D, double w, int m, int n );
53 // 群遅延計算用関数
54 double dbdk_bunshi ( double *R, double *q2, double D, double w, int m, int
55 n );
56 // メモリ領域確保 (整数ベクトル用)
57 int *dintvector ( int i, int j );
58 // メモリ領域解放 (整数ベクトル用)
59 void free_dintvector ( int *a, int i );
60 // メモリ領域確保 (実数ベクトル用)
61 double *drealvector ( int i, int j );
62 // メモリ領域解放 (実数ベクトル用)
63 void free_drealvector ( double *a, int i );
64 // メモリ領域確保 (マトリクス用)
65 double **dmatrix ( int nr1, int nr2, int nl1, int nl2 );
```

```

63 // メモリ領域解放 (マトリクス用)
64 void free_dmatrix ( double **a, int nr1, int nr2, int nl1, int nl2 );
65 // 2次元実数配列初期化
66 void init_realmatrix ( double **a, int nr1, int nr2, int nl1, int nl2 );
67 // 1次元整数配列初期化
68 void init_intvector ( int *a, int nr1, int nr2 );
69 // 1次元実数配列初期化
70 void init_realvector ( double *a, int nr1, int nr2 );
71 // 畳み込み積分実行関数
72 double* convolution ( int n1, double* P1, int n2, double* P2 );
73
74 /* 1. パラメータの定義 */
75 // 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
76 // 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,
77 // 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
78 // k: 波数, delta: 比屈折率, NA: 開口数, aa: 動径座標規格化サイズ
79 // v: 規格化周波数, w: 規格化伝搬定数, D: 規格化コア径
80 // tau: 群速度, beta: 伝搬定数  $\beta$ , dbeta: 伝搬定数刻み幅
81 // 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
82 // GI: 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
83 // R: 規格化横方向電場成分, Rb: 逆べき乗法用入れ子配列
84 // 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
85 // dd, ds: 逆べき乗法における収束判別パラメータ, eig: 固有値
86 // modem: 方位角モード次数, model: 動径モード次数, modep: 主モード次数
87 // Nz: 総ステップ数, Nzout: ファイル出力基準ステップ数
88 // 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
89 // 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
90 // beta: 伝搬定数  $\beta$ , tau: 群遅延, taumin: 最小群遅延, taumax: 最大群遅延
91 // zmax: ファイバ長, dZ: 空間座標刻み幅, Tv: 時間刻み幅
92 // hmn: 電力結合係数, Hmmin: H行列対角要素最小値, Hrowsum:
93 // 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)
94 // nmax: プロファイルループ内におけるLmaxの最大値.
95 // nstd: 相対時間の基準値 (y=0のtauimnを基準としたn=0の補正值)
96 // Dc: 相關長, sigma2: microbendingの軸揺らぎの分散
97 // wo: 摂動設定用パラメータ (0: w/o coupling, 1: w/ coupling_heterogeneity, 2: w/ coupling_microbending)
98 // ftou: ファイル出力設定用パラメータ (0: 部分出力, 1: 全出力)
99 // matdis: 材料分散考慮パラメータ (0:無視, 1:考慮)
100 // nP: インパルス応答格納配列用確保領域
101 // 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
102 // 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)
103 // 行列 H (各節点における伝達関数)
104 // 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)
105
106 int main ( void ) {
107     FILE *fp,*fp2,*fp3,*fp4,*fq,*fr,*fr2,*fr3,*fr4,*fr5,*fr6,*fr7,*fr8,*fr9,*fs,*fs2,*fs3,*fs4;
108     int i, j, l, m, n, x, y, nr, jmax, count = 0;
109     int mater, Nl, N, Nclad, Nbeta, NLP, NLP0, Ntotal, Nwkb, Ptotal, myu, nyu, nstd, nstdmin, nstdmax, mm;
110     int Nz, Nzout, Nf, Nfp, Ti, Li, Lmax, nmax, wo, gi, ss, fout, matdis, scc, nP, Mn, launch, Nxy;
111     double lamda, lamda0, lamdamin, lamdamax, lpmin, lpmax, dlp, fp0, dl, k, omega, A, AA, g, n0, n1, dr;
112     double delta, NA, aa, v, w, D, w0, r0, dx, dy, xx, yy, rr, Rxy, Ein, Emev, Emod, Amev, Amod;
113     double tau, beta, dbeta, bb, eps1, eps2, sum, sumcore, sumclad, Rinf, dd, ds, eig;
114     double deps2, Dc, sigma2, Db, dbmn, E_over, hmn;
115     double zmax, zout, dz, Tv, taumax, taumin, Hmmin, Hrowsum, tauminstd, nctaumax;

```

```

117 double fmin, fmax, df, fpmin, fpmax, dfp, A00, Aw00, Hw, ReHw, ImHw, me,
    bw, tav, rms, Ptot, ap, spct, Cpsum;
118 double *GI, *GC, *q, *q2, *R, *R2, *Rb, *a, *b, *ML, *MD, **Rlp, *Mtau,
    *Mbeta;
119 double **Amin, **Aplu, **H, *alpha, *P, *Pm, *Pg, *M, *Cp, *Pin, *Pout,
    *GIND, *Spin;
120 int *model, *modem, *modep, *pdeg, *kim, *Pnum;
121
122 /** 1. 初期設定 **/
123 /* 入力ファイルの読み込み */
124 if ( (fp = fopen("BW_input.csv", "r")) != NULL ) {
125     char s1 [81], s2 [128];
126     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &mater );
127     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &g );
128     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &A );
129     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &zmax );
130     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lamda0 );
131     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &wo );
132     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &deps2 );
133     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &Dc );
134     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &sigma2 );
135     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &Db );
136     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &launch );
137     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &w0 );
138     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &r0 );
139     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dx );
140     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &matdis );
141     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lamdamin );
142     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lamdamax );
143     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &NI );
144     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &sc );
145     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lpmin );
146     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &lpmax );
147     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dlp );
148     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Ti );
149     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &gi );
150     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &fout );
151     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &AA );
152     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dr );
153     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &aa );
154     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Nbeta );
155     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &eps1 );
156     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &eps2 );
157     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &jmax );
158     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &dz );
159     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &Tv );
160     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &zout );
161     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &fmin );
162     fscanf ( fp, "%[^,], %[^,], %lf¥n", s1, s2, &fmax );
163     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &Nf );
164     fscanf ( fp, "%[^,], %[^,], %d¥n", s1, s2, &NP );
165 } else { printf ( " U cannot open the file !¥n" ); exit ( EXIT_FAILURE ); }
166
167 /* 入力パラメータの単位変換 */
168 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)
169 if ( NI%2 != 0 ) { NI = NI+1; } dl = ( lamdamax - lamdamin ) / (double)
    NI; // 材料分散評価用波長ステップ
170 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
171 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)
172 lpmin = lpmin*1.0e-9, lpmax = lpmax*1.0e-9, dlp = dlp*1.0e-9; // 単位変
    換 (m)

```

```

173 dfp = dfp*1.0e-3; fpmmin = fpmmin*1.0e-3; fpmmax = fpmmax*1.0e-3; // 単位変換 (THz)
174 A = A*1.0e-6; dr = dr*1.0e-9; AA = AA *1.0e-6; // 単位変換 (m)
175 w0 = w0*1.0e-6; r0 = r0*1.0e-6; dx = dy = dx*1.0e-9; // 単位変換 (m)
176 Nxy = (int) ( 5.0*w0 / dx ); // とりあえず4w0の範囲
177 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
178 fmin = fmin*1.0e-3; fmax = fmax*1.0e-3; df = ( fmax-fmin ) / (double) Nf; // 単位変換 (THz(1/ps))
179 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.
180 D = A / aa; // 規格化コア径の換算.
181 lamda = lamda0; Aw00 = 0.0; spct = Cpsum = 0.0; // 変数初期化
182 xx = yy = rr = Rxy = 0.0;
183 if ( matdis == 0 ) { Nl = 0; }
184
185 fclose (fp);
186
187 /* 入力データの格納 */
188 // 入射光時間波形
189 Pin = drealvector ( 0, Ti ); init_realvector ( Pin, 0, Ti );
190 if ( (fr5 = fopen ( "Input_pulse_waveform.csv", "r" ) ) != NULL ) {
191     for ( i = 0; i < Ti; i++ ) { fscanf ( fr5, "%lf\n", &Pin[i] ); }
192 } else { printf ( " U cannot open the file !\n" ); exit ( EXIT_FAILURE ); }
193 fclose (fr5);
194 // 入射光スペクトル
195 Spin = drealvector ( 0, Nfp ); init_realvector ( Spin, 0, Nfp );
196 if ( (fr8 = fopen ( "Source_spectrum.csv", "r" ) ) != NULL ) {
197     for ( i = 0; i <= Nfp; i++ ) { fscanf ( fr8, "%lf\n", &Spin[i] ); }
198 } else { printf ( " U cannot open the file !\n" ); exit ( EXIT_FAILURE ); }
199 fclose (fr8);
200 // 屈折率分布 (@589nm) & ドーパント濃度分布分布
201 if ( gi == 1 ) {
202     GIND = drealvector ( 0, N ); init_realvector ( GIND, 0, N );
203     if ( (fr7 = fopen ( "GI_profile_NaD.csv", "r" ) ) != NULL ) {
204         for ( i = 0; i <= N; i++ ) { fscanf ( fr7, "%lf\n", &GIND
205             [i] ); }
206     } else { printf ( " U cannot open the file !\n" ); exit
207         ( EXIT_FAILURE ); }
208     GC = drealvector ( 0, N ); init_realvector ( GC, 0, N );
209     for ( i = 0; i <= N; i++ ) { GC[i] = ( GIND[i] - GIND[N] ) / dndc
210         ( 589.0, mater ); }
211     fclose (fr7);
212
213 /* 出力ファイルの設定 */
214 // 評価条件一覧
215 if ( ( fp2 = fopen ( "BW_setting.csv", "w" ) ) != NULL ) { }
216 else { printf ( " U cannot open the file !\n" ); exit ( EXIT_FAILURE ); }
217 // 屈折率プロファイル
218 if ( ( fp3 = fopen ( "BW_profile.csv", "w" ) ) != NULL ) {
219     if ( gi == 0 ) { fprintf ( fp3, "r [%u m], n, q, q2\n"; } if ( gi == 1 )
220         { fprintf ( fp3, "r [%u m], c [wt pct], n, q, q2\n"; }
221 } else { printf ( " U cannot open the file !\n" ); exit ( EXIT_FAILURE ); }
222 // 有限要素法計算結果
223 if ( ( fp4 = fopen ( "FEM_result.csv", "w" ) ) != NULL ) { }
224 else { printf ( " U cannot open the file !\n" ); exit ( EXIT_FAILURE ); }
225 // 電力結合係数計算結果
226 if ( ( fq = fopen ( "Hmn_result.csv", "w" ) ) != NULL ) {
227     if ( wo == 1 ) {
228         fprintf ( fq, "μ, m (mode μ), l (mode μ), p (mode μ), β (mode
229             μ), ν, m (mode ν), l (mode ν), p (mode ν), β (mode ν), Δp, |
230             Δβ |, E-field overlap, h μ ν\n"; }
231     if ( wo == 2 ) {
232         fprintf ( fq, "μ, m (mode μ), l (mode μ), p (mode μ), β (mode

```

```

         $\mu$ ),  $\nu$ , m (mode  $\nu$ ), l (mode  $\nu$ ), p (mode  $\nu$ ),  $\beta$  (mode  $\nu$ ),  $\Delta p$ , |
         $\Delta \beta$  |, h  $\mu$   $\nu$  ¥n"); } }
228     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
229     // 行列H
230     if ( ( fr = fopen ( "CPE_Hmatrix.csv", "w" ) ) != NULL ) { }
231     else { printf ( "The file cannot be opened !¥n" ); exit
232             ( EXIT_FAILURE ); }
233     // インパルス応答波長成分 P
234     if ( ( fr2 = fopen ( "CPE_Impulse-responce.csv", "w" ) ) != NULL ) { }
235     else { printf ( "The file cannot be opened !¥n" ); exit
236             ( EXIT_FAILURE ); }
237     // 出射波形波長成分 Pout
238     if ( ( fr6 = fopen ( "CPE_Output-pulse-waveform.csv", "w" ) ) != NULL ) { }
239     else { printf ( "The file cannot be opened !¥n" ); exit
240             ( EXIT_FAILURE ); }
241     // モードパワー分布 Pm
242     if ( ( fr3 = fopen ( "CPE_Mode-power-distribution.csv", "w" ) ) !=
243             NULL ) { }
244     else { printf ( "The file cannot be opened !¥n" ); exit
245             ( EXIT_FAILURE ); }
246     // モード群パワー分布 Pg
247     if ( ( fr9 = fopen ( "CPE_Group-power-distribution.csv", "w" ) ) !=
248             NULL ) { }
249     else { printf ( "The file cannot be opened !¥n" ); exit
250             ( EXIT_FAILURE ); }
251     // 周波数応答 M
252     if ( ( fr4 = fopen ( "CPE_Frequency-respnse.csv", "w" ) ) != NULL ) {
253         fprintf ( fr4, ", " ); for ( j = 0; j <= Nf; j++ ) { fprintf ( fr4,
254             "%f, ", (fmin*1.0e3)+(double)j*(df*1.0e3) ); } fprintf ( fr4,
255             "¥n" ); }
256     else { printf ( "The file cannot be opened !¥n" ); exit
257             ( EXIT_FAILURE ); }
258     // 主要な結果
259     if ( ( fs = fopen ( "BW_result.csv", "w" ) ) != NULL ) {
260         if ( matdis == 0 ) { fprintf ( fs, "g value, Nlp, Nlp0, tstart
261             [ns], length[m], pulse broadening [ps], -3dB bandwidth [GHz], tav
262             [ps], Ptot, Aw00, spct¥n" ); }
263         if ( matdis == 1 ) { fprintf ( fs, "g value, length[m], pulse
264             broadening [ps], -3dB bandwidth [GHz], tav[ps], Ptot, Aw00, spct
265             ¥n" ); } }
266     // else { fprintf ( fs, "g value, length[m], -3dB bandwidth [GHz], rms
267     width [ps], tav[ps], Ptot, Aw00, spct¥n" ); } }
268     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
269     // インパルス応答
270     if ( ( fs3 = fopen ( "BW_Impulse-responce.csv", "w" ) ) != NULL ) {
271         for ( n = 0; n <= nP; n++ ) { fprintf ( fs3, "%f, ", double(n)
272             *Tv ); } fprintf ( fs3, "¥n" ); }
273     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
274     // 出射波形波長成分 Pout
275     if ( ( fs4 = fopen ( "BW_Output-pulse-waveform.csv", "w" ) ) != NULL ) { }
276     else { printf ( "The file cannot be opened !¥n" ); exit
277             ( EXIT_FAILURE ); }
278     // 光源スペクトル
279     if ( ( fs2 = fopen ( "BW_source-spectrum.csv", "w" ) ) != NULL ) {
280         for ( i = 0; i <= Nfp; i++ ) { fprintf ( fs2, "%f, %f, %f¥n",
281             fpmin+double(i)*dfp, 1.0e-3*C / ( fpmin+double(i)*dfp ), Spin
282             [Nfp-i] ); } }
283     else { printf ( "U cannot open the file!¥n" ); exit ( EXIT_FAILURE ); }
284     printf ( "Total spatial steps Nf: %d¥n", Nf );

```



```

269  /** 2. 計算開始 **/
270  /* 変数の初期化 */
271  Aw00 = tauminstd = 0.0; nmax = nstd = nstdmin = nstdmax = Mn = 0;
272  /* 配列の記憶領域確保および初期化 */
273  if ( matdis == 1 ) { P = dreallvector ( 0, nP ), init_realvector ( P,
274  0, nP ); }
275  Pnum= dintvector ( 0, N1 ); init_intvector ( Pnum, 0, N1 );
276  M = dreallvector ( 0, Nf ); init_realvector ( M, 0, Nf );
277  for ( y = 0; y <= N1; y++ ) {
278  /* 波長指定 */
279  if ( matdis == 1 ) { lamda = lamdamax - y*d1; }
280  /* 各種パラメタ計算 */
281  if ( gi == 0 ) { n1 = ncore ( lamda*1.0e9, mater ); n0 = nclad
282  ( lamda*1.0e9, mater ); }
283  if ( gi == 1 ) { n1 = dndc ( lamda*1.0e9, mater )*GC[0] + nclad
284  ( lamda*1.0e9, mater ); n0 = nclad ( lamda*1.0e9, mater ); }
285  delta = (n1*n1-n0*n0) / (2.0*n1*n1), NA = sqrt (n1*n1-n0*n0);
286  k = 2.0*PI / lamda; omega = 2.0*PI*C / lamda; v = k*aa*n1*sqrt
287  ( 2.0*delta );
288  Nwkb = int( (1.0/4.0)*( g / (g+2.0) )*(k*k)*(n1*n1)*delta*
289  (A*A) );
290  /* 変数の初期化 */
291  bb = -1; dbeta= k*( n1 - n0 ) / (double) Nbeta;
292  NLP= 0; NLP0 = Ntotal = Ptotal = 0;
293  /* 配列の記憶領域確保および初期化 */
294  GI = dreallvector ( 0, N ); init_realvector ( GI, 0, N );
295  q = dreallvector ( 0, N ); init_realvector ( q, 0, N );
296  q2 = dreallvector ( 0, N ); init_realvector ( q2, 0, N );
297  R = dreallvector ( 0, N ); init_realvector ( R, 0, N );
298  R2 = dreallvector ( 0, N ); init_realvector ( R2, 0, N );
299  Rb = dreallvector ( 0, N ); init_realvector ( Rb, 0, N );
300  a = dreallvector ( 0, N ); init_realvector ( a, 0, N );
301  b = dreallvector ( 0, N ); init_realvector ( b, 0, N );
302  ML = dreallvector ( 0, N ); init_realvector ( ML, 0, N );
303  MD = dreallvector ( 0, N ); init_realvector ( MD, 0, N );
304  modem = dintvector ( 0, 2*Nwkb ); init_intvector ( modem, 0,
305  2*Nwkb );
306  model = dintvector ( 0, 2*Nwkb ); init_intvector ( model, 0,
307  2*Nwkb );
308  modep = dintvector ( 0, 2*Nwkb ); init_intvector ( model, 0,
309  2*Nwkb );
310  pdeg = dintvector ( 0, Nwkb ); init_intvector ( pdeg, 0, Nwkb );
311  Mbeta = dreallvector ( 0, 2*Nwkb ); init_realvector ( Mbeta, 0,
312  2*Nwkb );
313  Mtau = dreallvector ( 0, 2*Nwkb ); init_realvector ( Mtau, 0,
314  2*Nwkb );
315  Rlp = dmatrix ( 0, 2*Nwkb, 0, N ); init_realmatrix ( Rlp, 0,
316  2*Nwkb, 0, N );
317  /* 屈折率分布, 比屈折率分布および波長微分分布 */
318  if ( gi == 0 ) {
319  for ( j = 0; j <= N; j++ ) { GI[j] = n1*sqrt
320  ( 1.0-2.0*delta*pow ( ((double) j / (double) N), g ); }
321  for ( j = 0; j <= N; j++ ) { q[j] = (GI[j]*GI[j] - n0*n0) /
322  (n1*n1 - n0*n0); }
323  for ( j = 0; j <= N; j++ ) { q2[j] = GI[j]*GI[j] - (lamda*GI
324  [j]*dnd1 (lamda*1.0e9, GI[j], mater)) / (1 - (lamda/GI[j])
325  *dnd1 (lamda*1.0e9, GI[j], mater)); }
326  if ( y == N1/2 || fout == 1 ) { for ( j = 0; j <= N; j++ )
327  { fprintf ( fp3, "%f, %f, %f, %f¥n", A*1.0e6* (double) j /
328  (double) N, GI[j], q[j], q2[j] ); }}
329  if ( gi == 1 ) {
330  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
315 for ( j = 0; j <= N; j++ ) { q[j] = (GI[j]*GI[j] - GI[N]*GI
[N]) / (GI[0]*GI[0] - GI[N]*GI[N]); }
316 for ( j = 0; j <= N; j++ ) { q2[j] = GI[j]*GI[j] - (lamda*GI
[j]*dnd1 (lamda*1.0e9, GI[j], mater)) / (1 - (lamda/GI[j])
*dnd1 (lamda*1.0e9, GI[j], mater)); }
317 if ( y == NI/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] ); }}
318
319 /* 評価条件の出力 */
320 if ( y == 0 ) {
321     fprintf ( fp2, "Material, mater, %d¥n", mater );
322     fprintf ( fp2, "Central wavelength, λ0, %f nm¥n",
lamda0*1e9 );
323     fprintf ( fp2, "Step size of wavelength, dl, %f nm¥n",
dl*1e9 );
324     fprintf ( fp2, "Partition number of evaluated wavelength, NI,
%d¥n", NI );
325     fprintf ( fp2, "Core radius, A, %f μm¥n", A*1e6 );
326     fprintf ( fp2, "Analysis region in radial axis, AA, %f μm¥n",
AA*1e6 );
327     fprintf ( fp2, "Refractive index at the core center, n1, %f
¥n", n1 );
328     fprintf ( fp2, "Refractive index in the cladding, n0, %f¥n",
n0 );
329     fprintf ( fp2, "Relative refractive index, Δ, %f¥n", delta );
330     fprintf ( fp2, "Numerical aperture, NA, %f¥n", NA );
331     fprintf ( fp2, "Step size of the elements, dr, %f, nm¥n",
dr*1e9 );
332     fprintf ( fp2, "Step size of propagation constant, dβ, %f¥n",
dbeta );
333     fprintf ( fp2, "Partition number of propagation constant,
Nβ, %d¥n", Nbeta );
334     fprintf ( fp2, "Partition number of fiber core radius, N, %d
¥n", N );
335     fprintf ( fp2, "Partition number of fiber cladding, Nclad, %d
¥n", Nclad );
336     fprintf ( fp2, "Maximum allowable error for convergence
solution vector, eps1, %f¥n", eps1 );
337     fprintf ( fp2, "Maximum allowable error of zero eigen value,
eps2, %f¥n", eps2 );
338     fprintf ( fp2, "Maximum number of iterations in inverse power
method, jmax, %d¥n", jmax );
339     fprintf ( fp2, "Correlation length, Dc, %e, nm¥n", Dc*1e9 );
340     fprintf ( fp2, "Mean square of dielectric constant
fluctuation, <dε2>, %e¥n", deps2 );
341     fprintf ( fp2, "Total fiber length, zmax, %e, m¥n", zmax );
342     fprintf ( fp2, "Step size of fiber length, dz, %e, m¥n", dz );
343     fprintf ( fp2, "Step size of time, Tv, %e, ps¥n", Tv );
344     fprintf ( fp2, "Total spatial steps, Nz, %d¥n", Nz );
345     fprintf ( fp2, "File output step interval, Nzout, %d¥n",
Nzout );
346     fprintf ( fp2, "Minimum evaluated frequency, fminx, %e, GHz¥n",
fmin*1.0e3 );
347     fprintf ( fp2, "Maximum evaluated frequency, fmax, %e, GHz¥n",
fmax*1.0e3 );
348     fprintf ( fp2, "Step size of evaluated frequency, df, %e, GHz
¥n", df*1.0e3 );
349     printf ( "Central wavelength λ0: %f nm¥n", lamda0*1e9 );
350     printf ( "Step size of wavelength dl: %f nm¥n", dl*1e9 );
351     printf ( "Partition number of evaluated wavelength NI: %d¥n",
NI );

```

```

352     printf ( "Minimum evaluated spectral frequency, fpmin, %f THz \n", fpmin );
353     printf ( "Maximum evaluated spectral frequency, fpmax, %f THz \n", fpmax );
354     printf ( "Core radius A: %f  $\mu$ m\n", A*1.0e6 );
355     printf ( "Analysis region AA: %f  $\mu$ m\n", AA*1e6 );
356     printf ( "Refractive index at the core center n1: %f\n", n1 );
357     printf ( "Refractive index in the cladding n0: %f\n", n0 );
358     printf ( "Relative refractive index  $\Delta$ : %f\n", delta );
359     printf ( "Numerical aperture NA: %f\n", NA );
360     printf ( "Step size of the elements dr: %f nm\n", dr*1.0e9 );
361     printf ( "Partition number of fiber core radius N: %d\n", N );
362     printf ( "Partition number of fiber cladding Nclad: %d\n", Nclad );
363     printf ( "Step size of propagation constants  $d\beta$ : %f\n", dbeta );
364     printf ( "Partition number of propagation constants  $N\beta$ : %d\n", Nbeta );
365     printf ( "Maximum allowable error for convergence solution vector eps1: %f\n", eps1 );
366     printf ( "Maximum allowable error of zero eigen value eps2: %f\n", eps2 );
367     printf ( "Maximum number of iterations in inverse power method jmax: %d\n", jmax );
368     printf ( "Correlation length Dc: %e m\n", Dc );
369     printf ( "Mean square of dielectric constant fluctuation < $d\epsilon^2$ >: %f\n", deps2 );
370     printf ( "Total fiber length zmax: %e m\n", zmax );
371     printf ( "Step size of fiber length dz: %e m\n", dz );
372     printf ( "Step size of time Tv: %e ps\n", Tv );
373     printf ( "Total spatial steps Nz: %d\n", Nz );
374     printf ( "File output step interval Nzout: %d\n", Nzout );
375     printf ( "Minimum evaluated frequency fmin: %e GHz\n", fmin*1.0e3 );
376     printf ( "Maximum evaluated frequency fmax: %e GHz\n", fmax*1.0e3 );
377     printf ( "Step size of evaluated frequency df: %e GHz\n", df*1.0e3 );
378     printf ( "Wavelength: %f nm (%d/%d)\n", lamda*1.0e9, y, Nl );
379     if ( wo == 0 ) { printf ( "without mode coupling\n" ); }
380     if ( wo == 1 ) { printf ( "with microscopic heterogeneities\n" ); }
381     if ( wo == 2 ) { printf ( "with microbending\n" ); }
382     fprintf ( fp2, "lamda=%f nm\n", lamda*1.0e9 );
383     fprintf ( fp4, "lamda=%f nm\n", lamda*1.0e9 );
384     if ( y == Nl/2 || fout == 1 ) {
385         fprintf ( fp4,
386             "m, l, p, tau, beta, ne, confinement, R_infinite, eig, ");
387         for ( j = 0; j <= N; j++ ) { fprintf ( fp4, "%f, ", (double) j*dr*1.0e6 ); }
388         fprintf ( fp4, "\n" );
389         fprintf ( fq, "lamda=%f nm\n", lamda*1.0e9 );
390         fprintf ( fr2, "lamda=%f nm\n", lamda*1.0e9 );
391         fprintf ( fr3, "lamda=%f nm\n", lamda*1.0e9 );
392     }
393     /** 2. モード解析 (FEM. cpp) */
394     for ( m = 0; ; m++ ) {
395         l = 1; beta = k*n1;
396         /
397         *****
398         *****/
399         for ( ; ; ) {
400             /* 係数行列計算および改訂コレスキー分解 */
401             w = aa*sqrt ( (beta*beta) - (k*k)*(n0*n0) );

```



```

399 S_matrix ( a, b, q, m, N, v, w, D );
400 mcholesky ( a, b, ML, MD, m, N );
401 /* 初期ベクトル R0 の付与 */
402 RO ( MD, R, m, N );
403 /* 連立一次方程式 SR=(LDL)R=bR の反復評価 */
404 for ( j = 0 ; j <= N; j++ ) {
405     for ( i = 0; i <= N; i++ ) { Rb[i] = R[i]; }
406     mcholesky_sol ( ML, MD, R, m, N );
407     R_norm ( R, N );
408     /* 収束判定 */
409     dd = 0, ds = 0;
410     for ( i = 0; i <= N; i++ ) {
411         dd = dd + (Rb[i] - R[i])*(Rb[i] - R[i]);
412         ds = ds + (Rb[i] + R[i])*(Rb[i] + R[i]); }
413     if ( dd < eps1 || ds < eps1 ) break;
414     if ( j >= jmax ) goto next;
415     // ① RとRbの成分差ddが0に漸近すれば収束 (break) .
416     // ② RとRbの成分和dsが0に漸近すれば中止 (break) .
417     // 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) .
418 }
419 /* 固有値の計算 */
420 eig = Eigen ( R, a, b, N, m );
421 /* 固有値の妥当性評価 */
422 // 「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
423 if ( eig == bb ) {
424     dbeta = k*(n1 - n0) / (double) Nbeta;
425     beta = beta - 1.0*dbeta;
426     continue; }
427
428 // ① 「0< 収束固有値 eig < eps2」であれば零固有値として採用
429 if ( 0.0 < eig && eig < eps2 ) {
430     /* 横方向電場成分Rの規格化 (パワーを1Wとする) */
431
432     sum = sumcore = sumclad = 0.0;
433     for ( j = 0; j < N; j++ ) { sumcore = sumcore + R[j]*R[j]*(j*dr)*dr; }
434     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; }
435     for ( j = 0; j <= N; j++ ) { R2[j] = R[j] * sqrt ( (omega*Muo) / (PI*beta*(sumcore + sumclad)) ); }
436     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]
437     /* 計算結果の出力 */
438     Rinf = R[N]*( bessk (m, w*(Nclad*dr+A)) / bessk (m, w*A) );
439     if ( y == Nl/2 || fout == 1 ) { fprintf ( fp4, "%d, %d, %d, %f, %f, %f, %f, %f, %f, %f, %f\n", m, l, 2*l+m-1, tau, beta, beta/k, sumcore/(sumcore+sumclad), Rinf, eig ); }
440     modem[NLP] = m, model[NLP] = l, Mtau[NLP] = tau, Mbeta[NLP] = beta;
441     modep[NLP] = 2*l + m - 1; if ( Ptotal < modep[NLP] ) { Ptotal = modep[NLP]; } // 主モード次数
442     for ( j = 0; j <= N; j++ ) { if ( fout == 1 || y == Nl/2 ) { fprintf ( fp4, "%f, ", R2[j] ); } Rlp[NLP][j] = R2[j]; }
443     if ( y == Nl/2 || fout == 1 ) { fprintf ( fp4, "\n" ); }
444     //printf ( "%d, %d, %f, %f, %f, %f, %f\n", m, l, tau, beta, beta/k, eig, R[N] );

```

```

445         dbeta = k*(n1 - n0) / (double) Nbeta;
446         bb = eig;
447         l = l + 1;
448         NLP = NLP + 1; if ( m == 0 ) { NLP0 = NLP0 + 1; }
449         count = 0; }
450
451         // ② 「0 < 収束固有値 eig」かつ「-1 < 前回値 bb < 0」で
         あれば
452         if ( eig > 0.0 && bb < 0.0 && (fabs(bb) < 1.0) ) {
453             beta = beta + dbeta;
454             dbeta = dbeta / 2.0;
455             count = count + 1; }
456
457         // ③ その他
458         else { bb = eig; count = 0; }
459
460         if ( count > 1000 ) {
461             dbeta = k*(n1 - n0) / (double) Nbeta;
462             beta = beta - dbeta; }
463     next:
464         beta = beta - dbeta;
465         if ( beta < k*n0 ) break;
466     }
467     /
         *****
         *****/
468     if ( l == 1 ) { Ntotal = ( NLP0 ) * 2 + ( NLP - NLP0 ) * 4;
         break; } // mが最高次数に到達
469 }
470
471     free_drealvector ( GI, 0 ); free_drealvector ( q, 0 );
         free_drealvector ( q2, 0 );
472     free_drealvector ( R, 0 ); free_drealvector ( R2, 0 );
         free_drealvector ( Rb, 0 );
473     free_drealvector ( a, 0 ); free_drealvector ( b, 0 );
474     free_drealvector ( ML, 0 ); free_drealvector ( MD, 0 );
475
476     printf ( "Total numbers of LPml modes (WKB) Nwkb: %d¥n", Nwkb );
477     printf ( "Total numbers of LPml modes NLP: %d¥n", NLP );
478     printf ( "Total numbers of LP0l modes NLP0: %d¥n", NLP0 );
479     printf ( "Total numbers of all modes Ntotal: %d¥n", Ntotal );
480     printf ( "Total numbers of all mode groups Ptotal: %d¥n",
         Ptotal );
481
482     /** 3. LPモード特性の整理 **/
483     /* 群遅延範囲 */
484     // 各ループの単位長群遅延範囲 (taumin < tau < taumax)
485     taumax = tau[0];
486     for ( myu = 0; myu < NLP; myu++ ) {
487         if ( taumax < Mtau[myu] ) { taumax = Mtau[myu]; }
488         if ( taumin > Mtau[myu] ) { taumin = Mtau[myu]; }
489     }
490     // 各ループの最高次モード群単位長群遅延
491     nctaumax = taumin;
492     for ( myu = 0; myu < NLP; myu++ ) {
493         if ( modep [myu] == Ptotal ) { if ( nctaumax < Mtau[myu] )
         { nctaumax = Mtau[myu]; } }
494     }
495     // 各ループの基準時間補正要素数 (nstd)
496     if ( y == 0 ) { tauminstd = taumin; }
497     nstd = (int) ( (taumin - tauminstd)*zmax / Tv );
498     printf ( "nstd:%d,nstdmin:%d¥n", nstd, nstdmin );
499     fprintf ( fp2, "Minimum group delay per unit length,taumin,%e,ps/
         m¥n", taumin );
500     fprintf ( fp2, "Maximum group delay per unit length,taumax,%e,ps/
         m¥n", taumax );
         // 各ループの最大群遅延差 (Lmax)

```

```

501     Lmax = (int) ( (( taumax - taumin )*( (double) Nz*dz ) / Tv) +
                    0.5 );
502     fprintf ( fp2, "Total time steps, Lmax, %d¥n", Lmax );
503     if ( Lmax > ( 1.0e9 / (8.0*NLP) ) ) { printf ( "Memory over!
¥n" ); exit ( EXIT_FAILURE ); }
504     printf ( "Required memory for CPE analysis: %fGB¥n", (double)
                    (Lmax*NLP*8) /1.0e9 );
505     // プロファイルループの最大郡遅延差 (nmax)
506     if ( Lmax > nmax ) { nmax = Lmax; }
507     fprintf ( fp2, "Maximum time step, nmax, %d¥n", nmax );
508     // インパルス応答格納配列数 (Pnum)
509     if ( nstd == 0 ) { Pnum [y] = nmax + ( nstdmax - nstdmin ); }
510     if ( nstd > 0 ) {
511         if ( nstd < nstdmax ) { Pnum [y] = nmax + ( nstdmax -
                    nstdmin ); }
512         else { Pnum [y] = nmax + ( nstdmax - nstdmin ) + ( nstd -
                    nstdmax ); } }
513     if ( nstd < 0 ) {
514         if ( nstd < nstdmin ) { Pnum [y] = nmax + ( nstdmax -
                    nstdmin ) + ( nstdmin - nstd ); }
515         else { Pnum [y] = nmax + ( nstdmax - nstdmin ); } }
516     fprintf ( fp2, "Net total time steps, Pnum, %d¥n", Pnum [y] );
517
518     /* 配列の記憶領域確保および初期化 */
519     H = dmatrix ( 0, NLP-1, 0, NLP-1 ); init_realmatrix ( H, 0,
                    NLP-1, 0, NLP-1 );
520     Amin = dmatrix ( 0, NLP-1, 0, Lmax ); init_realmatrix ( Amin, 0,
                    NLP-1, 0, Lmax );
521     Aplu = dmatrix ( 0, NLP-1, 0, Lmax ); init_realmatrix ( Aplu, 0,
                    NLP-1, 0, Lmax );
522     alpha = drealvector ( 0, NLP-1 ); init_realvector ( alpha, 0,
                    NLP-1 );
523     kim = dintvector ( 0, NLP-1 ); init_intvector ( kim, 0, NLP-1 );
524     Pm = drealvector ( 0, NLP-1 ); init_realvector ( Pm, 0, NLP-1 );
525     Pg = drealvector ( 0, Ptotal-1 ); init_realvector ( Pg, 0,
                    Ptotal-1 );
526     if ( matdis == 0 ) { P = drealvector ( 0, Lmax ); init_realvector
                    ( P, 0, Lmax ); }
527     A00 = 0.0;
528
529     /** 4. LPモード  $\mu$  とLPモード  $\nu$  間の電力結合係数計算 **/
530     /* H行列の算出 */
531     /
532     *****
533     if ( wo == 0 ) { for ( myu = 0; myu < NLP; myu++ ) { H[myu][myu]
                    = 1.0; } }
534     else {
535         // ミクロ不均一構造
536         if ( wo == 1 ) { dbmn = 0.0; E_over = 0.0; hmn =0.0;
537         #pragma omp parallel
538         {
539             #pragma omp for
540             for ( myu = 0; myu < NLP; myu++ ) {
541                 for ( nyu = 0; nyu < NLP; nyu++ ) {
542                     dbmn = Mbeta [myu] - Mbeta [nyu];
543                     for ( n = 1; n <= N; n++ ) {
544                         E_over = E_over + (2.0*PI)*( Rlp[myu][n]*Rlp[nyu]
                    [n] )*( Rlp[myu][n]*Rlp[nyu][n] )*(double) (n-1)*dr*dr; }
545
546                     if ( modep[myu] == Ptotal || modep[nyu] == Ptotal ) { hmn
                    = 0.0; } // 最高次モードは無視
547                     else {

```

```

548      hmn = deps2*((omega*omega*(PI*sqrt(PI))*(Dc*Dc*Dc)) /
      8.0)*exp(-(dbmn*dbmn)*(Dc*Dc) / 4.0)*E_over; }

549
550      if ( myu == nyu ) { H[myu][nyu] = 0.0; } else { H[myu]
      [nyu] = hmn*dz; } // 仮入力

551
552      if ( y == Nl/2 || fout == 1 ) {
553          if ( myu != nyu && myu > nyu ) {
554              // if ( myu != nyu ) {
555                  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],
556                      nyu, modem [nyu], model [nyu], modep [nyu],
      Mbeta [nyu], abs(modep[myu]-modep[nyu]), fabs(dbmn),
      E_over, hmn ); }}
557      E_over = 0.0; }}
558  }
559
560  }
561
562  // マイクロベンディング
563  if ( wo == 2 ) { dbmn = 0.0; hmn = 0.0;
564  for ( myu = 0; myu < NLP; myu++ ) {
565      for ( nyu = 0; nyu < NLP; nyu++ ) {
566          dbmn = Mbeta [myu] - Mbeta [nyu];
567
568          if ( modep[myu] == Ptotal || modep[nyu] == Ptotal ) { hmn
      = 0.0; } // 最高次モードは無視
569          else if ( abs (modep [myu] - modep [nyu]) == 1 ) {
570              if ( modep[myu] > modep [nyu] ) { mm = modep[myu] ; }
      else { mm = modep[nyu]; }
571              hmn = (1.0/8.0)*(n1*k*A)*(n1*k*A)*pow( ((double)mm/
      (double)Ptotal), 4.0/(g+2.0) )
572              *sigma2*sqrt(PI)*Db*exp(-(dbmn*dbmn)*(Db*Db) /
      4.0); }
573          else { hmn = 0.0; }
574
575          if ( myu == nyu ) { H[myu][nyu] = 0.0; } else { H[myu]
      [nyu] = hmn*dz; } // 仮入力
576          if ( y == Nl/2 || fout == 1 ) {
577              if ( myu != nyu && myu > nyu ) {
578                  // if ( myu != nyu ) {
579                      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],
580                      nyu, modem [nyu], model [nyu], modep [nyu],
      Mbeta [nyu], abs(modep[myu]-modep[nyu]), fabs(dbmn),
      hmn ); }}
581          hmn = 0.0; }}
582  }
583
584
585  /* H行列の算出 */
586  /
      *****
      *****/
587  hmn = 0.0; myu = 0; nyu = 0;
588  // 対格要素
589  for ( myu = 0; myu < NLP; myu++ ) { Hrowsum = 0.0;
590  for ( nyu = 0; nyu < NLP; nyu++ ) {
591      if ( modem[nyu] == 0 ) { Hrowsum = Hrowsum + H[myu][nyu]; }
      else { Hrowsum = Hrowsum + 2.0*H[myu][nyu]; }}
592  H[myu][myu] = 1.0 - ( 2.0*alpha[myu]*dz + Hrowsum ); }
593  // 非対格要素
594  for ( myu = 0; myu < NLP; myu++ ) {
595  for ( nyu = 0; nyu < NLP; nyu++ ) {
596      if ( myu != nyu ) { if ( modem[myu] == 0 ) { H[myu][nyu] =

```

```

1.0*H[myu][nyu]; } else { H[myu][nyu] = 2.0*H[myu]
[nyu]; } } } }
597 // 安定条件の確認
598 Hmmin = H[0][0];
599 for ( m = 1; m < NLP; m++ ) { if ( Hmmin > H[m][m] ) { Hmmin =
H[m][m]; } }
600 if ( Hmmin < 0 ) { printf ( "Too large Δz value! Change the
value appropriately!¥n" ); exit ( EXIT_FAILURE ); } }
601 /* H行列の出力*/
602 if ( y == Nl/2 && fout == 1 ) {
603     for ( myu = 0; myu < NLP; myu++ ) {
604         for ( nyu = 0; nyu < NLP; nyu++ ) { fprintf ( fr, "%f,", H
[myu][nyu] ); } fprintf ( fr, "¥n" ); } }
605
606
607
608 /* 励振条件設定 (A+行列の算出) */
609 // OFL condition
610 if ( launch == 0 ) {
611     for ( m = 0; m < NLP; m++ ) {
612         if ( matdis == 0 ) { if ( modem[m] == 0 ) { Aplu[m][0] =
100.0; } else { Aplu[m][0] = 200.0; } } // 縮退数の考慮
613         if ( matdis == 1 ) {
614             if ( modem[m] == 0 ) { Aplu[m][0] = 100.0*Spin[ (int)
((lamda -lpmin)/dlp) ]; }
615             else { Aplu[m][0] = 200.0*Spin[ (int)((lamda -lpmin)/
dlp) ]; } } } }
616
617
618 // RML condition
619 if ( launch == 1 ) {
620 #pragma omp parallel
621 {
622 #pragma omp for
623     for ( m = 0; m < NLP; m++ ) { Amev = Amod = 0.0;
624         for ( i = 0; i < Nxy; i++ ) { xx = ( r0 - (double)
Nxy*dx / 2.0 ) + (double)i*dx;
625         for ( j = 0; j < Nxy; j++ ) { yy = - ( (double)
Nxy*dy / 2.0 ) + (double)j*dy;
626         rr = sqrt ( xx*xx + yy*yy ); nr = (int) ( rr / dr )
; // 切り捨て?
627         if ( nr == 0 ) { Rxy = Rlp [m][0] + ( Rlp [m][1] -
Rlp [m][0] ) * (rr /dr); }
628         else { Rxy = Rlp [m][nr] + ( Rlp [m][nr+1] - Rlp [m]
[nr]) * ((rr - (double)nr*dr) /dr); } // interpolation
629         else if ( nr <= N ) { Rxy = Rlp [m][nr] + ( Rlp [m]
[nr+1] - Rlp [m][nr] ) * ((rr - (double)nr*dr) /dr); } //
core (interpolation)
630         else if ( nr > N ) { Rxy = Rlp [m][N]*( bessk (m,
w*rr) / bessk (m, w*A) ); } // cladding
631         Emev = Rxy*cos( (double)(modem[m])*atan2(yy, xx) );
632         Emod = Rxy*sin( (double)(modem[m])*atan2(yy, xx) );
633         Ein = exp ( - ((xx - r0)*(xx - r0) + yy*yy) /
(w0*w0) ); // input field
634         Amev = Amev + Emev*Ein*dx*dy; Amod = Amod +
Emod*Ein*dx*dy; } }
635         Amev = Amev*Amev / (((2.0*omega*Mu0)/Mbeta[m])*
(PI*w0*w0/2.0));
636         Amod = Amod*Amod / (((2.0*omega*Mu0)/Mbeta[m])*
(PI*w0*w0/2.0));
637         if ( matdis == 0 ) { Aplu[m][0] = 100.0* (Amod + Amev); }
638         if ( matdis == 1 ) { Aplu[m][0] = 100.0* (Amod + Amev )
*Spin[ (int)((lamda -lpmin)/dlp) ]; }
639         //printf ("Aplu [%d][0] =%f¥n", modem[m], Aplu [m][0] );
640     } }
641 // Correction for the highest mode group ( elimination of power )

```

```

642     for ( m = 0; m < NLP; m++ ) { if ( modep[m] == Ptotal ) { Aplu[m]
        [0] = 0.0; } }
643     // Total power of input impulse
644     for ( m = 0; m < NLP; m++ ) { A00 = A00 + Aplu[m][0]; } Aw00 =
        Aw00 + A00;
645
646     fprintf ( fp2, "Amplitude of input impulse,A00,%e,¥n", A00 );
647     fprintf ( fp2, "Cumulative amplitude of input impulse,Aw00,%e,¥n
        ¥n", Aw00 );
648     free_dmatrix ( Rlp, 0, 2*Nwkb, 0, N-1 );
649
650
651     /* 出力ファイル */
652     if ( y == Nl/2 || fout == 1 ) {
653         // 時間波形 P
654         fprintf ( fr2, "nstc=%d,", nstd );
655         for ( n = 0; n <= Lmax; n++ ) { fprintf ( fr2, "%f,",
            (double)n*Tv ); } fprintf ( fr2, "pct.¥n" );
656         fprintf ( fr2, "%f,%f¥n", 0.0, A00 );
657         // モードパワー分布 Pm
658         fprintf ( fr3, "myu," ); for ( m = 0; m < NLP; m++ )
            { fprintf ( fr3, "%d,", m ); } fprintf ( fr3, "¥n" );
659         fprintf ( fr3, "LP," ); for ( m = 0; m < NLP; m++ ) { fprintf
            ( fr3, "LP%d_%d,", modep[m], model[m] ); } fprintf ( fr3,
            "¥n" );
660         fprintf ( fr3, "m," ); for ( m = 0; m < NLP; m++ ) { fprintf
            ( fr3, "%d,", modep[m] ); } fprintf ( fr3, "¥n" );
661         fprintf ( fr3, "l," ); for ( m = 0; m < NLP; m++ ) { fprintf
            ( fr3, "%d,", model[m] ); } fprintf ( fr3, "¥n" );
662         fprintf ( fr3, "p," ); for ( m = 0; m < NLP; m++ ) { fprintf
            ( fr3, "%d,", modep[m] ); } fprintf ( fr3, "¥n" );
663         fprintf ( fr3, "%f,", 0.0 );
664         for ( m = 0; m < NLP; m++ ) { for ( n = 0; n <= Lmax; n++ )
            { Pm[m] = Pm[m] + Aplu[m][n]; }
665         fprintf ( fr3, "%f,", Pm[m] ); } fprintf ( fr3, "¥n" );
666         // モード群パワー分布 Pg
667         fprintf ( fr9, ",," ); for ( j = 1; j <= Ptotal; j++ )
            { fprintf ( fr9, "%d,", j ); } fprintf ( fr9, "¥n" ); //
            モード群番号
668         fprintf ( fr9, ",," ); for ( i = 1; i <= Ptotal; i++ ) { count
            =0;
669         for ( myu = 0; myu < NLP; myu++ ) { if ( modep [myu] == i )
            { count = count +1; } } // 縮退数 (モード数)
670         fprintf ( fr9, "%d,", count ); pdeg[i-1]=count; } fprintf
            ( fr9, "¥n" );
671         fprintf ( fr9, "%f,", 0.0 );
672         for ( m = 0; m < NLP; m++ ) { Pg[modep[m]-1] = Pg[modep[m]-1]
            + Pm[m]; } // モード群パワー
673         for ( j = 0; j < Ptotal; j++ ) { fprintf ( fr9, "%f,", Pg
            [j] / (double)pdeg[j] ); Pg[j] = 0.0; } fprintf
            ( fr9, "¥n" );
674     }
675
676     printf("start!¥n");
677     /** 5. 光波伝搬解析 **/
678     /
679     *****
        *****/
680     for ( i = 1; i <= Nz; i++ ) // z(i-1) ~ zi
        {
681
682         /* 相対遅延ステップ数の最大値算出 */
683         Li =0;
684         Li = (int) ( (( taumax - taumin )*( (double)i*dz ) / Tv) +
            0.5 );
685

```



```

686      /* 相対遅延ステップ数のモード分布算出 */
687      for ( m = 0; m < NLP; m++ ) {
688          kim[m] = (int) ( (( Mtau[m] - taumin )*( (double)i*dz ) /
689                          Tv ) + 0.5 )
690                      - (int) ( (( Mtau[m] - taumin )*( (double)
691                          (i-1)*dz ) / Tv ) + 0.5 ); }
692
693 #pragma omp parallel
694 {
695     #pragma omp for
696     /* タイムシフト演算 */
697     for ( m = 0; m < NLP; m++ ) {
698         for ( n = 0; n < kim[m]; n++ ) { Amin[m][n] = 0.0; }
699         for ( n = kim[m]; n <= Li; n++ ) { Amin[m][n] = Aplu[m][n
700             - kim[m]]; }
701     }
702     /* カップリング演算 */
703     for ( m = 0; m < NLP; m++ ) {
704         for ( n = 0; n <= Li; n++ ) { me=0.0;
705             for ( l = 0; l < NLP; l++ ) { me = me + H[m][l]*Amin[l]
706                 [n]; }
707             Aplu[m][n] = me; }
708     }
709     // ① 波長分散を考慮する場合
710     /* 時間波形の重ね合わせ */
711     if ( matdis == 1 && i == Nz ) {
712         // 各波長成分のインパルス応答 (基準時間は非考慮)
713         if ( y == Nl/2 || fout == 1 ) { fprintf ( fr2, "%f, ",
714             (double)i*dz ); }
715         for ( n = 0; n <= Li; n++ ) { ap = 0.0;
716             for ( m = 0; m < NLP; m++ ) { ap = ap + Aplu[m][n]; }
717             if ( y == Nl/2 || fout == 1 ) { fprintf ( fr2, "%f, ",
718                 ap ); }
719             if ( y == Nl/2 || fout == 1 ) { fprintf ( fr2, "\n" ); }
720             // 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
721             // 考慮)
722             if ( y == 0 || nstd == 0 ) { for ( n = 0; n <= Li; n++ )
723                 { for ( m = 0; m < NLP; m++ ) { P[n] = P[n] + Aplu[m]
724                     [n]; } }; }
725             if ( y != 0 && nstd > 0 ) { for ( n = 0; n <= Li; n++ )
726                 { for ( m = 0; m < NLP; m++ ) { P[n+nstd] = P[n+nstd] +
727                     Aplu[m][n]; } }; }
728             if ( y != 0 && nstd < 0 ) {
729                 if ( nstd < nstdmin ) {
730                     for ( n = 0; n <= Pnum[y-1]; n++ ) { P[(Pnum
731                         [y-1]-n)+(nstdmin-nstd)] = P[(Pnum[y-1]-n)]; }
732                     for ( n = 0; n < nstdmin-nstd; n++ ) { P[n] =
733                         0.0; }
734                     for ( n = 0; n <= Li; n++ ) { for ( m = 0; m <
735                         NLP; m++ ) { P[n] = P[n] + Aplu[m][n]; } }
736                     else {
737                         for ( n = 0; n <= Li; n++ ) { for ( m = 0; m <
738                             NLP; m++ ) {
739                             P[n+(nstd-nstdmin)] = P[n+(nstd-nstdmin)] +
740                                 Aplu[m][n]; } }; }
741                     // for ( n = 0; n <= Pnum[y]; n++ ) { fprintf ( fs3, "%
742                         f, ", P[n] ); } fprintf ( fs3, "\n" );
743                 }
744             }
745         }
746     }
747     // ② 波長分散を考慮しない場合
748     /* 計算結果の出力 */
749     if ( matdis == 0 && (i%Nzout == 0 || i == Nz) ) {
750         /* モードパワー分布 Pm */
751         if ( fout == 1 || v == Nl/2 ) { fprintf ( fr3, "%f "

```

```

(double)i*dz ); }
735 for ( m = 0; m < NLP; m++ ) {
736     Pm[m] = 0.0;
737     for ( n = 0; n <= Li; n++ ) { Pm[m] = Pm[m] + Aplu[m]
[n]; }
738     if ( fout == 1 || y == NI/2 ) { fprintf ( fr3, "%f,",
Pm[m] ); }
739     Pg[modep[m]-1] = Pg[modep[m]-1] + Pm[m]; }
740 if ( fout == 1 || y == NI/2 ) { fprintf ( fr3, "¥n"); }
741 /* モード群パワー分布 Pg */
742 if ( fout == 1 || y == NI/2 ) { fprintf ( fr9, "%f,",
(double)i*dz );
743 for ( j = 0; j < Ptotal; j++ ) { fprintf ( fr9, "%f,", Pg
[j] / (double)pdeg[j] ); Pg[j] =0.0;} fprintf
( fr9, "¥n"); }
744 /* インパルス応答 P */
745 if ( fout == 1 || y == NI/2 ) { fprintf ( fr2, "%f,",
(double)i*dz ); }
746 for ( n = 0; n <= Li; n++ ) {
747     P[n] = 0.0;
748     for ( m = 0; m < NLP; m++ ) { P[n] = P[n] + Aplu[m]
[n]; }

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

751 /* 出力波形 */
752 Pout = drealvector ( 0, Ti+Li ); init_realvector ( Pout,
0, Ti+Li );
753 Pout = convolution ( Ti, Pin, Li, P );
754 if ( fout == 1 || y == NI/2 ) {
755     for ( n = 0; n < Ti+Li; n++ ) { fprintf ( fr6, "%f,",
Pout[n] ); } fprintf ( fr6, "¥n"); }
756 /* 周波数応答 M */
757 if ( fout == 1 || y == NI/2 ) { fprintf ( fr4, "%f,",
(double)i*dz ); }
758 for ( j = 0; j <= Nf; j++ ) {
759     Hw = ReHw = ImHw = 0.0;
760     for ( n = 0; n <= Li; n++ ) {
761         ReHw = ReHw + P[n]*cos ( (2.0*PI*(fmin+(double)
j*df))*(double)n*Tv );
762         ImHw = ImHw - P[n]*sin ( (2.0*PI*(fmin+(double)
j*df))*(double)n*Tv ); }
763     M[j] = sqrt ( ReHw*ReHw + ImHw*ImHw ) / A00; if ( y
== NI/2 || fout == 1 ) { fprintf ( fr4, "%f,", -10.0*log10
(1.0 / M[j]) ); } }
764 /* -3dB帯域幅 bw */
765 bw = tav = Ptot = rms = 0.0;
766 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; } }
767 if ( fout == 1 || y == NI/2 ) { fprintf ( fr4, "%f,",
bw ); }
768 /* インパルス応答RMS幅 rms */
769 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;
770 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; }
771 rms = sqrt ( rms - (tav*tav) );
772
773 if ( fout == 1 || y == NI/2 ) { fprintf ( fr4, "%f,%f",

```

```

    rms, tav - ( taumin*(double)(i-1)*dz) ); fprintf ( fr4,
    "¥n" ); }

774
775     fprintf ( fs, "%f,%d,%d,%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 );
776     printf ( "length:%f m ", (double)i*dz ); printf ( "-3db
    bandwidth:%f GHz ", bw ); printf ( "pulse broadening:%f ps
    ¥n", rms );
777 }
778
779 }
780 /
    *****
    *****/

781
782     if ( nstd < nstdmin ) { nstdmin = nstd; }
783     if ( nstd > nstdmax ) { nstdmax = nstd; }
784
785     free_dmatrix ( H, 0, NLP-1, 0, NLP-1 );
786     free_dmatrix ( Amin, 0, NLP-1, 0, Lmax );
787     free_dmatrix ( Aplu, 0, NLP-1, 0, Lmax );
788     free_drealvector ( alpha, 0 );
789     free_dintvector ( kim, 0 );
790     free_drealvector ( Pm, 0 );
791     free_drealvector ( Pg, 0 );
792     free_dintvector ( modem, 0 );
793     free_dintvector ( model, 0 );
794     free_dintvector ( modep, 0 );
795     free_dintvector ( pdeg, 0 );
796     free_drealvector ( Mbeta, 0 );
797     free_drealvector ( Mtau, 0 );
798     printf ( "A00=%f¥n", A00 );
799     printf ( "End¥n¥n");
800
801
802
803 } // 波長ループ終了
804 printf ( "Aw00=%f¥n", Aw00 );
805
806
807
808 /* 計算結果の出力 */
809 if ( matdis == 1 ) {
810     /* インパルス応答波形 (光源スペクトル考慮) Pw */
811     for ( n = 0; n <= Pnum[NI]; n++ ) { fprintf ( fs3, "%f, ", P[n] );
        fprintf ( fs3, "¥n" ); }
812     /* 出力波形 */
813     Pout = drealvector ( 0, Ti+Pnum[NI] ); init_realvector ( Pout, 0, Ti+Pnum
        [NI] );
814     Pout = convolution ( Ti, Pin, Pnum[NI], P );
815     if ( fout == 1 || y == NI/2 ) {
816         for ( n = 0; n < Ti+Pnum[NI]; n++ ) { fprintf ( fs4, "%f, ", Pout
            [n] ); } fprintf ( fs4, "¥n" ); }
817     /* 周波数応答 M */
818     fprintf ( fr4, "%f, ", (double)Nz*dz );
819     for ( j = 0; j <= Nf; j++ ) {
820         ReHw = ImHw = 0.0;
821         for ( n = 0; n <= Pnum[NI]; n++ ) {
822             ReHw = ReHw + P[n]*cos ( (2.0*PI*(fmin+(double)j*df))*(double)
                n*Tv );
823             ImHw = ImHw - P[n]*sin ( (2.0*PI*(fmin+(double)j*df))*(double)
                n*Tv ); }
824         M[j] = sqrt ( ReHw*ReHw + ImHw*ImHw ) / Aw00; fprintf ( fr4, "%f, ",
            -10.0*log10(1.0 / M[j]) ); }
825
826     /* -3dB帯域幅 bw */

```

```

827   bw = tav = Ptot = rms = 0.0;
828   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; }}
829   fprintf ( fr4, "%f,", bw );
830
831   /* インパルス応答RMS幅 rms */
832   for ( n = 0; n <= Pnum[NI]; n++ ) { tav = tav + ( taumin*(double)(i-1)*dz
      + (double)n*Tv)*P[n]*Tv; Ptot = Ptot + P[n]*Tv; } tav = tav / Ptot;

833   for ( n = 0; n <= Pnum[NI]; n++ ) { rms = rms + ( taumin*(double)(i-1)*dz
      + (double)n*Tv)*( taumin*(double)(i-1)*dz + (double)n*Tv )*( P[n] /
      Ptot)*Tv; }
834   rms = sqrt ( rms - (tav*tav) );
835   fprintf ( fr4, "%f,", rms); fprintf ( fr4, "%n" );
836
837   fprintf ( fs, "%f,%f,%f,%f,%f,%f,%f,%f,", g, (double)Nz*dz, rms, bw, tav,
      Ptot, Aw00 );
838   printf ( "length:%f m%n", (double)Nz*dz ); printf ( "-3db bandwidth:%f
      GHz%n", bw ); printf ( "rms width:%f ps%n", rms ); }

839
840
841
842   /* スペックルコントラストの計算 */
843   if ( scc == 1 ) {
844     /* 光源スペクトル自己相関関数 Cp */
845     Cp = dreallocator ( 0, Nf ); init_realvector ( Cp, 0, Nf );
846     for ( j = 0; j <= Nf; j++ ) {
847       for ( i = 0; i <= Nfp; i++ ) {
848         int jj = (int) ( (double)j*df / dfp );
849         Cp[j] = Cp[j] + Spin[Nfp-i]*Spin[ Nfp-i+jj ]; }
850     Cpsum = Cpsum + df*Cp[j]; }
851     // ガウス型スペクトル形状関数
852     // Cp[j] = Cp[j] + exp( -pow( (fpmin + dfp*double(i) - fp0) / fpgw,
      2.0 ))
853     // *exp( -pow( (fpmin + dfp*double(i) -
      df*double(j) - fp0 ) / fpgw, 2.0 ))*dfp; }

854
855     /* スペックルコントラスト spct */
856     fprintf ( fr4, "," );
857     for ( j = 0; j <= Nf; j++ ) {
858       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
859       spct = spct + Cp[j]*M[j]*M[j]*df; } spct = sqrt ( 2.0*spct );
860     fprintf ( fs, "%f%n", spct ); printf ( "speckle contrast:%f%n",
      spct );
861     free_drealvector (Cp, 0); }
862
863
864   free_dintvector ( Pnum, 0 );
865   free_drealvector ( P, 0 );
866   free_drealvector ( Pin, 0 );
867   free_drealvector ( Pout, 0 );
868   free_drealvector ( M, 0 );
869
870
871   fclose ( fp2 ); // BW_setting.csv
872   fclose ( fp3 ); // BW_profile.csv
873   fclose ( fp4 ); // FEM_result.csv
874   fclose ( fq ); // Hmn_result.csv
875   fclose ( fr ); // CPE_Hmatrix.csv
876   fclose ( fr2 ); // CPE_Impulse-responce.csv
877   fclose ( fr3 ); // CPE_Mode-power-distribution.csv
878   fclose ( fr4 ); // CPE_Frequency-response.csv
879   fclose ( fr6 ); // CPE_Output-pulse-waveform.csv
880   fclose ( fs ); // BW_result.csv

```

```

881 fclose (fs2);    // BW_Source-spectrum.csv
882 fclose (fs3);    // BW_Impulse-responce.csv
883 fclose (fs4);    // BW_Output-pulse-waveform.csv
884
885 system("pause");
886 return 0;
887 }
888
889
890
891 /*A.1. 第2種変形ベッセル関数 bessk (n, x) */
892 // 第1種変形Bessel関数 (n=0) I0(x)
893 double bessi0 (double x)
894 {
895     double ax, ans;
896     double y;
897     // Polynomial fit
898     if ( ( ax = fabs(x) ) < 3.75 ) {
899         y = x / 3.75;
900         y*= y;
901         ans = 1.0 + y*(3.5156229 + y*(3.0899424 + y*(1.2067492
902             + y*(0.2659732 + y*(0.360768e-1+y*0.45813e-2)))));}
903     else {
904         y = 3.75 / ax;
905         ans=(exp(ax) / sqrt(ax))*(0.39894228 + y*(0.1328592e-1
906             +y*(0.225319e-2 + y*(-0.157565e-2 + y*(0.916281e-2
907             +y*(-0.2057706e-1 + y*(0.2635537e-1 + y*(-0.1647633e-1
908             +y*0.392377e-2))))))));}
909     return ans;
910 }
911 // 第2種変形Bessel関数 (n=0) K0(x)
912 double bessk0 (double x)
913 {
914     double bessi0 (double x);
915     double y, ans;
916     //polynomial fit
917     if ( x <= 2 ) {
918         y = x*x / 4.0;
919         ans = (-log(x/2.0)*bessi0(x)) + (-0.57721566 + y*(0.42278420
920             + y*(0.23069756 + y*(0.3488590e-1 + y*(0.262698e-2
921             + y*(0.10750e-3 + y*0.74e-5))))));}
922     }
923     else {
924         y=2.0/x;
925         ans = (exp(-x)/sqrt(x))*(1.25331414 + y*(-0.7832358e-1
926             + y*(0.2189568e-1 + y*(-0.1062446e-1 + y*(0.587872e-2
927             + y*(-0.251540e-2 + y*0.53208e-3))))));}
928     }
929     return ans;
930 }
931 // 第1種変形Bessel関数 (n=1) I1(x)
932 double bessi1 (double x)
933 {
934     double ax, ans;
935     double y;
936     if ( ( ax = fabs(x) ) < 3.75 ) {
937         y = x / 3.75;
938         y*=y;
939         ans = ax*(0.5 + y*(0.87890594 + y*(0.51498869 + y*(0.15084934
940             + y*(0.2658733e-1 + y*(0.301532e-2 + y*0.32411e-3))))));}
941     else {
942         y = 3.75 / ax;
943         ans = 0.2282967e-1 + y*(-0.2895312e-1 + y*(0.1787654e-1
944             - y*0.420059e-2));}
945         ans = 0.39894228 + y*(-0.3988024e-1 + y*(-0.362018e-2
946             + y*(0.163801e-2 + y*(-0.1031555e-1 + y*ans)))));}
947         ans*=(exp(ax) / sqrt(ax));}

```

```

948     }
949     return x < 0.0 ? - ans : ans;
950 }
951 // 第2種変形Bessel関数 (n=1) K1 (x)
952 double bessk1 (double x) {
953     double bessi1 (double x);
954     double y, ans;
955     if ( x <= 2.0 ) {
956         y = x*x/4.0;
957         ans = (log(x/2.0)*bessi1(x)) + (1.0/x)*(1.0 + y*(0.15443144
958             + y*(-0.67278579 + y*(-0.18156897 + y*(-0.1919402e-1
959             + y*(-0.110404e-2 + y*(-0.4686e-4)))))););
960     }
961     else {
962         y = 2.0/x;
963         ans = (exp(-x)/sqrt(x))*(1.25331414 + y*(0.23498619
964             + y*(-0.3655620e-1 + y*(0.1504268e-1 + y*(-0.780353e-2
965             + y*(0.325614e-2 + y*(-0.68245e-3)))))););
966     }
967     return ans;
968 }
969 // 第2種変形Bessel関数 Kn(x)
970 double bessk (int n, double x)
971 {
972     double bessnorm = 1.0e7 ;
973     double bessk0 (double x);
974     double bessk1 (double x);
975     int j;
976     double bk, bkm, bkp, tox;
977     if ( n == 0 ) return bessk0 (x) / bessnorm;
978     if ( n == 1 ) return bessk1 (x) / bessnorm;
979     if ( n >= 2 ) {
980         tox = 2.0/x;
981         bkm = bessk0(x) / bessnorm;
982         bk = bessk1(x) / bessnorm;
983         for ( j=1; j<n; j++ ) {
984             bkp = bkm + j*tox*bk;
985             bkm = bk;
986             bk = bkp;
987         }
988         return bk;
989     }
990     else return 0;
991 }
992 /*A.2. 屈折率波長微分関数dndI_var (lamda, n_lamda, mater) */
993 double dndI ( double lamda, double n_lamda, int mater )
994 // { return ( ( 0.01925e-4*lamda - 16.31619e-4 ) * n_lamda + ( -0.02743e-4*lamda +
995 // 23.16674e-4 ) ) * 1.0e9; }
996 { return ( ( 0.02173e-4*lamda - 18.79107e-4 ) * n_lamda + ( -0.03109e-4*lamda +
997 // 26.85035e-4 ) ) * 1.0e9; } // 640 ~ 690 nm
998 /*A.3. 屈折率濃度微分関数dndC_var (lamda, mater) */
999 double dndC ( double lamda, int mater )
1000 { return 2.03716e-9*lamda*lamda - 3.27125e-6*lamda + 2.98314e-3; } // 589 ~
1001 // 690 nm
1002 /*A.3. コア中心屈折率 ncore (lamda, mater) */
1003 double ncore ( double lamda, int mater ) {
1004     double sell; sell = 1.0;
1005     /* DPS-doped PMMA ( 7.8 wt.%, 1.506@589nm ) */
1006     sell = sqrt ( 1.0 + ( 0.41241 / ( 1.0 - ( 22500 / (lamda*lamda) ) ) )
1007         + ( 0.81215 / ( 1.0 - ( 6400 / (lamda*lamda) ) ) )
1008         + ( 0.01117 / ( 1.0 - ( 11560000 / (lamda*lamda) ) ) ) );
1009     /* DPS-doped PMMA ( 9.0 wt.%, 1.506@655nm ) */
1010     // sell = sqrt ( 1.0 + ( 0.61249 / ( 1.0 - ( 8467.04111 / (lamda*lamda) ) ) )
1011     // + ( 0.61872 / ( 1.0 - ( 16525.45233 / (lamda*lamda) ) ) )
1012     // + ( 0.08880 / ( 1.0 - ( 120601870.30580 / (lamda*lamda) ) ) ) );

```



```

1012     return sell; }
1013
1014 /*A.4. クラッド屈折率 nclad (lamda, mater) */
1015 double nclad ( double lamda, int mater ) {
1016     double sell; sell =1.0;
1017     /* PMMA ( 1.492@589nm ) */
1018     sell = sqrt ( 1.0 + ( 0.496284 / (1.0 - ( 5154.872 / (lamda*lamda) )) ) )
1019             + ( 0.6964977 / (1.0 - ( 13802.53 / (lamda*lamda) )) ) )
1020             + ( 0.3223 / (1.0 - ( 85527690 / (lamda*lamda) )) ) ) );
1021     /* DPS-doped PMMA ( 1.079427062 wt. % ) */
1022     // sell = sqrt ( 1.0 + ( 0.5954 / (1.0 - ( 6200.94518 / (lamda*lamda) )) ) )
1023     //             + ( 0.602 / (1.0 - ( 14730.91236 / (lamda*lamda) )) ) )
1024     //             + ( 0.29126 / (1.0 - ( 85685787.87303 / (lamda*lamda) )) ) ) );
1025     return sell; }
1026
1027 /*A.5. 係数行列要素計算関数 S_matrix (a, b, q, m, n, v, w, D)*/
1028 // b[0]=S00, b[1]=S11, ..... b[j]=Sjj ..... b[n]=Snn
1029 // a[0]=___, a[1]=S01, a[2]=S12, ... a[j]=Sj-1,j ... a[n]=Sn-1,n
1030 // a[0]=___, a[1]=S10, a[2]=S21, ... a[j]=Sj,j-1 ... a[n]=Sn,n-1
1031 void S_matrix ( double *a, double *b, double *q, int m, int n, double v,
1032               double w, double D )
1033 {
1034     int j;
1035     if ( m == 0 ) {
1036         b[0] = - (1.0/2.0) + (3.0*q[0]+2.0*q[1])*(v*v/60.0)*((D*D)/(n*n))
1037                 - (1.0/12.0)*(w*w)*((D*D)/(n*n));
1038         b[n] = - ((2.0*n-1.0)/2.0) + ((5.0*n-2.0)*q[n-1]+3.0*(5.0*n-1.0)
1039                 *q[n])*(v*v/60.0)*((D*D)/(n*n)) - ((4.0*n-1.0)/12.0)*(w*w)*
1040                 ((D*D)/(n*n))
1041                 - w*D*bessk(1,w*D)/bessk(0,w*D);
1042     }
1043     else {
1044         b[0] = 0.0;
1045         b[n] = - (1.0-m*m)*((2.0*n-1.0)/2.0) + ((5.0*n-2.0)*q[n-1]+3.0*
1046                 (5.0*n-1.0)*q[n])*(v*v/60.0)*((D*D)/(n*n)) - ((4.0*n-1.0)/12.0)
1047                 *(w*w)*((D*D)/(n*n))
1048                 - (m*m)*((n-1.0)*(n-1.0))*log((double)n/((double)
1049                 n-1.0)) - m*m - w*D*bessk(m-1,w*D)/bessk(m,w*D) - m;
1050         b[1] = - 2.0*(1.0-m*m) + (3.0*q[0]+30*q[1]+7.0*q[2])*(v*v/60.0)*
1051                 ((D*D)/(n*n)) - (2.0/3.0)*(w*w)*((D*D)/(n*n)) - (m*m)*4.0*log
1052                 (2.0);
1053         for ( j = 2; j < n; j++ )
1054         {
1055             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
1056             +2.0)*q[j+1])*(v*v/60.0)*((D*D)/(n*n)) - (2.0/3.0)*j*(w*w)*((D*D)/
1057             (n*n))
1058             - (m*m)*((j-1.0)*(j-1.0))*log((double)j/((double)
1059             j-1.0)) + ((j+1.0)*(j+1.0))*log((double)j+1.0) /
1060             (double)j );
1061         }
1062         a[0] = 0.0; //未使用要素につき0を格納
1063         a[1] = (1.0/2.0) + (2*q[0]+3*q[1])*(v*v/60.0)*((D*D)/(n*n)) -
1064                 (1.0/12.0)*(w*w)*((D*D)/(n*n)) - (m*m)/2;
1065         for ( j = 2; j <= n; j++ )
1066         {
1067             a[j] = ((2.0*j-1.0)/2.0)*(1.0-m*m) + ((5.0*j-3.0)*q[j-1] +
1068             (5.0*j-2.0)*q[j])*(v*v/60.0)*((D*D)/(n*n)) - ((2.0*j-1.0)/12.0)*
1069             (w*w)*((D*D)/(n*n))
1070             + (m*m)*(j-1.0)*j*log((double)j / ((double)j-1.0));
1071         }
1072     }
1073 }
1074
1075 /*A.4. 改訂コレスキー分解 mcholesky ( a, b, ML, MD, m, n )*/
1076 void mcholesky ( double *a, double *b, double *ML, double *MD, int m, int n )
1077 {
1078     int i;
1079     if ( m == 0 ) {
1080         ML[0] = 0.0; //未使用要素につき0を格納
1081         MD[0] = b[0];

```

```

1062     for ( i = 1; i <= n; i++ )
1063     {   MD[i] = b[i] - a[i]*a[i] / MD[i-1];
1064         ML[i] = a[i] / MD[i-1]; }
1065     }
1066     else {
1067         ML[0] = 0.0; //未使用要素につき0を格納
1068         ML[1] = 0.0; //発散するから別扱い. 0で良いのか?
1069         MD[0] = 0.0;
1070         MD[1] = b[1];
1071         for ( i = 2; i <= n; i++ )
1072         {   MD[i] = b[i] - a[i]*a[i] / MD[i-1];
1073             ML[i] = a[i] / MD[i-1]; }
1074     }
1075 }
1076
1077 /*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 )*/
1078 void mcholesky_sol ( double *ML, double *MD, double *R, int m, int n )
1079 {
1080     int i;
1081     // 「Ly=R0」を解く
1082     if ( m==0 ) {
1083         for ( i=1; i <= n; i++ ) {
1084             R[i] = R[i] - R[i-1]*ML[i]; }
1085         // 「(D(LT))R1=y」を 「(LT)R1=(D-1)y=y'」に変える
1086         for ( i=1; i <= n; i++ ) {
1087             R[i] = R[i] / MD[i]; }
1088         // 「(LT)R1=y'」を解く
1089         for ( i=n-1; i >= 0; i-- ) {
1090             R[i] = R[i] - ML[i+1]*R[i+1]; }
1091     }
1092     else{
1093         for ( i=2; i <= n; i++ ) {
1094             R[i] = R[i] - R[i-1]*ML[i]; }
1095         // 「(D(LT))R1=y」を 「(LT)R1=(D-1)y=y'」に変える
1096         for ( i=2; i <= n; i++ ) {
1097             R[i] = R[i] / MD[i]; }
1098         // 「(LT)R1=y'」を解く
1099         for ( i=n-1; i >= 1; i-- ) {
1100             R[i] = R[i] - ML[i+1]*R[i+1]; }
1101     }
1102 }
1103
1104 /*A.6. 逆べき乗法の初期ベクトル計算 R0 ( MD, R, m, n )*/
1105 void R0 ( double *MD, double *R, int m, int n )
1106 /* 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*/
1107 {
1108     int i, j = 1;
1109     if ( m == 0 ) {
1110         for ( i = 0; i <= n-1; i++ ) {
1111             if ( fabs(MD[i+1]) < fabs(MD[j]) ) { j = i + 1; } }
1112     }
1113     else {
1114         for ( i = 1; i <= n-1; i++ ) {
1115             if ( fabs(MD[i+1]) < fabs(MD[j]) ) { j = i + 1; } }
1116     }
1117     //R0の初期値の代入
1118     for ( i = 0; i <= n; i++ ) {
1119         if ( i == j ) { R[i] = 1.0; }
1120         else { R[i] = 0.0; }
1121     }
1122 }
1123
1124 /*A.7. 逆べき乗法の解ベクトル規格化 R_norm ( R, n )*/
1125 void R_norm ( double *R, int n )
1126 {
1127     int i;

```

```

1129     double s = 0;
1130     // 行列要素の2乗和
1131     for ( i = 0; i <= n; i++ ) {   s = s + R[i]*R[i]; }
1132     if ( s != 0 )
1133     {   for ( i = 0; i <= n; i++ ) {   R[i] = R[i] / sqrt(s); }   }
1134 }
1135
1136 /*A.8. 固有値計算 (Rayleigh quotient) Eigen ( R, a, b, m, n)*/
1137 double Eigen ( double *R, double *a, double *b, int n, int m )
1138 {
1139     // Rベクトルを規格化しているため内積は1
1140     int i;
1141     double s=0;
1142     if ( m == 0 ) {
1143         s = ( R[0]*b[0] + R[1]*a[1] ) * R[0];
1144         for ( i = 1; i < n; i++ ) {
1145             s += ( R[i-1]*a[i] + R[i]*b[i] + R[i+1]*a[i+1] ) * R[i]; }
1146         s += ( R[n-1]*a[n] + R[n]*b[n] ) * R[n];
1147         return s;
1148     }
1149     else {
1150         s = ( R[1]*b[1] + R[2]*a[2] ) * R[1];
1151         for ( i = 2; i < n; i++ ) {
1152             s += ( R[i-1]*a[i] + R[i]*b[i] + R[i+1]*a[i+1] ) * R[i]; }
1153         s += ( R[n-1]*a[n] + R[n]*b[n] ) * R[n];
1154         return s;
1155     }
1156 }
1157 /*
1158     s = b[0]*R[0]*R[0];
1159     for ( i = 1; i <= n; i++ ) {
1160         s = s + ( b[i]*R[i]*R[i] + 2.0*a[i]*R[i-1]*R[i] ); }
1161 */
1162
1163 /*A.9. 群遅延計算用関数 dbdk_bunbo ( R, D, w, m, n)*/
1164 /* 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 */
1165
1166 // 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
1167 double dbdk_bunbo ( double *R, double D, double w, int m, int n )
1168 {
1169     int i;
1170     double s=0;
1171     for ( i = 0; i <= n-1; i++ )
1172     { 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] ); }
1173     if ( m == 0 ) {
1174         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); }
1175     else {
1176         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); }
1177 }
1178
1179 /*A.10. 群遅延計算用関数 dbdk_bunshi ( R,q2, D, w, m, n)*/
1180 /* 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 */
1181
1182 // 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
1183 // 屈折率分散パラメータ q2[0]~q2[n] ( = n*(d(kn)/dk) )
1184 double dbdk_bunshi ( double *R, double *q2, double D, double w, int m, int n)
1185 {
1186     int i;
1187     double s=0;
1188     for ( i=0; i<=n-1; i++ )

```

```

1189 { 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]); }
1190 if ( m == 0 ) {
1191     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); }
1192 else {
1193     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); }
1194 }
1195
1196 /* A.13. 整数ベクトル領域確保用関数 dvector (i, j) */
1197 int *dintvector ( int i, int j ) {
1198     int *a;
1199     if ( ( a = (int *) malloc ( (j -i+1)*sizeof (int) ) ) == NULL )
1200         { printf ("Memory cannot be allocated !¥n"); exit (1); }
1201     return (a-i); }
1202
1203 /* A.14. 整数ベクトル領域解放用関数 free_dvector (a, i) */
1204 void free_dintvector ( int *a, int i ) {
1205     free ( (void*) (a+i) ); }
1206
1207 /* A.15. 実数ベクトル領域確保用関数 dvector (i, j) */
1208 double *drealvector ( int i, int j ) {
1209     double *a;
1210     if ( ( a = (double *) malloc ( (j -i+1)*sizeof (double) ) ) == NULL )
1211         { printf ("Memory cannot be allocated !¥n"); exit (1); }
1212     return (a-i); }
1213
1214 /* A.16. 実数ベクトル領域解放用関数 free_dvector (a, i) */
1215 void free_drealvector ( double *a, int i ) {
1216     free ( (void*) (a+i) ); }
1217
1218 /* A.17. 実数行列領域確保用関数 dmatrix (nr1, nr2, nl1, nl2) */
1219 double **dmatrix ( int nr1, int nr2, int nl1, int nl2 ) {
1220     // nrow: 行の数, ncol: 列の数
1221     double **a;
1222     int i, nrow, ncol;
1223     nrow = nr2 - nr1 + 1;
1224     ncol = nl2 - nl1 + 1;
1225     /* 行の確保 */
1226     if ( ( a = (double **) malloc ( nrow*sizeof (double*) ) ) == NULL )
1227         { printf ("Memory cannot be allocated !¥n"); exit (1); }
1228     a = a - nr1; // 行をずらす
1229     /* 列の確保 */
1230     for ( i = nr1; i <= nr2; i++ ) a[i] = (double *) malloc (ncol*sizeof
      (double) );
1231     for ( i = nr1; i <= nr2; i++ ) a[i] = a[i] - nl1; // 列をずらす
1232     return (a); }
1233
1234 /* A.18. 実数行列領域解放用関数 free_dmatrix ( a, nr1, nr2, nl1, nl2) */
1235 void free_dmatrix ( double **a, int nr1, int nr2, int nl1, int nl2 ) {
1236     int i;
1237     for ( i = nr1; i <= nr2; i++ ) free ( (void*) (a[i] + nl1) );
1238     free ( (void*) (a+nr1) );
1239 }
1240
1241 /* A.19. 整数ベクトル初期化関数 init_vector ( a, nr1, nr2 ) */
1242 void init_intvector ( int *a, int nr1, int nr2 ) {
1243     int i;
1244     for ( i = nr1; i <= nr2; i++ ) { a[i] = 0; }
1245 }
1246
1247 /* A.20. 整数ベクトル初期化関数 init_vector ( a, nr1, nr2 ) */
1248 void init_realvector ( double *a, int nr1, int nr2 ) {
1249     int i;

```

```
1250     for ( i = nr1; i <= nr2; i++ ) { a[i] = 0.0; }
1251 }
1252
1253 /* A.21. 実数行列初期化関数 init_vector ( a, nr1, nr2, nl1, nl2 ) */
1254 void init_realmatrix (double **a, int nr1, int nr2, int nl1, int nl2 ) {
1255     for ( int i = nr1; i <= nr2; i++ ) {
1256         for ( int j = nl1; j <= nl2; j++ ) { a[i][j] = 0.0; } }
1257 }
1258
1259 /* 畳み込み積分 convolution (n1, P1, n2, P2) */
1260 double* convolution ( int n1, double* P1, int n2, double* P2 ) {
1261     int i, j;
1262     double* R;
1263     // R = (double*) malloc ( sizeof (double) *(n1+n2+1));
1264     if ( ( R = (double *) malloc ( (n1+n2+1)*sizeof (double) ) ) == NULL )
1265     { printf ("Memory cannot be allocated !¥n"); exit (1); }
1266     for ( i=0; i<=n1+n2; i++ ) { R[i] = 0.0; }
1267     for ( i=0; i<n1; i++ ) {
1268         for( j=0; j<=n2; j++ ) { R[i+j] = R[i+j] + P1[i]*P2[j]; } }
1269     //for( j=0; j<=n2; j++ ) { R[i+j]+=P1[i]*P2[j]; } }
1270     return R; }
1271
```