Université de Limoges
Faculté des Sciences
Master Procédés et Matériaux

Année 2006-2007

Travaux Pratiques - Langage C - Option

```
1  /************************************************
2   *      Traitement de données 2 - Option         *
3   *        Master Procédés et Matériaux           *
4   *                                               *
5   *    Étude numérique de la construction d'une   *
6   *      onde lumineuse dans une cavité laser     *
7   *          à Transformation de Fourier          *
8   *                                               *
9   *             Ludovic Grossard                  *
10  *            grossard@unilim.fr                 *
11  *      Année universitaire 2004-2005            *
12  *                                               *
13  ************************************************/
14
15
16  /************************************************
17   *            Fichiers d'en-tête                 *
18   ************************************************/
19
20  #include <stdio.h>
21  #include <sys/time.h>
22  #include <gsl/gsl_math.h>
23  #include <gsl/gsl_vector.h>
24  #include <gsl/gsl_rng.h>
25  #include <gsl/gsl_complex.h>
26  #include <gsl/gsl_complex_math.h>
27  #include <gsl/gsl_matrix.h>
28  #include <gsl/gsl_fft_complex.h>
29  #define N 64
30  #define ITERATIONS 60
31  #define NB_TRACES 10
32
33  /************************************************
34   *          Prototypes des fonctions             *
35   ************************************************/
36
37  void initialise_bruit(gsl_vector_complex* vecteur);
38  void filtrage( gsl_vector_complex* champ, gsl_vector_complex* filtre);
39  void normalisation( gsl_vector_complex* champ );
40  void gain( gsl_vector_complex* champ );
41  void pertes( gsl_vector_complex* champ , double coeff);
```

```
42 void exporte_colonne_module(gsl_vector_complex* vecteur , gsl_matrix* matrice , in
43 void exporte_colonne_phase(gsl_vector_complex* vecteur , gsl_matrix* matrice , int
44 void enregistre_matrice(gsl_matrix* matrice , gsl_vector* x , char* fichier );
45 void TF_directe(gsl_vector_complex* champ );
46 void TF_inverse(gsl_vector_complex* champ );
47 void initialise_x( gsl_vector* x , double plage , double pixel );
48 void initialise_filtre1( gsl_vector_complex* filtre1);
49 void initialise_filtre2( gsl_vector_complex* filtre2);
50
51 /***********************************************
52  *              Fonction principale            *
53  ***********************************************/
54
55
56 int main(void)
57 {
58
59     /* définition des variables scaaires */
60
61     double lambda = 1064e-6;
62     double f = 500;
63     double DeltaX1 = 2;
64     double dx1 = DeltaX1/N;
65     double dx2 = ( lambda * f ) / ( N * dx1 );
66     double DeltaX2 = N * dx2;
67     double R2 = 0.90;
68
69     int i;
70
71     /* vecteurs contenant le champ électromagnétique et les filtres */
72
73     gsl_vector_complex *champ = gsl_vector_complex_calloc( N );
74     gsl_vector_complex *filtre1 = gsl_vector_complex_calloc( N );
75     gsl_vector_complex *filtre2 = gsl_vector_complex_calloc( N );
76
77     /* vecteurs contenant les coordonnées spatiales
78        au niveau des miroirs */
79
80     gsl_vector *x1 = gsl_vector_calloc( N );
81     gsl_vector *x2 = gsl_vector_calloc( N );
82
83     /* matrices contenant le module pour toutes les itérations */
84
85     gsl_matrix *module1 = gsl_matrix_calloc(N,ITERATIONS);
86     gsl_matrix *module2 = gsl_matrix_calloc(N,ITERATIONS);
87
88     /* matrices contenant la phase pour toutes les itérations */
89
90     gsl_matrix *phase1 = gsl_matrix_calloc(N,ITERATIONS);
91     gsl_matrix *phase2 = gsl_matrix_calloc(N,ITERATIONS);
92
93
94     /* Initialisations */
95
96     initialise_x(x1 , DeltaX1 , dx1);
```

```
97      initialise_x(x2 , DeltaX2 , dx2);
98      initialise_filtre1(filtre1);
99      initialise_filtre2(filtre2);
100     initialise_bruit( champ );
101
102
103     /* début de la boucle */
104
105     for(i=0;i<ITERATIONS;i++){
106
107         filtrage( champ , filtre1 );
108         gain( champ );
109         TF_directe( champ );
110
111         exporte_colonne_module( champ , module2 , i );
112         exporte_colonne_phase( champ , phase2 , i );
113
114         filtrage( champ , filtre2 );
115         pertes( champ , R2 );
116         TF_inverse( champ );
117
118         gain( champ );
119
120         exporte_colonne_module( champ , module1 , i );
121         exporte_colonne_phase( champ , phase1 , i );
122
123     }
124     /* fin de la boucle */
125
126
127     /* Enregistrement du résultat */
128
129     enregistre_matrice( module1 , x1 , "module1.txt");
130     enregistre_matrice( module2 , x2 , "module2.txt");
131     enregistre_matrice( phase1  , x1 , "phase1.txt");
132     enregistre_matrice( phase2  , x2 , "phase2.txt");
133 }
134
135 /************************************************
136  *                                              *
137  *             F O N C T I O N S                *
138  *                                              *
139  ************************************************/
140
141
142
143
144 /************************************************
145  *        initialisation du bruit               *
146  ************************************************/
147
148 void initialise_bruit(gsl_vector_complex* vecteur)
149 {
150     int i;
151     gsl_complex z;
```

```
152
153     struct timeval tv;
154     struct timezone tz;
155     gettimeofday(&tv, &tz);
156
157     gsl_rng_default_seed=tv.tv_sec*1000000+tv.tv_usec;
158
159     gsl_rng * r;
160     const gsl_rng_type * T;
161     T = gsl_rng_default;
162     r = gsl_rng_alloc( T );
163
164     for(i=0;i<N;i++){
165         z = gsl_complex_rect(0.01*gsl_rng_uniform( r ),0.01*gsl_rng_uniform( r ));
166         gsl_vector_complex_set( vecteur , i , z);}
167     gsl_rng_free( r );
168 }
169
170
171 /***********************************************
172  *             Filtrage du champ               *
173  ***********************************************/
174
175 void filtrage( gsl_vector_complex* champ, gsl_vector_complex* filtre)
176 {
177     double module,phase;
178     gsl_complex z;
179     int i;
180
181     for(i=0;i<N;i++){
182         module = gsl_complex_abs(gsl_vector_complex_get( champ , i));
183         phase = gsl_complex_arg(gsl_vector_complex_get( champ , i));
184         module *= GSL_REAL(gsl_vector_complex_get(filtre,i));
185         z = gsl_complex_polar(module,phase);
186         gsl_vector_complex_set(champ,i,z);
187     }
188 }
189
190
191 /***********************************************
192  *        normalisation d'un vecteur           *
193  ***********************************************/
194
195 void normalisation( gsl_vector_complex* champ )
196 {
197     int i;
198     double maxi=0;
199     double module,phase;
200     gsl_complex z;
201
202     for(i=0;i<N;i++){
203         module = gsl_complex_abs(gsl_vector_complex_get( champ , i));
204         if(module>maxi){maxi = module;}
205     }
206
```

```
207       for(i=0;i<N;i++){
208           module = gsl_complex_abs(gsl_vector_complex_get( champ , i))/maxi;
209           phase  = gsl_complex_arg(gsl_vector_complex_get( champ , i));
210           z = gsl_complex_polar( module , phase);
211           gsl_vector_complex_set( champ , i , z );
212       }
213   }
214
215
216   /***********************************************
217    * exporter le champ dans une colonne de matrice *
218    *                    module                     *
219    ***********************************************/
220
221   void exporte_colonne_module(gsl_vector_complex* vecteur , gsl_matrix* matrice , in
222   {
223       int j;
224       double module;
225
226       for(j=0;j<N;j++){
227           module = gsl_complex_abs(gsl_vector_complex_get( vecteur , j));
228           gsl_matrix_set( matrice , j , i , module);
229       }
230   }
231
232
233   /***********************************************
234    * exporter le champ dans une colonne de matrice *
235    *                     phase                     *
236    ***********************************************/
237
238   void exporte_colonne_phase(gsl_vector_complex* vecteur , gsl_matrix* matrice , int
239   {
240       int j;
241       double phase;
242
243       for(j=0;j<N;j++){
244           phase = gsl_complex_arg(gsl_vector_complex_get( vecteur , j));
245           gsl_matrix_set( matrice , j , i , phase);
246       }
247   }
248
249
250   /***********************************************
251    *        enregistrement d'une matrice           *
252    *              dans un fichier                  *
253    ***********************************************/
254
255   void enregistre_matrice(gsl_matrix* matrice , gsl_vector* x , char* fichier )
256   {
257       FILE* f;
258       int i,j;
259       f = fopen( fichier , "w");
260       for(j=0;j<N;j++){
261           for(i=0;i<ITERATIONS;i=i+ceil(ITERATIONS/NB_TRACES)){
```

```
262         fprintf(f,"%f\t%d\t%f\n",gsl_vector_get(x,j),i,
263                 pow(gsl_matrix_get( matrice , j , i ),2));
264         }
265         fprintf(f,"\n");
266     }
267     fclose(f);
268 }
269
270
271 /***********************************************
272  *        Transformée de Fourier directe       *
273  ***********************************************/
274
275 void TF_directe(gsl_vector_complex* champ )
276 {
277     gsl_fft_complex_radix2_forward (champ->data , 1 , N);
278 }
279
280
281 /***********************************************
282  *        Transformée de Fourier inverse       *
283  ***********************************************/
284
285 void TF_inverse(gsl_vector_complex* champ )
286 {
287     gsl_fft_complex_radix2_inverse (champ->data , 1 , N);
288 }
289
290
291 /***********************************************
292  *          Gain du milieu amplificateur       *
293  ***********************************************/
294
295 void gain( gsl_vector_complex* champ )
296 {
297     int i;
298     double module,phase,G;
299     gsl_complex z;
300     double A = 1.0;
301     double B = 10.0;
302
303     for(i=0;i<N;i++){
304         module = gsl_complex_abs(gsl_vector_complex_get( champ , i));
305         phase  = gsl_complex_arg(gsl_vector_complex_get( champ , i));
306         G = exp(A/(1.0+B*module));
307         module*=G;
308         z = gsl_complex_polar( module , phase);
309         gsl_vector_complex_set( champ , i , z );
310     }
311 }
312
313
314 /***********************************************
315  *        Pertes dues au miroir de sortie      *
316  ***********************************************/
```

```
317
318 void pertes( gsl_vector_complex* champ , double R)
319 {
320     int i;
321     double module,phase;
322     gsl_complex z;
323
324     for(i=0;i<N;i++){
325         module = gsl_complex_abs(gsl_vector_complex_get( champ , i));
326         phase  = gsl_complex_arg(gsl_vector_complex_get( champ , i));
327         module*=sqrt(R);
328         z = gsl_complex_polar( module , phase);
329         gsl_vector_complex_set( champ , i , z );
330     }
331 }
332
333
334 /***********************************************
335  *        Initialisation du vecteur x           *
336  ***********************************************/
337
338 void initialise_x( gsl_vector* x , double plage , double pixel )
339 {
340     int i;
341     for(i=0;i<N;i++)
342         gsl_vector_set( x , i , -plage / 2 + i * pixel );
343 }
344
345 /***********************************************
346  *          Initialisation du filtre 1          *
347  ***********************************************/
348
349 void initialise_filtre1( gsl_vector_complex* filtre1)
350 {
351     int i;
352     gsl_complex z;
353     for(i=1+0.5*(N-1.4*ceil(sqrt(N)));i<0.5*(N+1.4*ceil(sqrt(N)));i++){
354         z = gsl_complex_polar(1.0,0.0);
355         gsl_vector_complex_set(filtre1,i,z);
356     }
357 }
358
359
360 /***********************************************
361  *          Initialisation du filtre 2          *
362  ***********************************************/
363
364 void initialise_filtre2( gsl_vector_complex* filtre2)
365 {
366     int i;
367     gsl_complex z;
368     for(i=1+0.5*(N-1.4*ceil(sqrt(N)));i<0.5*(N+1.4*ceil(sqrt(N)));i++){
369         z = gsl_complex_polar(1.0,0.0);
370         gsl_vector_complex_set(filtre2,i,z);
371     }
```

```
372     }
```