Université de Limoges
Faculté des Sciences
Master Procédés et Matériaux

Année 2005-2006

Travaux Dirigés - Langage C
Série 5

L'objectif de la séance de TD est de vous faire écrire les fonctions manquantes du programme de simulation numérique de la construction d'une onde lumineuse dans une cavité Laser.

1 Partie 1 : conditions initiales et filtrage

1.1 Initialisation du bruit

Le prototype de la fonction est le suivant :

```
void initialise_bruit(gsl_vector_complex* vecteur)
```

Dans cette fonction, vous devez créer et initialiser un générateur de nombres aléatoires. La graine sera déterminée à partir de l'horloge de l'ordinateur au moment ou la fonction est appelée. Vous utiliserez la fonction `gettimeofday` déjà étudiée au premier semestre. Afin de remplir le vecteur, créez un nombre complexe `z` et placez un nombre aléatoire compris entre 0 et 1 dans ses parties réelles et imaginaires. N'oubliez pas de libérer la mémoire utilisée par le générateur de nombres aléatoires.

1.2 Filtrage

Le prototype de la fonction est le suivant :

```
void filtrage( gsl_vector_complex* champ, gsl_vector_complex* filtre)
```

`champ` est le vecteur contenant le champ électrique. `filtre` est le vecteur contenant la transmittance complexe du filtre. Son module correspond à une atténuation de l'amplitude du champ, et sa phase (nulle dans votre cas) correspond à un éventuel déphasage de l'onde par le filtre. Dans cette fonction, multipliez le module de `champ` par le module de `filtre`. Attention, ces deux vecteurs complexes sont donnés avec leurs parties réelles et imaginaires.

1.3 Initialisation du filtre 1

Le prototype de la fonction est le suivant :

```
void initialise_filtre1( gsl_vector_complex* filtre1)
```

Remplissez le vecteur `filtre1` en tenant compte des contraintes vues en cours

1.4 Initialisation du filtre 2

Le prototype de la fonction est le suivant :

```
void initialise_filtre2( gsl_vector_complex* filtre2)
```

Même question que précédemment

2 Partie 2 : gain, pertes et transformées de Fourier

2.1 Modélisation du gain

Le prototype de la fonction est le suivant :

```
void gain( gsl_vector_complex* champ )
```

Multipliez le module du vecteur `champ` par un gain défini par la relation :

$$G = \exp\left(\frac{A}{1 + B \times \text{module}}\right)$$

avec A = 1 et B = 10

2.2 Modélisation des pertes

Le prototype de la fonction est le suivant :

```
void pertes( gsl_vector_complex* champ , double R)
```

Multipliez le module du vecteur `champ` par le coefficient de réflexion en amplitude du miroir ($\sqrt{R}$).

2.3 Fonction transformée de Fourier

Le prototype de la fonction est le suivant :

```
void TF_directe(gsl_vector_complex* champ )
```

Écrivez une fonction qui effectue la transformée de Fourier directe du vecteur qui lui est passé en paramètre.

2.4 Fonction transformée de Fourier inverse

Le prototype de la fonction est le suivant :

```
void TF_inverse(gsl_vector_complex* champ )
```

Écrivez une fonction qui effectue la transformée de Fourier inverse (avec normalisation) du vecteur qui lui est passé en paramètre.

3 Partie 3 : Manipulation et enregistrement des données

3.1 Placement d'un vecteur dans une colonne de matrice : module

Le prototype de la fonction est le suivant :

```
void exporte_colonne_module(gsl_vector_complex* vecteur , gsl_matrix* matrice , int i )
```

L'objectif de cette fonction est de placer le module du vecteur `vecteur` dans la colonne i de la matrice `matrice`.

3.2 Placement d'un vecteur dans une colonne de matrice : phase

Le prototype de la fonction est le suivant :

```
void exporte_colonne_phase(gsl_vector_complex* vecteur , gsl_matrix* matrice , int i )
```

L'objectif de cette fonction est de placer la phase du vecteur `vecteur` dans la colonne i de la matrice `matrice`.

3.3 Enregistrement d'une matrice dans un fichier

Le prototype de la fonction est le suivant :

```
void enregistre_matrice(gsl_matrix* matrice , gsl_vector* x , char* fichier )
```

Écrivez une fonction qui écrit le contenu de la matrice `matrice` dans un fichier dont le nom se trouve dans la variable `fichier`. Les données doivent être écrites au format gnuplot, où chaque ligne correspond à un point de l'espace (coordonnées x, y, z), comme le montre l'exemple ci-dessous :

```
-1.000000   0   0.000001
-1.000000   6   0.001516
-1.000000   12  0.007213
-1.000000   18  0.013249
-1.000000   24  0.017771
-1.000000   30  0.020898
-1.000000   36  0.022991
-1.000000   42  0.024367
-1.000000   48  0.025263
-1.000000   54  0.025842

-0.968750   0   0.000001
-0.968750   6   0.002466
-0.968750   12  0.005816
-0.968750   18  0.010335
-0.968750   24  0.013881
-0.968750   30  0.016346
-0.968750   36  0.017995
-0.968750   42  0.019079
-0.968750   48  0.019783
-0.968750   54  0.020239
...
```

Parmi les `ITERATIONS` colonnes de la matrice, seules `NB_TRACES` seront enregistrées dans le fichier.

3.4 Initialisation des coordonnées spatiales transverses

Le prototype de la fonction est le suivant :

```
void initialise_x( gsl_vector* x , double plage , double pixel )
```

Le but de cette fonction est de remplir le vecteur `x` de `-plage/2` à `+plage/2 - 1` par pas de `pixel`. Ce vecteur correspond aux dimensions physiques dans les plans des deux miroirs.