Programmation Parallèle et Distribuée
TD 2

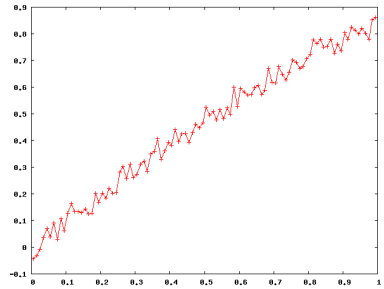
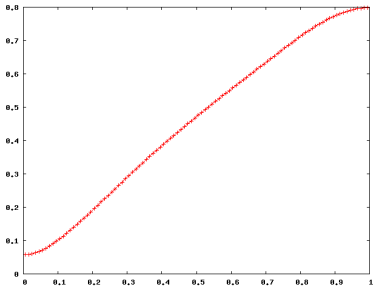
Exercice : Lissage d'une courbe

On considère une courbe « bruitée » représentée par n points, où n est strictement positif.
Soit x_i la i-ème valeur de cette courbe.
Une opération de lissage consiste à construire une courbe lissée $(y_i)_{1<=i<=n}$ de la façon suivante :

$y_1 = (x_1 + x_2) / 2, \quad y_n = (x_n + x_{n-1}) / 2$

$y_i = (x_{i-1} + x_i + x_{i+1}) / 3,$ pour $1<i<n$

Pour rendre la courbe de moins en moins bruitée, on itère l'opération KMAX fois (voir algorithme en annexe).

Exemple d'une courbe bruitée (à gauche), et de sa courbe lissée après KMAX itérations (à droite)

Récupérer le fichier lissage1D.tar et se déplacer dans le répertoire créé.

- Compiler le programme séquentiel : ./comp_seq sequentiel.c
- Executer le programme séquentiel : ./run_seq
- Pour compiler un programme MPI : ./comp_par *<fichier.c>* *<nom_executable>*
- Pour exécuter le programme MPI : ./run_par *<nom_executable>* *<nb_de_proc>*

On demande de paralléliser le programme de lissage sur un nombre quelconque de processeurs, et ceci à l'aide de MPI. Pour ce faire, répondre aux questions suivantes.

Répartition du travail sur *P* processeurs : compléter la fonction init_info()

On choisit le modèle de programmation SPMD.
Pour ce faire, on décide de faire du parallélisme en espace (i.e. selon la variable i de notre programme). On parle alors de parallélisme de domaine, ou bien de décomposition en sous-domaines.

1. Partager l'intervalle *[1, n]* en *P* sous-intervalles disjoints de tailles semblables. Pour chaque processus MPI, déterminer en fonction du rang et de *P* :
   - le nombre de points à traiter par le processus info->nloc ;
   - les indices de début et de fin de boucles info->ideb et info->ifin en fonction du rang et de *P*.

2. Pourquoi est il important que les intervalles aient des tailles semblables ?

Parallélisme de tâches ***(décomposition en sous-domaines)***

Dans cette première étape, chaque processus MPI travaille sur des tableaux x et y de tailles n+2, mais effectue les calculs uniquement sur leurs indices compris entre ideb (inclu) et ifin (exclu).

3. Modifier la fonction ecrire_fichier pour qu'elle fonctionne en parallèle (seul le processus 0 écrit l'ensemble des résultats de tous les processus)

4. Paralléliser la fonction bords pour obtenir le même résultat qu'en séquentiel.

5. Quelle est la mémoire totale des tableaux x et y sur tous les *P* processus MPI (on demande un ordre de grandeur en fonction de *n* et de *P*) ? Comparer la avec la mémoire prise par un seul processus ? Qu'en déduisez vous ?

Parallélisme de tâches et de données ***(parallélisme de domaine avec recouvrement)***

Dans cette seconde étape, chaque processus MPI effectue toujours les calculs sur les indices compris entre ideb et ifin, mais les tailles des tableaux x et y sont de nloc+2 (où nloc = ifin-ideb+1).

Chaque processsus va opérer sur sa propre numérotation locale (entre 1 et nloc inclus) tout en faisant le lien avec la numérotation globale (i.e. L'intervalle [1, nloc+1[en numérotation locale correspond à l'intervalle [ideb, ifin[en numérotation globale).

6. Modifier la fonction lire_fichier pour que chaque processus MPI récupère sa part de travail.

7. Modifier la fonction ecrire_fichier.

8. Réécrire le programme principal pour qu'il fonctionne avec cette nouvelle approche. Comparer l'écriture de ce programme MPI avec celle du programme séquentiel.

9. Donner un ordre de grandeur de la mémoire totale des tableaux x et y en fonction de *P* et de n. Comparer la avec la mémoire en séquentiel.

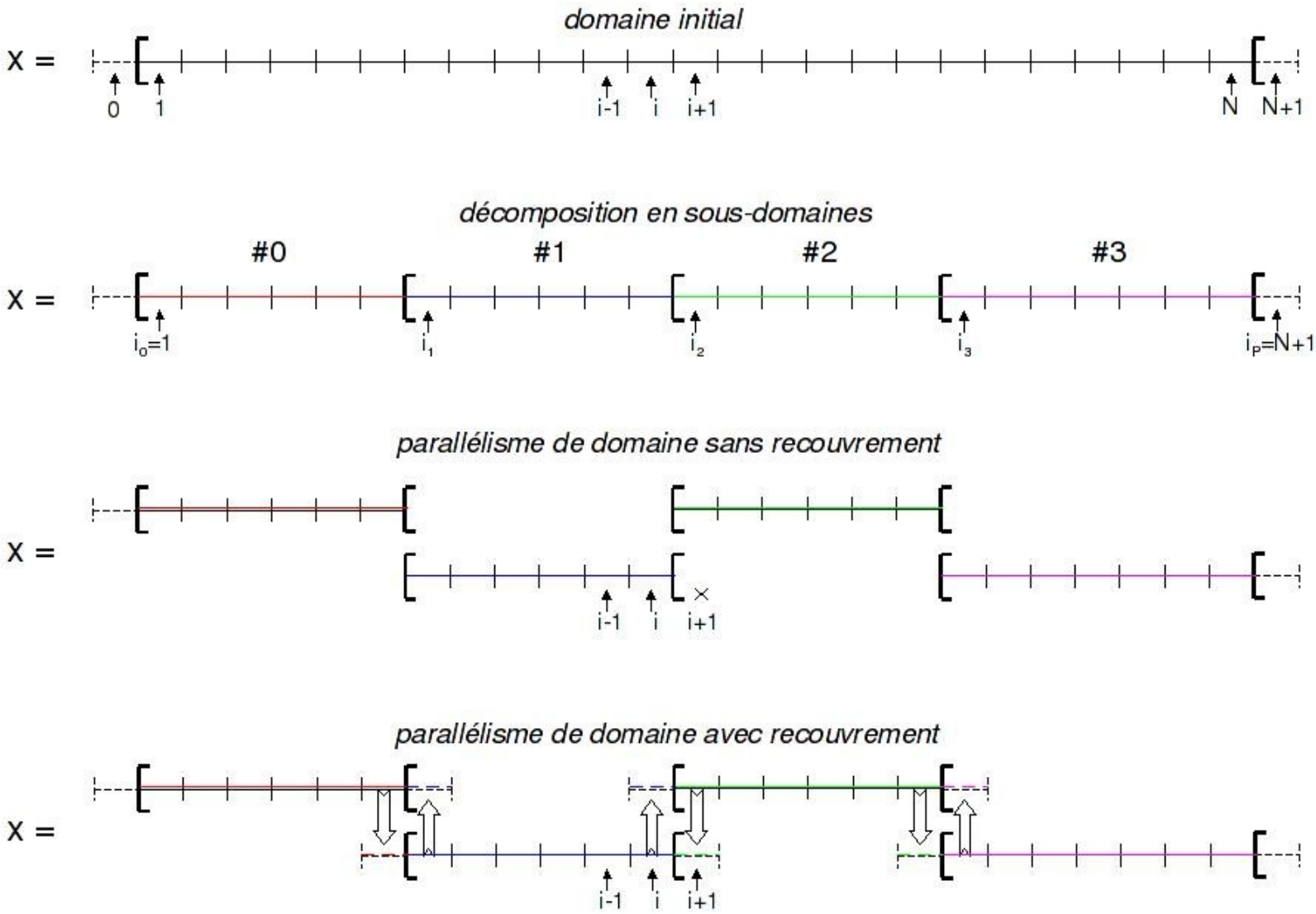
Annexe

Programme C séquentiel de lissage d'une courbe

```
1.  int main(int argc, char **argv) {
2.      int n = 100;    const double pas = 1./n;
3.      double *x = (double*)malloc((n+2)*sizeof(double));
4.      double *y = (double*)malloc((n+2)*sizeof(double));
5.      lire_fichier(« donnees », x, n) ;
6.      ecrire_fichier("avant_lissage.txt", x, n, pas);
7.      double *x0 = x, *x1 = y, *tmp;
8.      /* KMAX itérations de lissage */
9.      for( int k = 0 ; k < KMAX ; k++ ) {
10.         /* bords */
11.         x0[0] = (x0[1]+x0[2]) / 2;   x0[n+1] = (x0[n]+x0[n-1]) / 2;
12.         /* lissage */
13.         for( int i = 1 ; i <= n ; i++ )
14.             x1[i] = (x0[i-1] + x0[i] + x0[i+1]) / 3;
15.         /* passage à l'itération suivante */
16.         tmp = x0 ;   x0 = x1 ;  x1 = tmp ;
17.     }
18.     ecrire_fichier("apres_lissage.txt", y, n, pas);
19.     free(x);  free(y);
20.     return 0 ;
21. }
22. void ecrire_fichier(const char *nom, const double *x, const int n,
    const double pas)
23. {
24.     FILE *fd = fopen(nom, "w");
25.     for( int i = 1 ; i <= n ; i++ )
26.         fprintf(fd, "%.6e %.6e\n", i*pas-0.5*pas, x[i]);
27.     fclose(fd);
28. }
29. void lire_fichier(const char *nom, double *x, const int n)
30. {
31.     FILE *fd = fopen(nom, « r ») ;
32.     for( int i = 1 ; i <= n ; i++ )
33.         lire_ligne(fd, &(x[i]) ) ;
34.     fclose(fd) ;
35. }
```

Extrait du fichier donnees

```
-4.388460e-002
-3.128810e-002
-8.475810e-003
3.597094e-002
6.976661e-002
3.931828e-002
9.032808e-002
3.012532e-002
1.057275e-001
6.203704e-002
1.275321e-001
1.639456e-001
1.344114e-001
1.349855e-001
1.296777e-001
1.441338e-001
1.257529e-001
1.272560e-001
2.017856e-001
1.677274e-001
2.013165e-001
…
7.787793e-001
8.239885e-001
8.132157e-001
8.002262e-001
8.212316e-001
8.024460e-001
7.786495e-001
8.512851e-001
8.603883e-001
```


domaine initial
X =
0
1
i-1
i
i+1
N
N+1
décomposition en sous-domaines
X =
#0
#1
#2
#3
i0=1
i1
i2
i3
iP=N+1
parallélisme de domaine sans recouvrement
X =
i-1
i
i+1
parallélisme de domaine avec recouvrement
X =
i-1
i
i+1