Compte Rendu TP Méthode Numérique TP n°1 : Polynômes

Da Costa Tom, Mottino Loris, Verrier Martin

Lien du projet Git sur GitHub : https://github.com/iogepryga/MN_TP1.git

Introduction :

Lors de ce TP nous allons coder des fonctions de manipulation de polynômes. Nous allons considérer deux types de polynômes différents, les polynômes pleins, où tous les degrés du polynôme (meme ceux avec coefficient nul) seront conservés dans le polynôme, et les polynômes creux, où on ne conservera pas les degrés avec coefficient nul.

Polynômes pleins :

Nous avons choisi une implémentation par tableau, afin de stocker les coefficients des polynômes, et une structure contenant un tableau de float pour les coefficients et un entier pour le degré du polynôme.

Nous allons détailler notre code pour quelques fonctions clefs :

creer_polynome :

L'objectif de cette fonction est de retourné un polynôme de degré indiqué en paramètre de la fonction. Pour cela nous allouons dynamiquement un espace en mémoire pour la structure du polynôme p que nous voulons créer, à l'aide de la fonction malloc de la bibliothèque c stdlib.h. Cette allocation dynamique est importante car elle nous permet ensuite de gérer la libération de cette mémoire plus facilement, à l'aide de la fonction free(). Par ailleurs, nous allouons dynamiquement le tableau qui doit contenir les coefficients du polynôme. L'allocation dynamique du tableau permet, en plus de la gestion de la mémoire, d'initialiser le tableau en fonction d'un paramètre, ici le degré passé en paramètre.

lire_polynome_float :

Dans cette fonction nous voulons lire un fichier contenant un polynôme afin de l'afficher. Pour cela nous allons avoir besoin de plusieurs choses :

La structure FILE déclarée dans la bibliothèque stdio.h, qui nous permet de stocker les informations relatives à un flux de donnée.

Nous aurons ensuite besoin d'une fonction pour ouvrir un flux de donné et stocker les informations relatives dans la structure FILE. Pour cela nous avons choisi d'utiliser la fonction fopen, qui retourne NULL si l'ouverture du fichier donné en paramètre échoue, ou bien retourne les informations du flux de donné du fichier dans un structure FILE.

Il nous faudra ensuite extraire les coefficients du polynôme présent dans le fichier, ainsi que le degré du polynôme. Nous allons donc utiliser la fonction fscanf qui prend un paramètre une structure FILE et stock l'information extraire à une adresse passé en paramètre. Afin d'extraire les données la fonction va utiliser une méthode de formatage. Afin de préciser le format à extraire nous passons en paramètre de la fonction une chaine de charactère. Nous allons utiliser ici %d car nous souhaitons extraire des valeurs entières correspondent aux coefficients du polynôme. Si la fonction s'exécute correctement alors elle retournera 1.

Enfin nous aurons besoin de la fonction fclose afin de fermer le flux ouvert par fopen.

addition_polynome :

Dans cette fonction nous souhaitons additionner deux polynômes et retourner le polynôme résultat.

Pour faire cela nous allons créer un nouveau polynôme avec la fonction créer_polynome qui sera le polynôme résultat à retourner. Nous appelons cette fonction avec en paramètre le degré maximum présent dans le polynôme 1 ou 2 à additionner.

Ensuite nous nous additionnons les coefficients des polynômes entre eux en fonction de leur degré associé.

Polynômes creux :

Comme pour les polynômes pleins nous utilisons une structure pour définir le polynôme et, ici, deux tableaux pour stocker les coefficients du polynôme, mais aussi ses degrés correspondants aux coefficients. Nous utilisons également un entier que l'on nommera nbMonome qui nous indiquera le nombre de monôme du polynôme creux.

Le fonctionnement et raisonnement derrière les fonctions creer_polynome, lire_polynome_float et addition_polynome sont quasiment identiques à celles des polynômes creux.

On va donc détailler pour les polynômes creux la fonction inserer_monome qui est plus intéressante.

Inserer_monome :

Le but de cette fonction est d'insérer un nouveau monôme à un polynôme existant.

Si le polynôme creux est vide nous initialisons donc 2 nouveaux tableaux avec malloc afin de stocker le degrés et coefficient du nouveau et unique monôme du polynôme.

Sinon nous copions temporairement les deux tableau degré et coeff du polynôme passé en paramètre. Puis allouons 2 nouveaux tableaux de taille p -> nbMonome + 1 et recopions les anciens tableaux degré et coeff dans les nouveaux.

Il ne nous reste alors plus qu'a libérer les copies des tableaux temporaire et ajouter les coefficients et degré du nouveau monôme à la fin des 2 tableaux du polynôme.

Performances :

Afin de comparer les implémentations de polynômes creux et polynômes pleins nous codons une fonction, perf_poly et perf_poly_creux.

Nous avons ensuite testé nos programmes à l'aide de quatre polynômes p1, p2, p4 et p5 et obtenu les résultats suivants :

```
Performance polynômes pleins p1 et p2 :


p1 = 2.000000 + 0.000000*X + 3.000000*X^2 + 4.000000*X^3
p2 = 0.000000 + 0.000000*X + 1.000000*X^2 + 2.000000*X^3 + 3.000000*X^4 + 4.000000*X^5 + 5.000000*X^6
p3 = 2.000000 + 0.000000*X + 4.000000*X^2 + 6.000000*X^3 + 3.000000*X^4 + 4.000000*X^5 + 5.000000*X^6
addition 530 cycles
p1+p2 4 operations 0.019623 GFLOP/s
addition 17521 cycles
p4+p5 1025 operations 0.152103 GFLOP/s


Performance polynômes creux p1 et p2 :


p1 = 2.000000 + 3.000000 X^2 + 4.000000 X^3
p2 = 1.000000 X^2 + 2.000000 X^3 + 3.000000 X^4 + 4.000000 X^5 + 5.000000 X^6
p3 = 2.000000 + 4.000000 X^2 + 6.000000 X^3 + 3.000000 X^4 + 4.000000 X^5 + 5.000000 X^6
addition 3418 cycles
p1+p2 2 operations 0.001521 GFLOP/s
addition 45867 cycles
p4+p5 1024 operations 0.058046 GFLOP/s
```

Note : ces tests ont été réalisé sur un ordinateur avec processeur Intel Core i5 2.5GHz de 2 cœurs, sous machine virtuel.

Observations :

On remarque que les polynômes creux utilisent plus de cycles pour effectuer les additions que les polynômes pleins.

Cependant le nombre d'opérations effectué par les polynômes creux lors de l'addition et légèrement inférieur au nombre effectué par les polynômes pleins. Le temps d'exécutions pour les polynômes creux étant aussi plus court.

Tests :

Afin de mener à bien ce TP et de s'assurer du bon fonctionnement de notre code nous avons fait deux programmes de tests : test_poly.c et tes_poly_creux.c.

Nous avons utilisé ces programmes, couplé avec de nombreux polynômes (creux et pleins) de tests.

Tous les polynômes que nous avons utilisé, ainsi que les programmes sont présents dans le projet sur Github.

Conclusion :

Lors de ce TP nous avons appris à faire des choix d'implémentations, et à comprendre que ces choix ont un impact direct sur les performances de notre programme.

En effet, un polynôme creux et un polynôme plein sont mathématiquement identique, mais pour l'ordinateur leur traitement sont différents et leur traitement nécessite des ressources différents (meme si légère au vu de nos tests).

Ce TP fut également l'occasion d'améliorer notre travail en équipe et l'utilisation d'un projet git.