NOTES TECHNIQUES

SCIENCES DE LA TERRE

GÉOLOGIE-GÉOPHYSIQUE

N° 24

2002

Détection automatique des séismes de la station sismologique de Dzumac

Pierre LEBELLEGARD
Catherine BALDASSARI
Robert PILLET
Marc RÉGNIER

Institut de recherche
pour le développement

/Lebellegard, P.
/Baldassari, C.
/Pillet, R.
/Régnier, M.

Détection automatique des séismes de la station sismologique de Dzumac

Nouméa : IRD. Août 2002. 51 p.
Notes Tech. : Sci. Terre ; Géol.-Géophys. ; 24

AUTOMATISATION ; SURVEILLANCE ; SEISME ; ESSAI NUCLEAIRE / NOUVELLE CALEDONIE ; PROVINCE SUD ; DZUMAC

Imprimé par la Service de Reprographie
Centre IRD de Nouméa
Août 2002

I – Introduction

Installée par le Département Analyse et Surveillance de l 'Environnement (DASE), du Commissariat à l'Energie Atomique (CEA) dans le cadre du Traité d'Interdiction Complète des Essais Nucléaires [4] (CTBT, voir : http://www.ctbto.org/), la station sismologique du Mont Dzumac est opérationnelle depuis juillet 2000. Cette station fait partie du réseau sismique SSI[5], dont le but est de détecter les explosions nucléaires souterraines. La localisation des stations de ce réseau est la suivante :

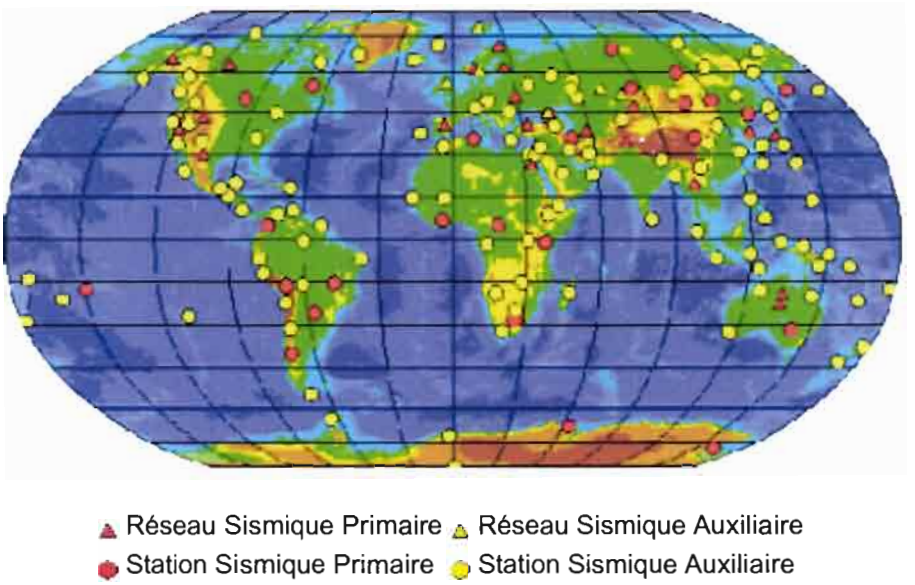

Fig. I-1 : Le réseau sismique SSI

La station de Dzumac fonctionne en continu, et fournit les données en trois composantes Z, N, E (verticale, Nord-Sud, et Est-Ouest). La station ayant été installée

---

[4] Extrait de http://www.can-ndc.nrcan.gc.ca/index_f.php:
Traité d'interdiction complète des essais nucléaires (CTBT)
Le Traité d'interdiction complète des essais nucléaires (CTBT) comprendra un réseau de 321 stations mondiales et fournira un régime global de vérification une système global de transmission, un centre de données international et inspection sur place pour vérification de conformité. Pour assurer la conformité dans tous les environnements - sous terre, sous eau, et dans l'atmosphère, des plateformes de capteurs incorporent quatre technologies différentes seront développées et déployées. Ces technologies comporteront des systèmes sismique (des vibrations acoustiques dans la terre), hydroacoustique (des vibrations dans l'océan), infrasoniques (des ondes dans l'atmosphère), et radionucléide (particules radioactives dans l'air). Le Canada supporte chacune des quatre technologies. L'information plus détaillée sur les divers systèmes de surveillance peut être obtenue à partir de la Commission préparatoire (du CTBTO) à Vienne, Autriche.

[5] **Le réseau sismique du SSI**
Le réseau sismique du SSI sera la technologie principale pour la vérification de CTBT. Son but principal est de détecter, de localiser et d'identifier les explosions nucléaires souterraines. Le réseau sismique primaire est composé de 50 stations sismiques qui fournissent des données en temps réel au CID. Ce réseau primaire sera complété par 120 stations auxiliaires, à partir desquelles les données peuvent être rendues disponibles sur demande. Les données des stations auxiliaires sont principalement utilisées pour améliorer la précision de localisation des événements sismiques détectés par le réseau primaire. Le CID pourrait demander les données du réseau auxiliaire lorsque c'est nécessaire pour la détermination précise d'un épicentre.

directement par le DASE, elle apparaît comme une « boîte noire » , où l'accès aux données n'est ni direct, ni en temps réel. Il n'est pas a priori non plus question d'installer sur le PC d'acquisition d'autres logiciels que celui de pilotage de la station, développé et installé par le DASE. Le but de la présente note technique est de détailler d'une part, la synchronisation des données entre ce PC purement dédié à l'acquisition, et un PC plus spécifiquement orienté vers le traitement (détection des séismes) et la gestion des données sismiques, et d'autre part, le logiciel de décodage et de détection automatique des événements sismiques. La synchronisation doit se faire dans le délai le plus court possible par rapport à la cadence d'acquisition ; le traitement doit permettre au-delà de la détection automatique, le dépouillement manuel (par exemple, le marquage des temps d'arrivée), et l'exportation vers d'autres formats, pour permettre la diffusion des données et des événements détectés (formats SAC, SEED).

II – Synchronisation des données acquises

Sur le PC d'acquisition, on n'a pas accès au données en temps réel, comme ce pourrait être le cas par l'intermédiaire d'un « ring buffer ». Au contraire, les données sont fournies sous forme de fichiers recouvrant un intervalle de temps de une demi-heure (un fichier par composante). En outre les données ne sont pas directement accessibles telles quelles: le programme ***sigchk*** fourni par le DASE (l'exécutable seul est fourni) permet de connaître les informations principales d'un fichier (entre autres : instants de début et de fin, type de la composante, fréquence d'échantillonnage, unité de mesure, sensibilité, compression, etc.) ; ensuite, le programme ***sigconv*** fourni par le DASE (l'exécutable seul est fourni), donne la liste des valeurs numériques du fichier sous forme de valeurs ASCII (une valeur par ligne).

```
Fichier C:\DASE\2002-217\19\CAL11050.00c :
- station                          : CAL
- voie                             : CP-Z
- fréquence                        : 50 Hz
- date de début (incluse) (dmy)    : 05/08/2002 19:10:50.000
- date de fin   (exclue)  (dmy)    : 05/08/2002 19:40:50.000
- durée                            : 1800000 ms soit 90000 échantillons
- unité des mesures                : nm/s
- sensibilité                      : 0.212 unités/échantillon
- nature des mesures               : long
- nombre d'échantillons par bloc   : 256
- version du format                : 102
- ordre des octets                 : Intel
- compression (100%=aucun gain)    : 29%
- informations utilisateur         : CAL1|X25Usat
Lecture ok.
```

Fig. II-1 : Exemple de sortie *sigchk*

La seule manière à notre disposition pour accéder aux informations essentielles d'un fichier est de dépouiller ce type de sortie ASCII.

On dispose donc des données dans un délai compris entre 0 et 30 minutes. Les données sont rangées dans des répertoires horaires. Le fait que les données d'un tel répertoire soient prêtes pour le dépouillement est déterminé par la présence d'un fichier témoin (vide), appelé OPLCOMP.STP. La structure générale des données sur le PC d'acquisition est la suivante :

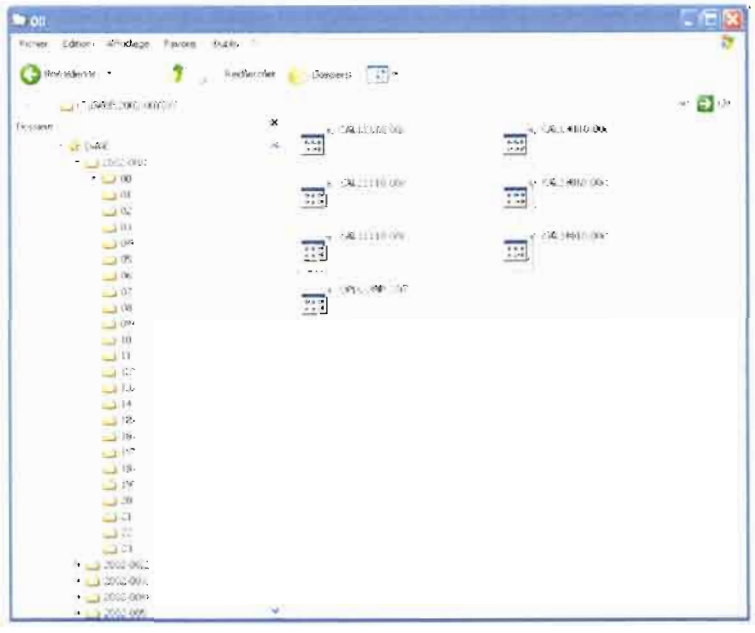

Fig. II-2 : L'arborescence DASE

La composante est repérée par le chiffre qui suit immédiatement les trois caractères de l'acronyme (1, 2, 3 correspondant à Z, N, E). Quant aux 4 caractères qui suivent, ils déterminent l'instant de début des données par rapport à l'heure considérée. Ainsi le fichier 2002-001\00\CAL11110.00c contient les données correspondant à la composante verticale, 1er janvier 2002, à 00h11'10''. Toutes les dates sont bien entendu des dates UTC.

Le programme de synchronisation examinera donc cette arborescence à intervalles périodiques, afin de déterminer s'il existe des fichiers à transférer (c'est-à-dire non encore présents sur le PC de traitement). Afin de ne pas perturber l'acquisition, il est impératif que cet examen soit le plus bref possible. L'arborescence étant une structure qui augmente continuellement avec le temps, il a été décidé de n'en examiner les fichiers que sur une période limitée, typiquement deux semaines. Cela permet de pallier des coupures éventuelles de la liaison avec le PC de traitement, tout en limitant le nombre de répertoires à examiner. Ce sont en effet les E/S disque qui rallongent la durée d'exécution. Le schéma fonctionnel de l'ensemble de la chaîne est le suivant :

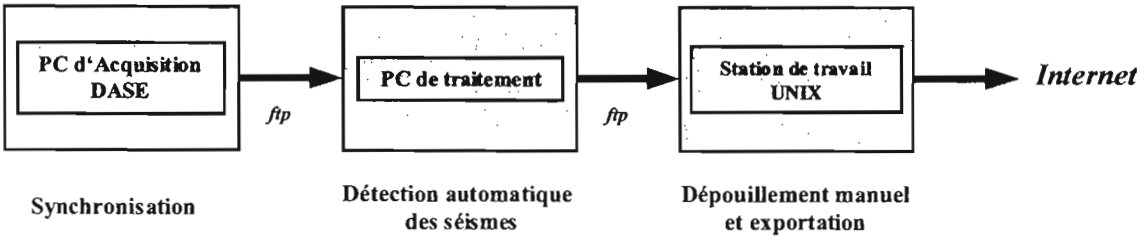

Fig. II-3 : Schéma fonctionnel

Le transfert effectif des données est réalisé par ftp, il faut que la machine « destinataire » soit capable de traiter des requêtes ftp entrantes, c'est-à-dire qu'elle agisse en serveur ftp. Dans la mise en œuvre présente, les transferts se font entre les différentes machines à travers le réseau Ethernet de l'IRD Nouméa, mais il est important de noter que dans le cas de machines séparées éloignées géographiquement, **ce schéma fonctionnel reste identique**. On peut imaginer par exemple, que le PC d'acquisition se situant sur le terrain, près d'une station sismologique, l'initiation d'un transfert ftp déclenche une connexion internet (RTC, ADSL...) vers le site de traitement, par exemple le Centre IRD de Nouméa.

Les machines UNIX agissent naturellement en serveur ftp sans ajout supplémentaire de logiciel ; pour ce qui est du PC de traitement, le logiciel ftp Serv-U y a été installé. Quelle que soit la plate-forme d'installation, la syntaxe des commandes ftp est standard. Le test de présence des fichiers de données sur le PC de traitement est réalisé par le biais des commandes ftp.

III – Détection automatique des séismes

III.1 – Principe de la détection automatique :

Lors de l'examen d'un signal sismique, on cherche a caractériser les différents fronts d'onde, et plus particulièrement les différents instants d'arrivée :

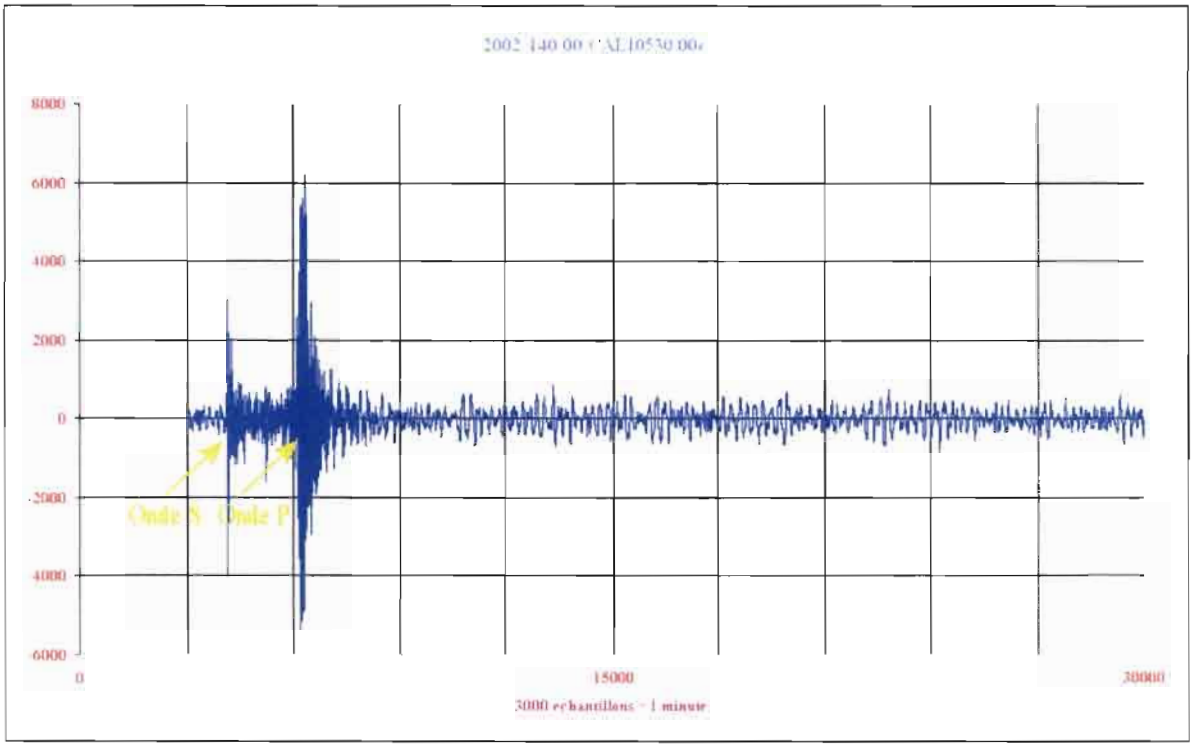

Fig. III-1 : Exemple de signal sismique

Dans cet exemple typique, extraire l'information pertinente revient à déterminer les instants d'arrivée des ondes S et P, le délai entre les deux valeurs permettant de préciser l'éloignement du foyer du séisme. Ce dépouillement est jusqu'à présent réalisé manuellement (« piquage des temps d'arrivée »), et réaliser une détection automatique consiste à élaborer un programme de traitement du signal sismique qui sépare le séisme proprement dit du « bruit de fond ».

III.2 – Caractérisation du signal : rapports STA et LTA

De manière courante, on considère que le « bruit de fond », c'est-à-dire le signal hors impulsion, est caractérisé de manière satisfaisante par une moyenne du signal redressé (la valeur absolue) sur un « long » intervalle de temps, de l'ordre de une minute. C'est le « Long Term Average » , LTA en abrégé. Parallèlement, une valeur instantanée est caractérisée par le « Short Term Average », STA en abrégé, qui représente la moyenne du signal redressé sur une « courte » période, de l'ordre de la seconde. Ainsi, à tout point du signal sismique à l'instant t, à la valeur de l'échantillon proprement dite S(t), on peut associer deux valeurs supplémentaires, STA(t) et LTA(t) :

$$\mathbf{STA(t_0)} = 1/STA \sum_{t = t_0 - STA}^{t = t_0} |S(t)| \quad (1)$$

$$\mathbf{LTA(t_0)} = 1/LTA \sum_{t = t_0 - LTA}^{t = t_0} |S(t)| \quad (2)$$

Les valeurs STA et LTA représentent le nombre d'échantillons de l'intervalle considéré : pour un signal échantillonné à 50 Hz, STA représente une moyenne sur 50 échantillons, et LTA une moyenne sur 3000 échantillons.

D'un point de vue algorithmique, en passant de l'instant t à l'instant $t+1$, on ne réeffectue pas la sommation sur STA ou LTA points : on a mémorisé les sommes Σ_{STA} et Σ_{LTA} , à laquelle on rajoute abs(S(t+1)), et desquelles on retire abs(S(t-STA)), et abs(S(t-LTA)), respectivement. A chaque itération, on effectue seulement deux additions et une division (cette denière pouvant être évitée, si l'aspect temps de calcul est critique, comme ce pourrait être le cas pour des détections temps réel). On obtient :

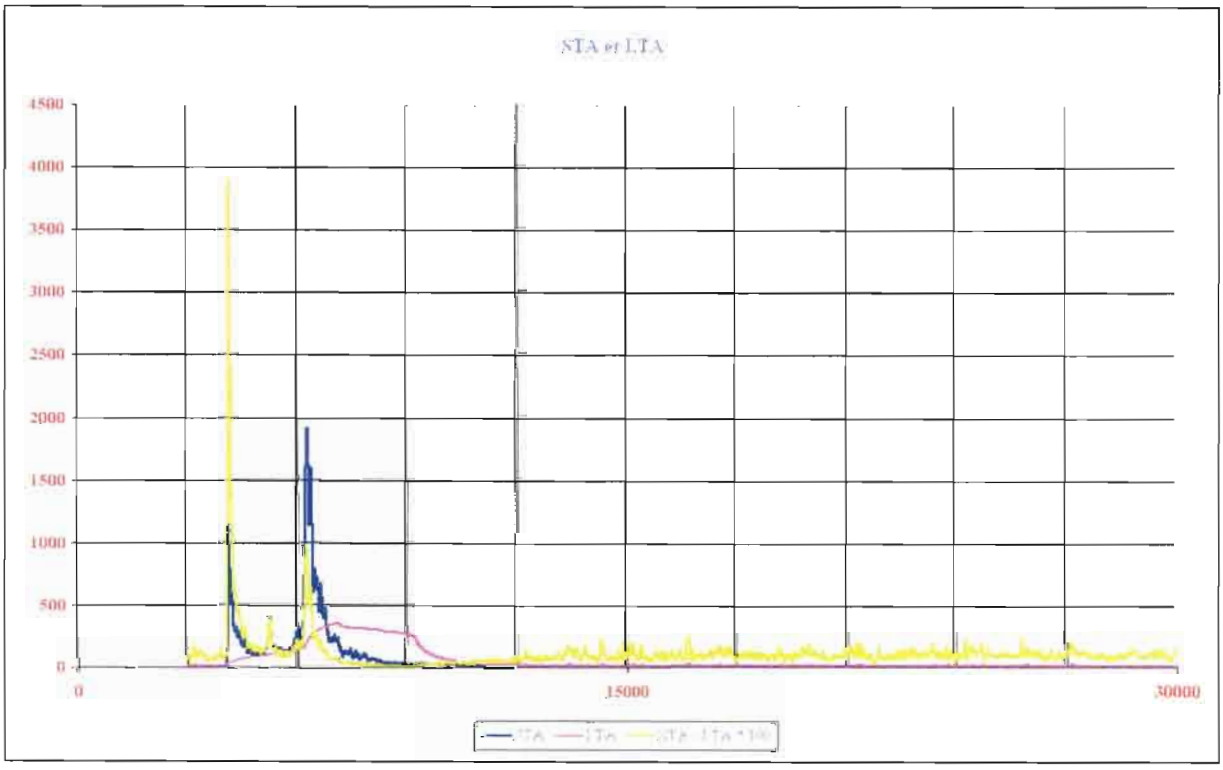

Fig. III-2 : Variation des moyennes STA et LTA le long du signal

On constate notamment que hors événement sismique, les deux moyennes STA et LTA sont sensiblement égales, c'est-à-dire que leur rapport est voisin de 1.

III.3 – Détermination automatique des séismes : rapport STA/LTA

On déduit de ce qui précède que une brusque augmentation du rapport STA/LTA témoigne d'un événement sismique, rapport d'autant plus élevé qu'on est proche de l'événement, et que le rapport à long terme LTA n'a pas encore intégré suffisament de

points postérieurs à l'événement. Déterminer un évenement sismique est alors aisé : on calculera pour chaque échantillon le rapport entre les deux moyennes STA et LTA, et un événement sera détecté lorsque ce rapport franchira un seuil fixé a priori. On utilise généralement des seuils de l'ordre de 3 à 5 ; il faut garder à l'esprit qu'un rapport trop faible entraîne de nombreuses « fausses » détections : une impulsion parasite — de très courte durée — suffit alors au déclenchement. Dans le cas de la station de Dzumac, ce seuil a été fixé à 5.5, et la détection est faite sur la composante verticale (composante Z).

III.4 – Nécessité d'un filtrage : mise en place d'un filtre récursif

Malheureusement dans la pratique, il ne suffit pas de trouver la valeur adéquate du rapport STA/LTA : trop faible, elle entraîne trop de « fausses » détections, et trop élevée, les séismes sont de moins en moins bien détectés. Il faut donc éliminer les impulsions trop brèves, autrement dit effectuer un filtrage passe-bas avant de lancer l'algorithme de détection. Le type de filtre mis en œuvre dans le cas présent a une importance relative, nous avons choisi un filtre récursif qui présente l'avantage d'être facile à mettre en œuvre, et peu gourmand en temps de calcul. On pourra consulter [Serge Castan, Jun Chen][6] pour plus de détails. Le principe de ce filtre récursif est que la réponse en un point est fonction de la valeur du point, et de la réponse au point précédent ; l'effet du filtre ne dépend que d'un seul coefficient flottant, compris entre 0 et 1 :

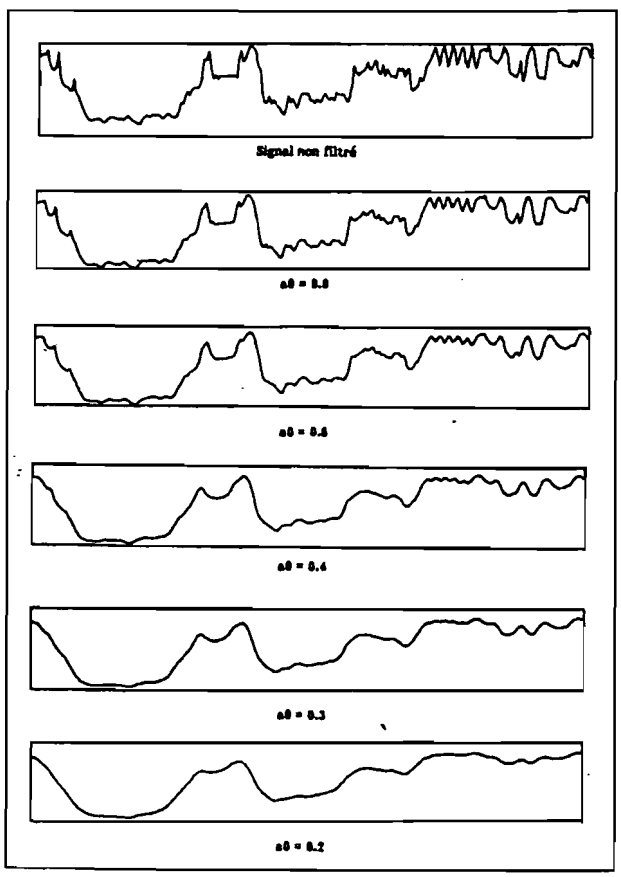

Fig. III-3 : Coefficient du filtre récursif

[6] Un nouvel algorithme de détection de contours, Congrès AFCET/RFIA, Grenoble, 1985, Tome 1, pp. 201 à 213]

Le filtre passe-haut est déduit du filtre passe-bas par une soustraction du signal initial, les algorithmes sont donnés ici ; on en trouvera le texte complet en annexe 2 :

```
/*----- Filtre passe-bas recursif -----*/
p_b_rec (double *val, int cont, double a0)
{
        double    *val2;
        int               i;

<_>

        /* Premier passage gauche/droite */
        *val2 = *val;
        for (i=1; i<cont; i++)
                val2[i]=val2[i-1]+a0*(val[i]-val2[i-1]);

        /* Second passage droite/gauche */
        val[cont-1] = val2[cont-1];
        for (i=cont-1; i>=0; --i)
                val[i] = val[i+1]+a0*(val2[i]-val[i+1]);

<_>

        /* On retourne le nombre de valeurs du signal filtre */
        return (cont);
}
```

Fig. III-4 : Filtre passe-bas récursif

```
/*----- Filtre passe-haut recursif -----*/
int p_h_rec(double *val, int cont, double a0)
{
        double    *val2, *val3;
        int               i;

<_>

        /* Premier passage gauche/droite */
        *val2 = *val;
        for (i=1; i<cont; i++)
                val2[i]=val2[i-1]+a0*(val[i]-val2[i-1]);

        /* Second passage droite/gauche */
        val3[cont-1] = val2[cont-1];
        for (i=cont-1; i>=0; --i)
                val3[i] = val3[i+1]+a0*(val2[i]-val3[i+1]);

        /* On retranche le signal filtre du signal initial */
        for (i=0; i<cont; i++)
                val[i] -= val3[i];

        /* On retourne le nombre de valeurs du signal filtre */
        return (cont);
}
```

Fig. III-5 : Filtre passe-haut récursif

Un exemple d'application de ce filtre passe-bas (a0 = 0.25) est donné ici, l'instant d'arrivée apparaît nettement :

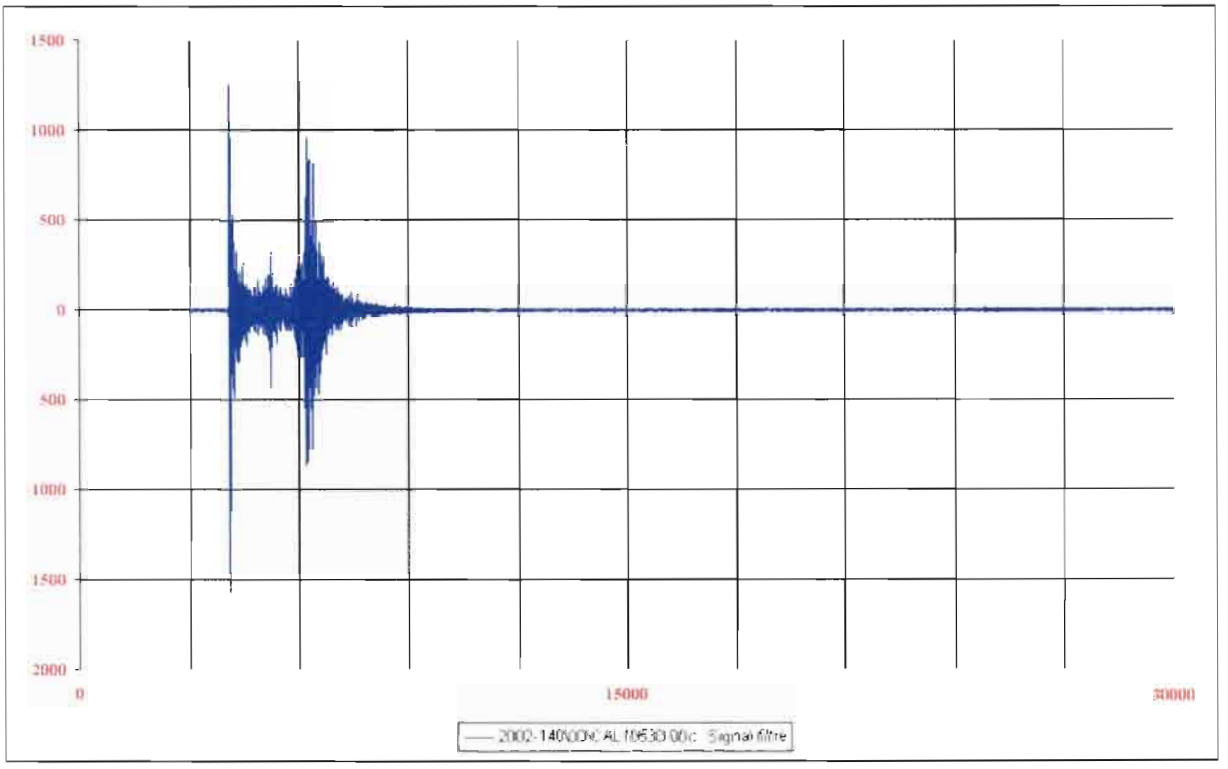

Fig. III-6 : Filtrage récursif passe-bas

III.5 – Conclusion

Une fois le séisme détecté, pour s'assurer de ne pas perdre d'information, on rajoute au signal extrait, une durée de pré-événement de une minute, et une durée de post-événement de trois minutes. Le programme détecte même des séismes qui n'apparaissent pas à l'écran, compte tenu de la moindre résolution de l'écran ; il faut alors utiliser un filtre pour la mise en évidence et permettre un marquage manuel. Le séisme extrait est mise à disposition aux formats Sismalp et Sac, et bientôt SEED.

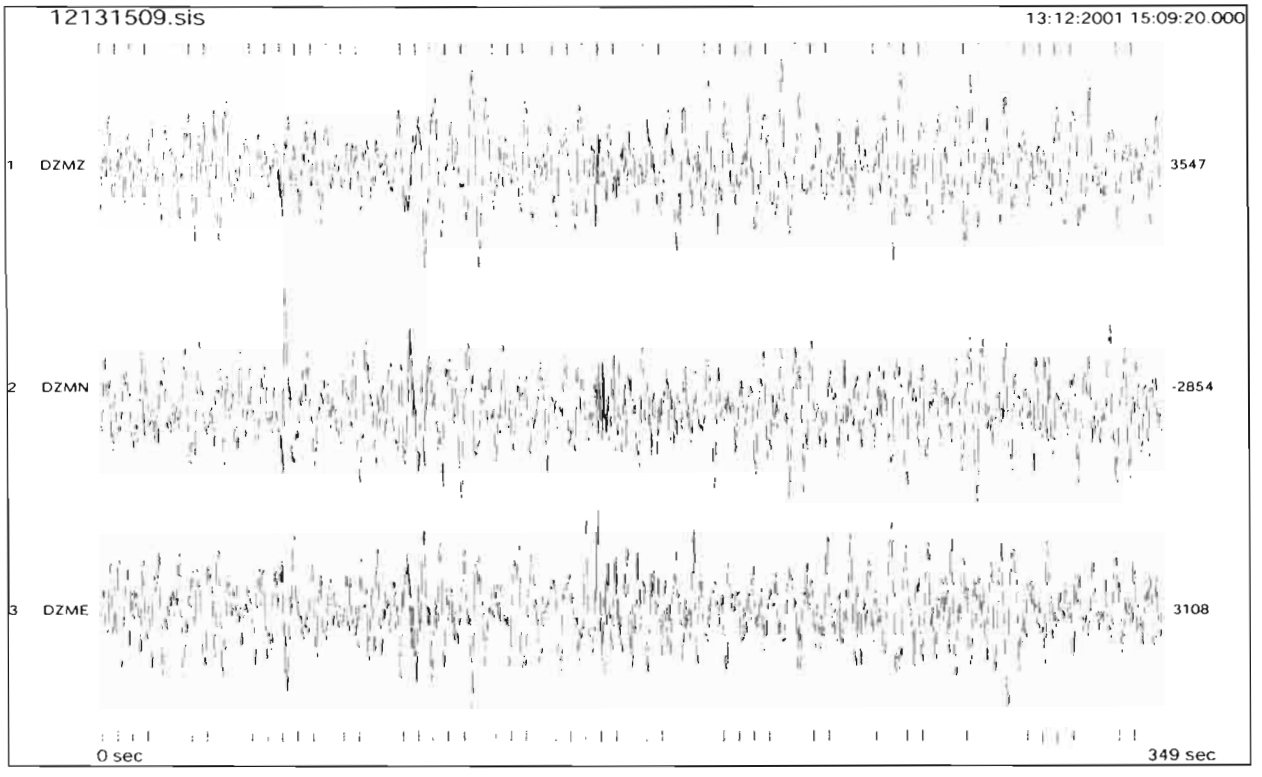

Fig. III-7 : Séisme détecté (13 décembre 2001, 15 :09)

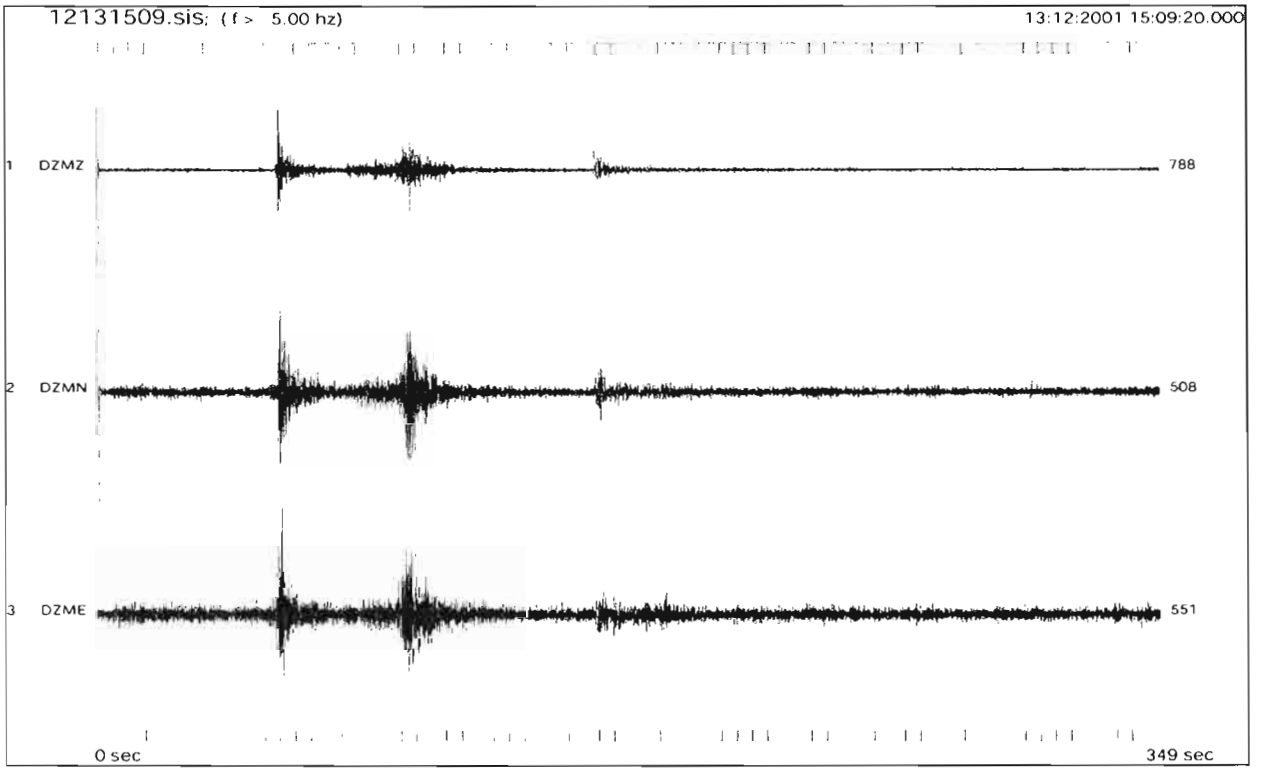

Fig. III-8 : Le même séisme après filtrage

ANNEXE 1

Code source C du programme de synchronisation

```
#define   LOCAL_DATA          "D:\\APPDASE\\DATA\\DO\\"
#define   DISTANT_DATA        "C:\\DASE\\"
#define   DISTANT_PC          "PC-PIERROT"
#define   TEMP_DIR  "C:\\TMP"
#define   TIME_STAMP          "OPLCOMP.STP"
#define   FLISTE              "XFERES.TXT"
#define   COMMANDE  "COMMANDE.BAT"
#define   F_IN                "IN.TXT"
#define   F_OUT               "OUT.TXT"

#define   MINUTE              (60)
#define   HEURE               (60*MINUTE)
#define   JOUR                (24*HEURE)
#define   SEMAINE             (7*JOUR)

 .
 * Cette chaine est présente dans les noms de tous les fichiers
 * à transférer
 */
#define   CHAINE              "CAL"

#define   MAXLEN              1024
```

```
extern int          present(char *, char *, char *);
extern int          ping(char *);
extern int          creerep(char *, CString);
extern int          ftp(char *, char *, long);
extern int          depouille_ftp(char *, char *);
extern void         ajouter_liste(char *, char *, char *);
```

```
#include "stdafx.h"
#include "direct.h"
#include "Constantes.h"
#include <fstream.h>

/*
 * Recherche la chaîne dans le fichier fichier situé sous le répertoire
 * répertoire; renvoie 0 si présent, -1 sinon.
 */
int present(char *repertoire, char *fichier, char *chaine)
{
        int             Status;
        CString ch;
        char    Tampon[MAXLEN];

        Status = chdir(repertoire);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire:\n%s !!!",
                        repertoire);
                AfxMessageBox(ch);
                exit(1);
        }

        ifstream Liste(fichier);
        if (Liste.bad())
        {
                ch.Format("Impossible d'ouvrir:\n%s !!!",
                        fichier);
                AfxMessageBox(ch);
                exit(1);
        }

        for (;;)
        {
                Liste.getline(Tampon, MAXLEN, '\n');
                if (Liste.eof())
                {
                        Liste.close();
                        return(-1);
                }
                int i = strlen(Tampon);
                int j = strlen(chaine);
                if (strstr(Tampon, chaine) != NULL)
                {
                        Liste.close();
                        return(0);
                }
        }

        Liste.close();
        return(-1);
}

int ping(char *machine_distante)
{
        int     Status;
        CString ch;
        FILE *f_in;

        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire temporaire:\n%s !!!",
                        TEMP_DIR);
                AfxMessageBox(ch);
                exit(1);
        }

        f_in = fopen(COMMANDE, "w");
        if (f_in == NULL)
        {
                ch.Format("Ping: impossible de générer le fichier %s", COMMANDE);
                AfxMessageBox(ch);
                fclose(f_in);
                return(-1);
        }

        fprintf(f_in, "@ECHO OFF\n");
        fprintf(f_in, "PING %s > %s", DISTANT_PC, F_OUT);
        fclose(f_in);
        Status = system(COMMANDE);
        if (Status == -1)
        {
                ch.Format("L'exécution de la commande ping a échoué.");
                AfxMessageBox(ch);
                return(-1);
        }

        Status = present(TEMP_DIR, F_OUT, "octets");
        if (Status == 0)
        {
                return(0);
        }
        else
        {
                return(-1);
        }
}

/*
 * Crée les sous-répertoires sur la machine distante par l'intermédiaire d'un script ftp
 * Sur le PC distant, on est directement positionné sur le bon répertoire de base. On a
 * donc à créer deux niveaux de répertoire, par exemple 2002-001, et les sous-répertoires
 * horaires, par exemple 2002-001\00.
 */
int creerep(char *machine_distante, CString repertoire)
{
        int             Status;
        CString ch;
        FILE    *f_in;
        char    Tampon[MAXLEN];

        getcwd(Tampon, MAXLEN);

        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire temporaire:\n%s !!!",
                        TEMP_DIR);
                AfxMessageBox(ch);
                exit(1);
        }

        f_in = fopen(F_IN, "w");
        if (f_in == NULL)
        {
                ch.Format("Cree_rep: impossible de générer le fichier %s", F_IN);
                AfxMessageBox(ch);
                fclose(f_in);
                return(-1);
        }
        fprintf(f_in, "user kwakwe kwakwe\n");
        fprintf(f_in, "mkdir %s\n", repertoire);
        fprintf(f_in, "quit\n");

        fclose(f_in);
        f_in = fopen(COMMANDE, "w");
        if (f_in == NULL)
        {
                ch.Format("Cree_rep: impossible de générer le fichier %s", COMMANDE);
                AfxMessageBox(ch);
                fclose(f_in);
                return(-1);
        }

        fprintf(f_in, "@ECHO OFF\n");
        fprintf(f_in, "FTP -v -n %s < %s > %s\n", machine_distante, F_IN, F_OUT);
        fclose(f_in);
```

```
        Status = system(COMMANDE);
        if (Status == -1)
        {
                ch.Format("L'exécution de la commande ftp a échoué.");
                AfxMessageBox(ch);
                return(-1);
        }

        Status = present(TEMP_DIR, F_OUT, "created");
        if (Status == 0)
        {
                Status = chdir(Tampon);
                if (Status != 0)
                {
                        ch.Format("Impossible de revenir au répertoire initial:\n%s !!!",
                                Tampon);
                        AfxMessageBox(ch);
                }
                return(0);
        }
        else
        {
                Status = chdir(Tampon);
                if (Status != 0)
                {
                        ch.Format("Impossible de revenir au repertoire initial:\n%s !!!",
                                Tampon);
                        AfxMessageBox(ch);
                }
                return(-1);
        }
}

void ajouter_liste(char *rep, char *liste, char *fich)
{
        int             Status;
        CString ch;
        FILE    *Liste;

        Status = chdir(rep);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire:\n%s !!!",
                        rep);
                AfxMessageBox(ch);
                exit(1);
        }

        Liste = fopen(liste, "a");
        if (Liste == NULL)
        {
                ch.Format("Impossible d'ouvrir en append:\n%s !!!",
                        liste);
                AfxMessageBox(ch);
                exit(1);
        }
        fputs(fich, Liste);
        fclose(Liste);
}

long depouille_ftp(char *rep, char *nom)
{
        int             Status;
        CString ch;
        char    Tampon[MAXLEN];
        char    *pt;
        long    longueur_transferee;

        Status = chdir(rep);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire:\n%s !!!",
                        rep);
                AfxMessageBox(ch);
                exit(1);
        }

        ifstream Listing(nom);
        if (Listing.bad())
        {
                ch.Format("Impossible d'ouvrir:\n%s !!!",
                        nom);
                AfxMessageBox(ch);
                exit(1);
        }

        for (;;)
        {
                Listing.getline(Tampon, MAXLEN, '\n');
                if (Listing.eof())
                {
                        Listing.close();
                        return(-1);
                }
                pt = strstr(Tampon, "octets envoy");
                if (pt == NULL)
                        continue;
                /*
                 * Pour pointer sur le nombre d'octets transférés
                 */
                pt = Tampon;
                longueur_transferee = atol(pt);
                Listing.close();
                return(longueur_transferee);
        }
}

/*
 * Transfert par ftp d'un fichier vers le PC distant.
 *
 * Parametres en entrée:        - nom du fichier;
 *                              - répertoire du fichier;
 *                              - longueur du fichier.
 *
 * On effectue le transfert en générant un fichier BAT qui contient la commande
 * FTP. Les entrées/sorties sont redirigées sur les fichiers IN.TXT et OUT.TXT,
 * respectivement. Ces fichiers sont situés dans le répertoire temporaire.
 */
int ftp(char *nom, char *dir, long longueur)
{
        CString ch;
        char    temp[MAXLEN];
        char    *pt;
        int             Status;
        long    Etat;
        FILE    *f_in;

        pt = nom;
        pt += strlen(nom);
        --pt;
        *pt = '\0';

        ch.Format("Transfert de %s dans %s (%ld octets): ",
                nom, dir, longueur);
        printf("%s", ch);

        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire temporaire:\n%s !!!",
                        TEMP_DIR);
                AfxMessageBox(ch);
                exit(1);
        }

        f_in = fopen(F_IN, "w");
        if (f_in == NULL)
        {
                ch.Format("Cree_rep: impossible de générer le fichier %s", F_IN);
                AfxMessageBox(ch);
```

```
                fclose(f_in);
                return(-1);
        }
        fprintf(f_in, "user kwakwe kwakwe\n");
        fprintf(f_in, "cd %s\n", dir);
        strcpy(temp, LOCAL_DATA);
        pt = temp;
        pt += strlen(pt); --pt;
        *pt = '\0';
        fprintf(f_in, "lcd %s\n", temp);
        fprintf(f_in, "lcd %s\n", dir);
        fprintf(f_in, "prompt\n");
        fprintf(f_in, "bin\n");
        fprintf(f_in, "hash\n");
        fprintf(f_in, "put %s\n", nom);
        fprintf(f_in, "quit\n");

        fclose(f_in);
        f_in = fopen(COMMANDE, "w");
        if (f_in == NULL)
        {
                ch.Format("Cree_rep: impossible de générer le fichier %s", COMMANDE);
                AfxMessageBox(ch);
                fclose(f_in);
                return(-1);
        }

        fprintf(f_in, "@ECHO OFF\n");
        fprintf(f_in, "ftp -v -n %s < %s > %s\n", DISTANT_PC, F_IN, F_OUT);
        fclose(f_in);

        Status = system(COMMANDE);
        if (Status == -1)
        {
                ch.Format("L'exécution de la commande ftp a échoué.");
                AfxMessageBox(ch);
                return(-1);
        }

        Etat = depouille_ftp(TEMP_DIR, F_OUT);

        /*
         * Si la longueur transférée est égale à la longueur à transférer,
         * alors le transfert s'est bien passé
         */
        if (Etat == longueur)
                return(0);
        else
                return(-1);
}
```

```
// Transfert.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "direct.h"
#include <io.h>
#include "Transfert.h"
#include "Constantes.h"
#include "Procedures.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////////////
// The one and only application object

CWinApp theApp;

using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
        int                     nRetCode = 0;
        CString         ch;
        char            Tampon[MAXLEN];
        char            ch2[MAXLEN];
        char            ch3[MAXLEN];
        char            *pt;

        int                     Status;
        FILE            *liste;

        struct
        {
                char            dir[MAXLEN];
                char            nom[MAXLEN];
                long            longueur;
        } fichier;

        long                    hier, maintenant;
        struct tm *ptemps;
        int                             annee_presente, jour_present, heure_presente;
        int                             annee, jour, heure;

        // initialize MFC and print and error on failure
        if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
        {
                // TODO: change error code to suit your needs
                cerr << _T("Fatal Error: MFC initialization failed") << endl;
                nRetCode = 1;
        }
        else
        {
                /*
                 * Procedure automatique de transfert de données depuis le
                 * PC d'acquisition (kwakwe) vers pc-baldassari.
                 *
                 * Auteur: Pierre Lebeflegard
                 *
                 * Ce programme gère une liste des fichiers qui ont déjà été transférés,
                 * et transfère les fichiers pas encore transférés sur pc-baldassari
                 * en générant un script ftp vers pc-baldassari. Un serveur ftp est supposé
                 * tourner sur le PC distant, avec le login/mot de passe: kwakwe/kwakwe.
                 *
                 * Structure de données sur le PC local:
                 *
                 * Répertoire principal des données: D:\APPDASE\DATA\DO
                 * Sous-répertoires, aaaa-jjj, par exemple 2002-001 pour le 1er janvier 2002.
                 * Chacun de ces sous-répertoires contient 24 sous-répertoires de 00 à 23. Si
                 * Le contenu de ces sous-répertoires est prêt à être transféré, il contient
                 * un 'time stamp', c'est-à-dire un fichier nommé oplcomp.stp.
                 *
                 * la liste des fichiers déjà transférés est XFERES.TXT, directement
                 * sous le répertoire de base.
                 *
                 * Structure de données sur le PC distant: C:\DASE
                 * La structure en sous-répertoires est la même.
                 */
                Status = chdir(LOCAL_DATA);
                if (Status != 0)
                {
                        ch.Format("Impossible d'accéder au répertoire des données:\n%s !!!",
                                LOCAL_DATA);
                        AfxMessageBox(ch);
                        exit(1);
                }

                Status = chdir(TEMP_DIR);
                if (Status != 0)
                {
                        Status = mkdir(TEMP_DIR);
                        if  Status != 0)
                        {
                                ch.Format "Impossible de créer le répertoire
temporaire:\n%s !!!",
                                        TEMP_DIR);
                                AfxMessageBox(ch),
                                exit(1);
                        }
                }

                liste = fopen(FLISTE, "r");
                if (liste == NULL)
                {
                        liste = fopen(FLISTE, "w");
                        if (liste == NULL)
                        {
                                ch.Format("Impossible de créer le fichier liste:\n%s !!!",
                                        FLISTE);
                                AfxMessageBox(ch);
                                exit(1);
                        }
                        else
                        {
                                fclose(liste);
                        }
                }
                else
                {
                        fclose(liste);
                }

                time(&maintenant);
                ptemps                  =       gmtime(&maintenant);
                annee_presente  =       ptemps->tm_year; annee_presente += 1900;
                jour_present    =       ptemps->tm_yday; ++jour_present;
                heure_presente  =       ptemps->tm_hour;

                hier                    =       maintenant - (SEMAINE);

                /*
                 * On vérifie seulement le transfert des fichiers de moins de une semaine
                 */
                for (;;)
                {
                        ptemps                  = gmtime(&hier);
                        annee                   = ptemps->tm_year; annee += 1900;
                        jour                    = ptemps->tm_yday; ++jour;

                        if (annee > annee_presente)
                                return nRetCode;

                        if ( (annee == annee_presente) && (jour > jour_present) )
                                return nRetCode;

                        ch.Format("Examen du repertoire: %04d-%03d", annee, jour);
                        printf("%s\n", ch);
```

```
Status = chdir(LOCAL_DATA);
if (Status != 0)
{
        ch.Format("Impossible d'accéder au répertoire des
données:\n%s !!!",
                LOCAL_DATA);
        AfxMessageBox(ch);
        exit(1);
}
ch.Format("%04d-%03d", annee, jour);
Status = chdir(ch);
if (Status != 0)
{
        hier += JOUR;
        continue;
}

ch.Format("%04d-%03d", annee, jour);
creerep(DISTANT_PC, ch);

for (heure=0; heure <=23; heure++)
{
        if (
                (annee == annee_presente) &&
                (jour == jour_present) &&
                (heure > heure_presente)
                )
                return nRetCode;

        Status = chdir(LOCAL_DATA);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire des
données :\n%s !!!",
                        LOCAL_DATA);
                AfxMessageBox(ch);
                exit(1);
        }

        ch.Format("%04d-%03d\\%02d", annee, jour, heure);
        Status = chdir(ch);
        if (Status != 0)
                continue;

        getcwd(Tampon, MAXLEN);
        /*
         * Si on trouve le time stamp, alors le contenu de ce répertoire
est bon pour le transfert
         *
         * Si pas de time stamp, on ne regarde pas le contenu de ce
répertoire
         */
        if (access(TIME_STAMP, 00) != 0)
                continue;

        Status = chdir(LOCAL_DATA);
        if (Status != 0)
        {
                ch.Format("Impossible d'accéder au répertoire des
données :\n%s !!!",
                        LOCAL_DATA);
                AfxMessageBox(ch);
                exit(1);
        }

        ch.Format("%04d-%03d\\%02d", annee, jour, heure);
        Status = chdir(ch);
        if (Status != 0)
                continue;

        /*
         * Récupérer le contenu de ce répertoire (pas moyen de faire
mieux, en l'absence
```

```
 * de readdir...) Et en plus, il faut utiliser l'option -C de
DIR pour avoir la taille
 * affichée correctement...
 */
ch.Format("DIR /-C %s*.00c >%s\\%s", CHAINE, TEMP_DIR, F_OUT);
Status = system(ch);
if (Status == -1)
{
        ch.Format("Impossible de lister le répertoire:\n%s!!!",
Tampon);
        AfxMessageBox(ch);
        continue;
}
/*
 * Dépouillement
 */
ch.Format("%s\\%s", TEMP_DIR, F_OUT);
liste = fopen(ch, "r");
if (liste == NULL)
{
        ch.Format("Impossible de lister le répertoire:\n%s!!!",
Tampon);
        AfxMessageBox(ch);
        continue;
}
for (;;)
{
        if (fgets(ch2, MAXLEN, liste) == NULL)
        {
                fclose(liste);
                break;
        }
        /*
         * Extraire les caractéristiques du fichier à
transférer
         */
        pt = strstr(ch2, CHAINE);
        if (pt == NULL)
                continue;
        strcpy(fichier.nom, pt);
        /*
         * On recule le pointeur de 10 pour pointer à la
Longueur du fichier
         */
        pt -= 10;
        fichier.longueur = atol(pt);
        if (fichier.longueur == 0)
        {
                ch.Format("Longueur incorrecte (0) pour le
fichier:\n%s!!!", fichier.nom);
                puts(ch);
                continue;
        }
        sprintf(fichier.dir, "%04d-%03d\\%02d", annee, jour,
heure);
        /*
        * Il faut regarder si ce fichier a déjà été transféré,
c'est-à-dire
         * s'il est présent dans la liste, sous la forme
hh\<nom du fichier>
         * Pour économiser les transferts/examens, il y a une
liste par répertoire
         * quotidien (aaaa-jjj\xferts.txt)
         */
        sprintf(ch3, "%04d-%03d", LOCAL_DATA, annee, jour);
        sprintf(ch2, "%02d\\%s", heure, fichier.nom);
        pt = ch2; pt += strlen(ch2); *pt = '\0';
        if (present(ch3, FLISTE, ch2) != 0)
        {
                ch.Format("%04d-%03d\\%02d", annee, jour,
heure);
                creerep(DISTANT_PC, ch);
                if (ftp(fichier.nom, fichier.dir,
fichier.longueur) != 0)
```

```
                                                {
                                                        printf("non realise...\n");
                                                        continue;
                                                }
                                                else
                                                {
                                                        printf("OK.\n");
                                                }
                                                /*
                                                 * Le transfert est OK, on ajoute
repertoire+nom à la liste                        */
                                                sprintf(ch2, "%02d\\%s\n", heure,

fichier nom);                                   ajouter_liste(ch3, FLISTE, ch2);
                                                                }
                                                        }
                                                hier += JOUR;
                                        }
                                }
                        return nRetCode;
```

ANNEXE 2

Code source C du programme de détection des séismes

```
#include "time.h"

#define   DATA                "C:\\DASE\\"
#define   OUT_DIR             "Y:\\"
#define   TEMP_DIR            "C:\\TMP"
#define   TEMP_FILE           "TEMPO.TXT"
#define   LOG_DIR             "C:\\DASE\\LOG"
#define   NULL_FILE           "NULL.TXT"
#define   TIME_STAMP          "DETECTE.STP"
#define   BATCHFILE           "COMMANDE.BAT"
#define   F_IN                "IN.TXT"
#define   F_OUT               "OUT.TXT"

#define   MAXLEN              1024

#define   SECOND              (1)
#define   MINUTE              (60*SECOND)
#define   HOUR                (60*MINUTE)
#define   DAY                 (24*HOUR)
#define   WEEK                (7*DAY)

/*
 * Number of days checked backward
 */
#define   NBDAYS              (2*WEEK)

/*
 * Recursive filter coeff
 */
#define   A0                  (0.25)

/*
 * Maximum number of earthquakes in one 1/2h slice
 */
#define   MAXQK               16

/*
 * Duration for STA/LTA intervals
 */
#define   STA                 (1*SECOND)
#define   LTA                 (60*SECOND)

/*
 * Triggering threshold (STA/LTA ratio)
 */
#define   S_DEC               (5.5)

 *
 * Triggering duration: an earthquake is detected if STA/LTA
 * is above S_DEC during at least D_DEC samples
 */
#define   D_DEC               (STA/2.0)

 *
 * Releasing duration (STA/LTA ratio)
 */
#define   S_REL               (1.5)

/*
 * Releasing duration: earthquake is over if STA/LTA is below S_REL
 * during at least D_REL samples. One have to multiply by frequency
 * to get number of points.
 */
#define   D_REL               (STA*1.5)

/*
 * Pre-event duration. To get number of points, one have
 * to multiply by frequency
 */
#define   D_PRE               (LTA*1.0)

/*
 * Post-event duration. To get number of points, one have
 * to multiply by frequency
 */
#define   D_POST              (LTA*3.0)

/*
 * A priori value for frequency
 */
#define   FREQ                100

/*
 * Max number of points for 1/2 h
 */
#define   MAXVALS             (LTA*FREQ+FREQ*30*MINUTE)

#define   STATION             "DZM"

/*
 * Block size in bytes
 */
#define   BLOCKSIZE (1024/sizeof(short))

struct information
{
        char      ndir[MAXLEN];
        char      nsdir[MAXLEN];
        char      filZ[MAXLEN];
        char      filE[MAXLEN];
        char      filN[MAXLEN];
        time_t    begin;
        time_t    end;
        int               frequency;
        int               duration;
};
```

```
extern int          present(char *, char *, char *);
extern int          dysize(int);
extern void         examine(char *, char *, char *, int *, struct information *);
extern void         detect(char *, char *, int, int *, struct information *);
extern int          p_h_rec(double *, int, double);
extern int          p_b_rec (double *, int, double);

extern    FILE      *log;
```

```
#include "stdafx.h"
#include "direct.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include "Constants.h"
#include "procedures.h"
#include "time.h"
#include <fstream.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>

int dysize(int year)
{
        if (year % 4 != 0)
                        return (365);
        if (year % 400 == 0)
                        return (366);
        if (year % 100 == 0)
                        return (365);
        /*
         * Otherwise leap year
         */
        return (366);
}

void examine(char *rep, char *srep, char *nfic, int *NbFiles, struct information *info)
{
        char    orig[MAXLEN];
        char    ligne[MAXLEN];
        char    ch[MAXLEN];
        int             Status;
        char    component, *pt;
        FILE    *batch;
        FILE    *listing;
        int             year, month, day, hour, minute, second;
        struct  tm date;
        struct  information     *ptinfo, ptemp;

        if( _getcwd( orig, MAXLEN ) == NULL )
        {
                fprintf(stderr, "Can't get working dir...\n");
                exit(1);
        }

        strcpy(ptemp.ndir, rep);
        strcpy(ptemp.nsdir, srep);
        ptemp.begin = 0;
        ptemp.end = 0;
        ptemp.fileE[0] = '\0';
        ptemp.fileN[0] = '\0';
        ptemp.fileZ[0] = '\0';

        /*
         * In the temp dir, sigchk is made on the file given as a parameter. Next, sigchk
         * result, which is an ASCII file, is checked.
         */
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to temp dir: %s.\n", TEMP_DIR);
                exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
                fprintf(stderr, "Can't create batch file: %s.\n", BATCHFILE);
                exit(1);
        }

        fprintf(batch, "@ECHO OFF\n", DATA);
        fprintf(batch, "cd %s\n", DATA);
        fprintf(batch, "cd %s\n", rep);
        fprintf(batch, "cd %s\n", srep);
        fprintf(batch, "sigchk %s > %s\\%s\n", nfic, TEMP_DIR, TEMP_FILE);
        fclose(batch);

        if (system(BATCHFILE) == -1)
        {
                fprintf(stderr, "Can't execute batch file: %s.\n", BATCHFILE);
                exit(1);
        }

        /*
         * Check sigchk results
         */
        sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
        listing = fopen(ch, "r");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't check sigchk output \n");
                exit(1);
        }
        for(;;)
        {
                if (fgets(ligne, MAXLEN, listing) == NULL)
                {
                        fclose(listing);
                        break;
                }
                if (strstr(ligne, "voie") != NULL)
                {
                        pt = ligne+38;
                        component = *pt;
                        switch (component)
                        {
                        case 'E':
                                strcpy(ptemp.fileE, nfic);
                                break;
                        case 'N':
                                strcpy(ptemp.fileN, nfic);
                                break;
                        case 'Z':
                                strcpy(ptemp.fileZ, nfic);
                                break;
                        }
                        continue;
                }
                if (strstr(ligne, "quence") != NULL)
                {
                        pt = ligne+34;
                        ptemp.frequency = atoi(pt);
                        continue;
                }
                if (strstr(ligne, "dur") != NULL)
                {
                        pt = ligne+34;
                        ptemp.duration = atoi(pt);
                        ptemp.duration /= 1000; /* Seconds */
                        continue;
                }
                /*
                 * Date of begin of file (seconds from 1970/01/01)
                 */
                if (strstr(ligne, "incluse") != NULL)
                {
                        pt = ligne+41; year             = atoi(pt);
                        pt = ligne+38; month            = atoi(pt);
                        pt = ligne+35; day              = atoi(pt);
                        pt = ligne+46; hour             = atoi(pt);
                        pt = ligne+49; minute           = atoi(pt);
                        pt = ligne+52; second           = atoi(pt);
                        date.tm_year            =       year-1900;
                        date.tm_mon                     =               month-1;
                        date.tm_mday            =       day;
                        date.tm_hour            =       hour;
                        date.tm_min                     =               minute;
```

```
                date.tm_sec             =       second;
                ptemp.begin             =       mktime(&date);
                continue;
        }
        /*
         * Date of end of file (seconds from 1970/01/01)
         */
        if (strstr(ligne, "exclue") != NULL)
        {
                pt = ligne+41; year             = atoi(pt);
                pt = ligne+38; month            = atoi(pt);
                pt = ligne+35; day              = atoi(pt);
                pt = ligne+46; hour             = atoi(pt);
                pt = ligne+49; minute           = atoi(pt);
                pt = ligne+52; second           = atoi(pt);
                date.tm_year            =       year-1900;
                date.tm_mon             =       month-1;
                date.tm_mday            =       day;
                date.tm_hour            =       hour;
                date.tm_min             =       minute;
                date.tm_sec             =       second;
                ptemp.end               =       mktime(&date);
                continue;
        }
}

/*
 * One have to check now if this is a component belonging to an already existing hourly slice,
 * or a new hourly slice (maw two hourly slices in a hourly subdirectory).
 */
for (int i=0; i< *NbFiles; i++)
{
        ptinfo = &(info[i]);
        if ( (ptemp.begin==ptinfo->begin) && (ptemp.end==ptinfo->end) )
        {
                switch(component)
                {
                case 'E':
                        strcpy(ptinfo->filE, ptemp.filE);
                        break;
                case 'N':
                        strcpy(ptinfo->filN, ptemp.filN);
                        break;
                case 'Z':
                        strcpy(ptinfo->filZ, ptemp.filZ);
                        break;
                }
                Status = chdir(orig);
                if (Status != 0)
                {
                        fprintf(stderr, "Can't get back to original directory: %s.\n",

                        exit(1);
                }
                return;
        }
}
/*
 * This is a new hourly slice
 */
if (*NbFiles < 2)
/*
 * 2x3 components max in a hourly subdirectory
 */
{
        ptinfo = info; ptinfo += *NbFiles; ++(*NbFiles);
        strcpy(ptinfo->rdir, ptemp.rdir);
        strcpy(ptinfo->sdir, ptemp.sdir);
        ptinfo->begin = ptemp.begin;
        ptinfo->end = ptemp.end;
        ptinfo->duration = ptemp.duration;
        ptinfo->frequency = ptemp.frequency;
        switch(component)
        {
        case 'E':
                strcpy(ptinfo->filE, ptemp.filE);
                break;
        case 'N':
                strcpy(ptinfo->filN, ptemp.filN);
                break;
        case 'Z':
                strcpy(ptinfo->filZ, ptemp.filZ);
                break;
        }
}

Status = chdir(orig);
if (Status != 0)
{
        fprintf(stderr, "Can't get back to original directory: %s.\n", orig);
        exit(1);
}
}

void Detect(char *rep, char *srep, int FileIndex, int *NbFiles, struct information *info)
{
        struct  information     ptemp, *previous, *next, *current;
        struct  information *ptinfo;

        ptinfo = info; ptinfo += FileIndex; current = ptinfo;

        printf("Earthquake detection on Z component (file %s\\%s\\%s)\n",
                ptinfo->ndir, ptinfo->nsdir, ptinfo->filZ);

        char    reps[MAXLEN];
        char    sreps[MAXLEN];
        char    orig[MAXLEN];
        char    ligne[MAXLEN];
        char    ch[MAXLEN];
        char    Tamp[MAXLEN];
        int             Status;
        char    component, *pt;
        FILE    *batch;
        FILE    *listing;
        int             year, month, day, hour, minute, second;
        int             NbValues;
        struct  tm date;
        struct  _finddata_t     rep2;   long    h_rep2;
        short   *valZ = NULL;
        short   *valE = NULL;
        short   *valN = NULL;
        short   valsign[MAXVALS];
        short   *valabs, *ptval_o, *ptval_d;
        double  *HighPass, *LowPass, *PtPB, *PtPH;
        int             iflag = 0; clock_t  start = clock();;
        long    SumSTA, SumLTA;
        int             i, nbptsSTA, nbptsLTA, BeginSTA, BeginLTA, fin;

        if( _getcwd( orig, MAXLEN ) == NULL )
        {
                fprintf(stderr, "Can't get working dir...");
                exit(1);
        }

        /*
         * In the temp dir, sigchk is made on the file given as a parameter. Next, sigchk
         * result, which is an ASCII file, is checked.
         */
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to temp dir:%s.\n", TEMP_DIR);
                exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
                fprintf(stderr, "Can't create batch file: %s.\n", BATCHFILE);
                exit(1);
```

```
}

fprintf(batch, "@ECHO OFF\n", DATA);
fprintf(batch, "cd %s\n", DATA);
fprintf(batch, "cd %s\n", rep);
fprintf(batch, "cd %s\n", srep);
fprintf(batch, "sigchk %s > %s\\%s\n", ptinfo->filZ, TEMP_DIR, TEMP_FILE);
fclose(batch);

if (system(BATCHFILE) == -1)
{
        fprintf(stderr, "Can't execute batch file: %s \n", BATCHFILE);
        exit(1);
}

/*
 * Examining sigchk output
 */
sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
listing = fopen(ch, "r");
if (listing == NULL)
{
        fprintf(stderr, "Can't examine sigchk output.\n");
        exit(1);
}

strcpy(ptemp.ndir, rep);
strcpy(ptemp.nsdir, srep);
ptemp.begin = 0;
ptemp.end = 0;
ptemp.filE[0] = '\0';
ptemp.filN[0] = '\0';
ptemp.filZ[0] = '\0';

for(;;)
{
        if (fgets(ligne, MAXLEN, listing) == NULL)
        {
                fclose(listing);
                break;
        }
        if (strstr(ligne, "voie") != NULL)
        {
                pt = ligne+38;
                component = *pt;
                switch (component)
                {
                case 'E':
                        strcpy(ptemp.filE, ptinfo->filE);
                        break;
                case 'N':
                        strcpy(ptemp.filN, ptinfo->filN);
                        break;
                case 'Z':
                        strcpy(ptemp.filZ, ptinfo->filZ);
                        break;
                }
                continue;
        }
        if (strstr(ligne, "quence") != NULL)
        {
                pt = ligne+34;
                ptemp.frequency = atoi(pt);
                continue;
        }
        if (strstr(ligne, "dur") != NULL)
        {
                pt = ligne+34;
                ptemp.duration = atoi(pt);
                ptemp.duration /= 1000; /* In seconds */
                continue;
        }
        /*
         * Date of the beginning of file (seconds since 1970/01/01)
         */
        if (strstr(ligne, "incluse") != NULL)
        {
                pt = ligne+41; year         = atoi(pt);
                pt = ligne+38; month        = atoi(pt);
                pt = ligne+35; day          = atoi(pt);
                pt = ligne+46; hour         = atoi(pt);
                pt = ligne+49; minute       = atoi(pt);
                pt = ligne+52; second       = atoi(pt);
                date.tm_year       =        year-1900;
                date.tm_mon                 =        month-1;
                date.tm_mday       =        day;
                date.tm_hour       =        hour;
                date.tm_min                 =        minute;
                date.tm_sec                 =        second;
                ptemp.begin        =        mktime(&date);
                continue;
        }
        /*
         * Date of the end of file (seconds since 1970/01/01)
         */
        if (strstr(ligne, "exclue") != NULL)
        {
                pt = ligne+41; year         = atoi(pt);
                pt = ligne+38; month        = atoi(pt);
                pt = ligne+35; day          = atoi(pt);
                pt = ligne+46; hour         = atoi(pt);
                pt = ligne+49; minute       = atoi(pt);
                pt = ligne+52; second       = atoi(pt);
                date.tm_year       =        year-1900;
                date.tm_mon                 =        month-1;
                date.tm_mday       =        day;
                date.tm_hour       =        hour;
                date.tm_min                 =        minute;
                date.tm_sec                 =        second;
                ptemp.end          =        mktime(&date);
                continue;
        }
}
if ( (ptemp.begin!=0) && (ptemp.end!=0) )
        if (ptemp.duration == 0)
                ptemp.duration = ptemp.end - ptemp.begin;
/*
 * In order to make detection even on the beginning of signal, it is necessary to get back
last samples
 * (more precisely those corresponding to an STA interval) of the previous Z component. In the
current
 * directory, there are two Z components: the current one, and either the previous one, or the
next one.
 * If this is the current and the previous one, everything is OK; if not the previous Z
component is in
 * the previous directory.
 */
ptinfo = info; previous = NULL;
for (i=0; i<*NbFiles; i++, ++ptinfo)
{
        /*
         * This is Z component currently examined
         */
        if (ptinfo->begin == ptemp.begin)
                continue;

        /*
         * Found it! This is previous Z component
         */
        if (ptinfo->end == ptemp.begin)
        {
                previous = ptinfo;
                break;
        }
}

/*
 * If previous Z component is not in the current directory, one has to seek it in the previous
directory
 */
```

```
if (previous == NULL)
{
        int yy, dd, hh;

        yy = atoi(rep);
        dd = atoi(rep+5);
        hh = atoi(srep);

        if (hh < 0)
        {
                --hh;
                goto finished;
        }
        if (dd < 1)
        {
                --dd;
                hh = 23;
                goto finished;
        }
        --yy;
        dd = dysize(yy);
        hh = 23;

finished:
        sprintf(reps, "%04d-%03d", yy, dd);
        sprintf(sreps, "%02d", hh);
        Status = chdir(DATA);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to data directory. %s.\n", DATA);
                exit(1);
        }
        Status = chdir(reps);
        if (Status != 0)
                goto OnTheRoadAgain;

        Status = chdir(sreps);
        if (Status != 0)
                goto OnTheRoadAgain;

        struct information infop[2];
        int NbFilesPrev = 0;
        /*
         * Only 8 bytes names
         */
        if ( (h_rep2=_findfirst("????????.00c", &rep2)) == -1L )
        {
                fprintf(stderr, "Empty sub-directory.\n");
                exit(1);
        }
        else
        {
                examine(reps, sreps, rep2.name, &NbFilesPrev, &(infop[0]));
        }

        while ( _findnext(h_rep2, &rep2) == 0)
        {
                if (!strcmp(rep2.name, ".."))
                        continue;
                examine(reps, sreps, rep2.name, &NbFilesPrev, &(infop[0]));
        }
        _findclose(h_rep2);

        ptinfo = &(infop[0]);
        for (int i=0; i<NbFilesPrev; i++, ++ptinfo)
        {
                /*
                 * This is Z component currently examined
                 */
                if (ptinfo->begin == ptemp.begin)
                        continue;

                /*
                 * Found it! This is previous Z component
                 */
                if (ptinfo->end == ptemp.begin)
                {
                        previous = ptinfo;
                        break;
                }
        }
}

/*
 * If previous file exists, it has been found, and corresponding
 * structure is pointed at by the "previous" pointer
 */
OnTheRoadAgain:
/*
 * Seeking next Z component
 */
ptinfo = infop; next = NULL;
for (i=0; i<NbFiles; i++, ++ptinfo)
{
        /*
         * This is Z component currently examined
         */
        if (ptinfo->begin == ptemp.begin)
                continue;

        /*
         * Found it! This is next Z component
         */
        if (ptinfo->begin == ptemp.end)
        {
                next = ptinfo;
                break;
        }
}
/*
 * If next Z component is not in the current directory, one has to seek it in the next
directory
 */
if (next == NULL)
{
        int yy, dd, hh;

        yy = atoi(rep);
        dd = atoi(rep+5);
        hh = atoi(srep);

        if (hh < 23)
        {
                ++hh;
                goto Finished2;
        }
        if (dd < dysize(yy))
        {
                ++dd;
                hh = 00;
                goto Finished2;
        }
        ++yy;
        dd = 01;
        hh = 00;

Finished2:
        sprintf(reps, "%04d-%03d", yy, dd);
        sprintf(sreps, "%02d", hh);
        Status = chdir(DATA);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to data directory. %s.\n", DATA);
                exit(1);
        }
        Status = chdir(reps);
        if (Status != 0)
                goto OnTheRoadAgain2;

        Status = chdir(sreps);
```

```
        if (Status != 0)
                goto OnTheRoadAgain2;

        struct information infos[2];
        int NbFilesNext = 0;
        /*
         * Only 8 bytes names
         */
        if ( (h_rep2=_findfirst("????????.00c", &rep2)) == -1L )
        {
                fprintf(stderr, "Empty sub-directory.\n");
                exit(1);
        }
        else
        {
                examine(reps, sreps, rep2.name, &NbFilesNext, &(infos[0]));
        }

        while ( _findnext(h_rep2, &rep2) == 0)
        {
                if (!strcmp(rep2.name, ".."))
                        continue;
                examine(reps, sreps, rep2.name, &NbFilesNext, &(infos[0]));
        }
        _findclose(h_rep2);

        ptinfo = &(infos[0]);
        for (int i=0; i<NbFilesNext; i++, ++ptinfo)
        {
                /*
                 * This is Z component currently examined
                 */
                if (ptinfo->begin == ptemp.begin)
                        continue;

                /*
                 * Found it! This is previous Z component
                 */
                if (ptinfo->begin == ptemp.end)
                {
                        next = ptinfo;
                        break;
                }
        }
}
OnTheRoadAgain2:;
/*
 * First read samples of previous Z component, if found
 */
NbValues = 0;
/*
 *      Reading samples of previous Z component (only LTA last samples will be kept)
 */
if (previous != NULL)
{
        /*
         * Reading previous Z component
         */
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to temp dir: %s.\n", TEMP_DIR);
                exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
                fprintf(stderr, "Can't create batch file. %s.\n", BATCHFILE);
                exit(1);
        }

        fprintf(batch, "@ECHO OFF\n", DATA);
        fprintf(batch, "cd %s\n", DATA);
        fprintf(batch, "cd %s\n", previous->ndir);
        fprintf(batch, "cd %s\n", previous->nsdir);
        fprintf(batch, "sigconv ascii e %s %s\\%s > %s\\%s\n",
                        previous->filZ,
                        TEMP_DIR, TEMP_FILE,
                        TEMP_DIR, NULL_FILE);
        fclose(batch);

        if (system(BATCHFILE) == -1)
        {
                fprintf(stderr, "Can't execute batch file: %s.\n", BATCHFILE);
                exit(1);
        }

        /*
         * Read samples values
         */
        /*
         * Decode sigconv output (ASCII samples values)
         */
        sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE),
        listing = fopen(ch, "r");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't check sigconv output.\n");
                exit(1);
        }

        for(;;)
        {
                if (fgets(ligne, MAXLEN, listing) == NULL)
                {
                        fclose(listing);
                        break;
                }
                if (NbValues >= MAXVALS)
                {
                        fprintf(stderr, "Maximum number of points reached: %d.\n",
MAXVALS);
                        exit(1);
                }
                valsign[NbValues++] = atoi(ligne);
        }
        /*
         * Move LTA last samples of the previous Z component at the beginning of the array
         */
        ptval_d = &(valsign[0]);
        ptval_o = &(valsign[NbValues-1]); ptval_o -= (LTA*previous->frequency);
        for (i=0; i<LTA*previous->frequency; i++)
                *ptval_d++ = *ptval_o++;

        NbValues = LTA*previous->frequency;

}
/*
 *      Now, read (and keep) all values of current Z component
 */
Status = chdir(TEMP_DIR);
if (Status != 0)
{
        fprintf(stderr, "Can't cd to temp dir: %s.\n", TEMP_DIR);
        exit(1);
}

batch = fopen(BATCHFILE, "w");
if (batch == NULL)
{
        fprintf(stderr, "Can't create batch file: %s.\n", BATCHFILE);
        exit(1);
}

fprintf(batch, "@ECHO OFF\n", DATA);
fprintf(batch, "cd %s\n", DATA);
fprintf(batch, "cd %s\n", current->ndir);
fprintf(batch, "cd %s\n", current->nsdir);
fprintf(batch, "sigconv ascii e %s %s\\%s > %s\\NULL\n",
```

```
        ptemp.fil2, TEMP_DIR, TEMP_FILE, TEMP_DIR);
fclose(batch);

if (system(BATCHFILE) == -1)
{
        fprintf(stderr, "Can't execute batch file: %s \n", BATCHFILE);
        exit(1);
}
/*
 * Read samples values
 */
/*
 * Decode sigconv output (ASCII samples values)
 */
sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
listing = fopen(ch, "r");
if (listing == NULL)
{
        fprintf(stderr, "Can't check sigconv output.\n");
        exit(1);
}
for(;;)
{
        if (fgets(ligne, MAXLEN, listing) == NULL)
        {
                fclose(listing);
                break;
        }
        if (NbValues >= MAXVALS)
        {
                fprintf(stderr, "Maximum number of points reached: %d.\n", MAXVALS);
                exit(1);
        }
        valsign[NbValues++] = atoi(ligne);
}

val2 = (short *) calloc (NbValues, sizeof(short));
if (val2 == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}
for (i=0; i< NbValues; i++)
        val2[i] = valsign[i];

valabs = (short *) calloc (NbValues, sizeof(short));
if (valabs == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}

HighPass = (double *) malloc (NbValues*sizeof(double));
if (HighPass == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}

LowPass = (double *) malloc (NbValues*sizeof(double));
if (LowPass == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}

ptval_o = &(valsign[0]);
PtPB = LowPass;
PtPH = HighPass;
for (i=0; i<NbValues; i++)
{
        *PtPB++= (double) *ptval_o;
        *PtPH++= (double) *ptval_o++;
}
```

```
/*
 * High Pass and Low Pass filters
 */
p_b_rec(LowPass, NbValues, A0);
p_h_rec(HighPass, NbValues, A0);

PtPH = HighPass;
ptval_d = &(valsign[0]);
for (i=0; i<NbValues; i++)
        *ptval_d++= (short) *PtPH++;

ptval_o = &(valsign[0]);
ptval_d = valabs;
for (i=0; i<NbValues; i++)
        if (*ptval_o < 0.0)
                *ptval_d++ = -(*ptval_o++);
        else
                *ptval_d++ = *ptval_o++;

/*
 * Initialise sum for LTA (LTA*frequency)
 */
nbptsLTA  = current->frequency*LTA;
nbptsSTA  = current->frequency*STA;
BeginLTA  = 0;
fin                     = BeginLTA+nbptsLTA;
BeginSTA  = fin-nbptsSTA;

i = BeginLTA; SumLTA    = 0;
for(i=BeginLTA;i<fin;i++)
        SumLTA += valabs[i];

/*
 * Initialise sum for STA (STA*frequency)
 */
i = BeginSTA; SumSTA    = 0;
for(i=BeginSTA;i<fin;i++)
        SumSTA += valabs[i];

/*
 * Computation on all values (sliding sum/mean/STA/LTA)
 */
int             ioldSTA                 =BeginSTA;
int             ioldLTA                 =BeginLTA;
int             BeginQuake              = -1;
int             EndQuake        = -1;
bool    Triggered       =false;
bool    ThereIsAQuake   =false;
bool    Fine                    =true;
double  Ratio                   = 0.0;
double  sta, lta, memoLTA;

struct  seisme
{
        int     begin;
        int     end;
} quake[MAXQK];

int     NbQuakes = 0;

Status = chdir(TEMP_DIR);
if (Status != 0)
{
        fprintf(stderr, "Can't cd to temporary directory: %s.\n", TEMP_DIR);
        exit(1);
}

listing = fopen(TEMP_FILE, "w");
if (listing == NULL)
{
        fprintf(stderr, "Can't create temporary file: %s.\n", TEMP_FILE);
        exit(1);
}

for(int inew=fin; inew <NbValues; inew++)
```

```
{
        SumLTA -= valabs[inew];
        SumSTA -= valabs[inew];

        sta = (double) SumSTA / (double) nbptsSTA;
        lta = (double) SumLTA / (double) nbptsLTA;

        Ratio = sta / lta;

        if (Ratio >= S_DEC)
        {
                if (Triggered == false)
                {
                        memoLTA = lta;
                        BeginQuake = inew;
                        Triggered = true;
                }
                /*
                 * Triggered, and above threshold: nothing to do
                 */
                else;
        }
        else
        {
                /* Going below threshold */
                if ( (Triggered == true) && (ThereIsAQuake == false) )
                {
                        /*
                         * One has to check if duration above threshold has been long
enough
                         */
                        if ( (inew-BeginQuake) >= D_DEC*ptinfo->frequency )
                        {
                                if (++NbQuakes>MAXQK)
                                {
                                        fprintf(stderr, "Too much earthquakes
detected in this file.\n");
                                        --NbQuakes;
                                        BeginQuake = -1;
                                        Triggered = false;
                                        ThereIsAQuake = false;
                                }
                                else
                                {
                                        quake[NbQuakes-1].begin       = BeginQuake;
                                        /*
                                        quake[NbQuakes-1].end                 =
(previous == NULL)? inew : inew-nbptsLTA;
                                         */
                                        quake[NbQuakes-1].end                 =
inew;
                                        ThereIsAQuake = true;
                                        /*
                                         * Cas de séisme se produisant tout près de
la fin
                                         */
                                        if (quake[NbQuakes-1].end <= quake[NbQuakes-
1].begin)
                                        {
                                                --NbQuakes;
                                                BeginQuake = -1;
                                                ThereIsAQuake = false;
                                                Triggered = false;
                                        }
                                }
                        }
                        else
                        /*
                         * Duration not long enough
                         */
                        {
                                BeginQuake = -1;
                                Triggered = false;
                        }
                }
                /*
                 * Not triggered, and below threshold: nothing to do
                 */
                else;
        }

        /*
         * If an earthquake has been detected, one has to check if STA/LTA ratio has not
gone
         * below releasing threshold.
         */
        if (ThereIsAQuake == true)
                if (Ratio < S_REL)
                {
                        /*
                         * First time below releasing threshold.
                         */
                        if (EndQuake == -1)
                                EndQuake = (previous == NULL)? inew : inew-nbptsLTA;
                        /*
                         * End of earthquake
                         */
                        else if ( (inew-EndQuake) >= D_REL*ptinfo->frequency )
                        {
                                quake[NbQuakes-1].end = inew;

                                /*
                                 * Add pre-event and post-event durations
                                 */
                                quake[NbQuakes-1].begin -= (int)
D_PRE*ptemp.frequency,
                                if (quake[NbQuakes-1].begin <0)
                                        quake[NbQuakes-1].begin = 0;
                                quake[NbQuakes-1].end += (int) D_POST*ptemp.frequency;

                                /*
                                if (quake[NbQuakes-1].end >= NbValues)
                                        quake[NbQuakes-1].end = NbValues-1;
                                */

                                BeginQuake = -1;
                                Triggered = false;
                                ThereIsAQuake = false;
                        }
                        /*
                         * Transition below releasing threshold has not been long enough
                         */
                        else;
                }

        /*
        if (previous == NULL)
        {
                i = inew;
                fprintf(listing, "%d\t%d\t%.02lf\t%.02lf\t%.02lf\n",
                        inew, valZ[inew], sta, lta, Ratio*100.0);
        }
        else
        {
                i = inew - nbptsLTA;
                fprintf(listing, "%d\t%d\t%.02lf\t%.02lf\t%.02lf\n",
                        i, valZ[inew], sta, lta, Ratio*100.0);
        }

        if (previous == NULL)
        {
                i = inew;
                fprintf(listing, "%d\t%d\t%.02lf\t%.02lf\n",
                        inew, valsign[inew], HighPass[inew], LowPass[inew]);
        }
        else
        {
                i = inew - nbptsLTA;
                fprintf(listing, "%d\t%d\t%.02lf\t%.02lf\n",
                        i, valsign[inew], HighPass[inew], LowPass[inew]);
```

```
        }
        */

        SumLTA -= valabs[ioldLTA--];
        SumSTA -= valabs[ioldSTA--];
    }

    fclose(listing);

    for (i=0; i<NbQuakes; i++)
    {
        /*
         * Rounding to an integer number of second
         */
        quake[i].begin = current->frequency * (quake[i].begin/current->frequency);
    }

    /*
     * Merge adjacent earthquakes
     */
    if (NbQuakes > 1)
    {
        Fine = true;
        i = 0;
        int j = 1;
        for (;;)
        {
            if (j>=NbQuakes)
                break;

            if (quake[j].begin > quake[i].end)
            {
                ++i; ++j;
                continue;
            }
            else
            {
                quake[i].end = quake[j].end;
                for (int k=j; k<NbQuakes-1; k++)
                {
                    quake[k] = quake[k+1];
                }
                --NbQuakes;
            }
        }
    }

    /*
     * Time Stamp is created to avoid redundant detections
     */
    else
    {
        Status = chdir(DATA);
        if (Status != 0)
        {
            fprintf(stderr, "Can't cd to directory: %s.\n", DATA);
            exit(1);
        }
        Status = chdir(current->ndir);
        if (Status != 0)
        {
            fprintf(stderr, "Can't cd to directory: %s.\n", current->ndir);
            exit(1);
        }
        Status = chdir(current->nsdir);
        if (Status != 0)
        {
            fprintf(stderr, "Can't cd to directory: %s.\n", current->nsdir);
            exit(1);
        }
        listing = fopen(TIME_STAMP, "w");
        if (listing == NULL)
        {
            fprintf(stderr, "Can't create time stamp.\n");
            exit(1);
        }
        else
            fclose(listing);
    }

    if (NbQuakes > 0)
        fprintf(log, "=========================================\n"); fflush(log);

    for (i=0; i<NbQuakes; i++)
    {
        printf("Earthquake detected: %s\\%s\\%s, begin: %d, end: %d\n",
            current->ndir,
            current->nsdir,
            current->filZ,
            quake[i].begin,
            quake[i].end);
        fprintf(log, "Earthquake detected: %s\\%s\\%s, begin: %d, end: %d\n",
            current->ndir,
            current->nsdir,
            current->filZ,
            quake[i].begin,
            quake[i].end);
    }


    /*
     * When earthquakes are detected, is is necessary to read the two other components (N and E)
     */
    if (NbQuakes > 0)
    {
        int Nbv;

        Nbv = 0;
        valE = (short *) calloc (NbValues, sizeof(short));
        if (valE == NULL)
        {
            fprintf(stderr, "Can't allocate data for array.\n");
            exit(1);
        }

        if (previous != NULL)
            if (strlen(previous->filE) != 0)
            {
                /*
                 * Read previous E component
                 */
                Status = chdir(TEMP_DIR);
                if (Status != 0)
                {
                    fprintf(stderr, "Can't cd to temp dir: %s.\n",
TEMP_DIR);
                    exit(1);
                }

                batch = fopen(BATCHFILE, "w");
                if (batch == NULL)
                {
                    fprintf(stderr, "Can't create batch file: %s.\n",
BATCHFILE);
                    exit(1);
                }

                fprintf(batch, "echo OFF\n", DATA);
                fprintf(batch, "cd %s\n", DATA);
                fprintf(batch, "cd %s\n", previous->ndir);
                fprintf(batch, "cd %s\n", previous->nsdir);
                fprintf(batch, "sigconv ascii e %s %s\\%s > %s\\%s\n",
                    previous->filE,
                    TEMP_DIR, TEMP_FILE,
                    TEMP_DIR, NULL_FILE);
                fclose(batch);

                if (system(BATCHFILE) == -1)
                {
```

```
                fprintf(stderr, "Can't execute batch file: %s.\n",
BATCHFILE);
                exit(1);
            }
            /*
             * Read samples values
             */
            /*
             * Decode sigconv output (ASCII samples values)
             */
            sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
            listing = fopen(ch, "r");
            if (listing == NULL)
            {
                fprintf(stderr, "Can't check sigconv output.\n");
                exit(1);
            }

            for(;;)
            {
                if (fgets(ligne, MAXLEN, listing) == NULL)
                {
                    fclose(listing);
                    break;
                }
                if (Nbv >= MAXVALS)
                {
                    fprintf(stderr, "Maximum number of points
reached. %d.\n", MAXVALS);
                    exit(1);
                }
                valE[Nbv++] = atoi(ligne);
            }
            /*
             * Move LTA last samples of the previous E component at the
beginning of the array
             */
            ptval_d = &(valE[0]);
            ptval_o = &(valE[Nbv-1]); ptval_o -= (LTA*previous->frequency);
            for (i=0; i<LTA*previous->frequency; i++)
                *ptval_d++ = *ptval_o++;

            Nbv = LTA*previous->frequency;

        }
    /*
     *      Now, read (and keep) all values of current E component
     */
    if (strlen(current->filE) != 0)
    {
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
            fprintf(stderr, "Can't cd to temp dir: %s.\n", TEMP_DIR);
            exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
            fprintf(stderr, "Can't create batch file: %s.\n", BATCHFILE);
            exit(1);
        }

        fprintf(batch, "@ECHO OFF\n", DATA);
        fprintf(batch, "cd %s\n", DATA);
        fprintf(batch, "cd %s\n", current->rdir);
        fprintf(batch, "cd %s\n", current->nsdir);
        fprintf(batch, "sigconv ascii e %s %s\\%s > %s\\NULL\n",
            current->filE, TEMP_DIR, TEMP_FILE, TEMP_DIR);
        fclose(batch);

        if (system(BATCHFILE) == -1)
        {
            fprintf(stderr, "Can't execute batch file: %s \n", BATCHFILE);
            exit(1);
        }
        /*
         * Read samples values
         */
        /*
         * Decode sigconv output (ASCII samples values)
         */
        sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
        listing = fopen(ch, "r");
        if (listing == NULL)
        {
            fprintf(stderr, "Can't check sigconv output.\n");
            exit(1);
        }
        for(;;)
        {
            if (fgets(ligne, MAXLEN, listing) == NULL)
            {
                fclose(listing);
                break;
            }
            if (Nbv >= MAXVALS)
            {
                fprintf(stderr, "Maximum number of points
reached. %d.\n", MAXVALS);
                exit(1);
            }
            valE[Nbv++] = atoi(ligne);
        }
    }

    Nbv = 0;
    valN = (short *) calloc (NbValues, sizeof(short));
    if (valN == NULL)
    {
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
    }

    if (previous != NULL)
        if (strlen(previous->filN) != 0)
        {
            /*
             * Read previous N component
             */
            Status = chdir(TEMP_DIR);
            if (Status != 0)
            {
                fprintf(stderr, "Can't cd to temp dir: %s.\n",
TEMP_DIR);
                exit(1);
            }

            batch = fopen(BATCHFILE, "w");
            if (batch == NULL)
            {
                fprintf(stderr, "Can't create batch file: %s.\n",
BATCHFILE);
                exit(1);
            }

            fprintf(batch, "@ECHO OFF\n", DATA);
            fprintf(batch, "cd %s\n", DATA);
            fprintf(batch, "cd %s\n", previous->rdir);
            fprintf(batch, "cd %s\n", previous->nsdir);
            fprintf(batch, "sigconv ascii e %s %s\\%s > %s\\%s\n",
                previous->filN,
                TEMP_DIR, TEMP_FILE,
                TEMP_DIR, NULL_FILE);
            fclose(batch);

            if (system(BATCHFILE) == -1)
            {
```

```
                fprintf(stderr, "Can't execute batch file: %s.\n",
BATCHFILE);
                exit(1);
            }
            /*
             * Read samples values
             */
            /*
             * Decode sigconv output (ASCII samples values)
             */
            sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
            listing = fopen(ch, "r");
            if (listing == NULL)
            {
                fprintf(stderr, "Can't check sigconv output.\n");
                exit(1);
            }

            for(;;)
            {
                if (fgets(ligne, MAXLEN, listing) == NULL)
                {
                    fclose(listing);
                    break;
                }
                if (Nbv >= MAXVALS)
                {
                    fprintf(stderr, "Maximum number of points
reached: %d.\n", MAXVALS);
                    exit(1);
                }
                valN[Nbv++] = atoi(ligne);
            }
            /*
             * Move LTA last samples of the previous N component at the
beginning of the array
             */
            ptval_d = &(valN[0]);
            ptval_o = &(valN[Nbv-1]); ptval_o -= (LTA*previous->frequency);
            for (i=0; i<LTA*previous->frequency; i++)
                *ptval_d++ = *ptval_o++;

            Nbv = LTA*previous->frequency;

        }
    /*
     *      Now, read (and keep) all values of current N component
     */
    if (strlen(current->filN) != 0)
    {
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
            fprintf(stderr, "Can't cd to temp dir: %s.\n", TEMP_DIR);
            exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
            fprintf(stderr, "Can't create batch file: %s.\n", BATCHFILE);
            exit(1);
        }

        fprintf(batch, "@ECHO OFF\n", DATA);
        fprintf(batch, "cd %s\n", DATA);
        fprintf(batch, "cd %s\n", current->ndir);
        fprintf(batch, "cd %s\n", current->nsdir);
        fprintf(batch, "sigconv -ascii -e %s %s\\%s > %s\\NULL\n",
                current->filN, TEMP_DIR, TEMP_FILE, TEMP_DIR);
        fclose(batch);

        if (system(BATCHFILE) == -1)
        {
            fprintf(stderr, "Can't execute batch file: %s.\n", BATCHFILE);
            exit(1);
        }
        /*
         * Read samples values
         */
        /*
         * Decode sigconv output (ASCII samples values)
         */
        sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
        listing = fopen(ch, "r");
        if (listing == NULL)
        {
            fprintf(stderr, "Can't check sigconv output.\n");
            exit(1);
        }
        for(;;)
        {
            if (fgets(ligne, MAXLEN, listing) == NULL)
            {
                fclose(listing);
                break;
            }
            if (Nbv >= MAXVALS)
            {
                fprintf(stderr, "Maximum number of points
reached: %d.\n", MAXVALS);
                exit(1);
            }
            valN[Nbv++] = atoi(ligne);
        }
    }
    /*
     * When earthquakes are detected, check if the last one ends after (including post-event
interval)
     * the current component. If so, it is necessary to read three next samples, in order not to
lose
     * information.
     */
    if (NbQuakes > 0)
        if  (quake[NbQuakes-1].end > NbValues)
            if (next != NULL)
            {
                /*
                 * Copy current Z component
                 */
                for(int i=0; i<NbValues; i++)
                    valsign[i] = valZ[i];
                /*
                 * Free previous valZ array
                 */
                free(valZ);
                int oldnbv = NbValues;

                /*
                 * Next Z component
                 */
                Status = chdir(TEMP_DIR);
                if (Status != 0)
                {
                    fprintf(stderr, "Can't cd to temp dir: %s.\n",
TEMP_DIR);
                    exit(1);
                }

                batch = fopen(BATCHFILE, "w");
                if (batch == NULL)
                {
                    fprintf(stderr, "Can't create batch file: %s.\n",
BATCHFILE);
                    exit(1);
                }

                fprintf(batch, "@ECHO OFF\n", DATA);
                fprintf(batch, "cd %s\n", DATA);
```

```
fprintf(batch, "cd %s\n", next->ndir);
fprintf(batch, "cd %s\n", next->nsdir);
fprintf(batch, "sigconv ascii e %s %s\\%s > %s\\NULL\n",
        next->filZ, TEMP_DIR, TEMP_FILE, TEMP_DIR);
fclose(batch);

if (system(BATCHFILE) == -1)
{
        fprintf(stderr, "Can't execute batch file: %s.\n",
BATCHFILE);
        exit(1);
}
/*
 * Read samples values
 */
/*
 * Decode sigconv output (ASCII samples values)
 */
sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
listing = fopen(ch, "r");
if (listing == NULL)
{
        fprintf(stderr, "Can't check sigconv output.\n");
        exit(1);
}
for(;;)
{
        if (fgets(ligne, MAXLEN, listing) == NULL)
        {
                fclose(listing);
                break;
        }
        if (NbValues >= MAXVALS)
        {
                fprintf(stderr, "Maximum number of points
reached: %d!!!\n", MAXVALS);
                exit(1);
        }
        valsign[NbValues++] = atoi(ligne);
}
valZ = (short *) calloc (NbValues, sizeof(short));
if (valZ == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}
for (i=0; i< NbValues; i++)
        valZ[i] = valsign[i];

/*
 * Copy current E component
 */
for (i=0; i<oldnbv; i++)
        valsign[i] = valE[i];

/*
 * Free previous valE array
 */
free(valE);
if ( strlen(next->filE) != 0)
{
        /*
         * Read next E component
         */
        NbValues = oldnbv;
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to temp dir. %s\n",
TEMP_DIR);
                exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
                fprintf(stderr, "Can't create batch
file: %s.\n", BATCHFILE);
                exit(1);
        }

        fprintf(batch, "@ECHO OFF\n", DATA);
        fprintf(batch, "cd %s\n", DATA);
        fprintf(batch, "cd %s\n", next->ndir);
        fprintf(batch, "cd %s\n", next->nsdir);
        fprintf(batch, "sigconv ascii e %s %s\\%s
> %s\\NULL\n",
                next->filE, TEMP_DIR, TEMP_FILE, TEMP_DIR);
        fclose(batch);

        if (system(BATCHFILE) == -1)
        {
                fprintf(stderr, "Can't execute batch
file: %s.\n", BATCHFILE);
                exit(1);
        }
        /*
         * Read samples values
         */
        /*
         * Decode sigconv output (ASCII samples values)
         */
        sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
        listing = fopen(ch, "r");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't check sigconv
output.\n");
                exit(1);
        }
        for(;;)
        {
                if (fgets(ligne, MAXLEN, listing) == NULL)
                {
                        fclose(listing);
                        break;
                }
                if (NbValues >= MAXVALS)
                {
                        fprintf(stderr, "Maximum number of
points reached: %d!!!\n", MAXVALS);
                        exit(1);
                }
                valsign[NbValues++] = atoi(ligne);
        }
}
else
{
        for (i=oldnbv; i<NbValues; i++)
                valsign[i] = 0;
}

valE = (short *) calloc (NbValues, sizeof(short));
if (valE == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}
for (i=0; i< NbValues; i++)
        valE[i] = valsign[i];

/*
 * Copy current N component
 */
for (i=0; i<oldnbv; i++)
        valsign[i] = valN[i];

/*
 * Free previous valN array
 */
```

```
free(valN);
if ( strlen(next->filN) != 0)
{
        /*
         * Read next N component
         */
        NbValues = oldnbv;
        Status = chdir(TEMP_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to temp dir: %s \n",

                exit(1);
        }

        batch = fopen(BATCHFILE, "w");
        if (batch == NULL)
        {
                fprintf(stderr, "Can't create batch

                exit(1);
        }

        fprintf(batch, "@ECHO OFF\n", DATA);
        fprintf(batch, "cd %s\n", DATA);
        fprintf(batch, "cd %s\n", next->ndir);
        fprintf(batch, "cd %s\n", next->nsdir);
        fprintf(batch, "sigconv ascii e %s %s\\%s

                next->filN, TEMP_DIR, TEMP_FILE, TEMP_DIR);
        fclose(batch);

        if (system(BATCHFILE) == -1)
        {
                fprintf(stderr, "Can't execute batch

                exit(1);
        }
        /*
         * Read samples values
         */
        /*
         * Decode sigconv output (ASCII samples values)
         */
        sprintf(ch, "%s\\%s", TEMP_DIR,TEMP_FILE);
        listing = fopen(ch, "r");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't check sigconv

                exit(1);
        }
        for(;;)
        {
                if (fgets(ligne, MAXLEN, listing) == NULL)
                {
                        fclose(listing);
                        break;
                }
                if (NbValues >= MAXVALS)
                {
                        fprintf(stderr, "Maximum number of

                        exit(1);
                }
                valsign[NbValues++] = atoi(ligne);
        }
}
else
{
        for (i=oldnbv; i<NbValues; i++)
                valsign[i] = 0;
}

valN = (short *) calloc (NbValues, sizeof(short));
if (valN == NULL)
{
        fprintf(stderr, "Can't allocate data for array.\n");
        exit(1);
}
for (i=0; i< NbValues; i++)
        valN[i] = valsign[i];
                }

long tempo;
struct tm *pointe;

char *AsciiMonth[13]=
{
        "",
        "jan",
        "feb",
        "mar",
        "apr",
        "may",
        "jun",
        "jul",
        "aug",
        "sep",
        "oct",
        "nov",
        "dec"
};

/*
 * Generation of index (.ndx) files
 */
for (i=0; i<NbQuakes; i++)
{
        /*
         * Pointer to the beginning of earthquake
         */
        tempo = current->begin;
        tempo += quake[i].begin / current->frequency;
        /*
         * Pointer must reflect the actual begin (must substract D_PRE seconds)
         */
        tempo -= (int) D_PRE;
        pointe = localtime(&tempo);

        Status = chdir(OUT_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to earthquake directory: %s.\n", OUT_DIR);
                exit(1);
        }

        /*
         * Directory for the year, eg. 2002
         */
        char YearDir[MAXLEN];
        sprintf(YearDir, "%04d", 1900+pointe->tm_year);
        Status = chdir(YearDir);
        if (Status != 0)
                if( _mkdir(YearDir) != 0 )
                {
                        fprintf(stderr, "Can't create dir %s\n", YearDir);
                        exit(1);
                }
                else
                {
                        Status = chdir(YearDir);
                        if (Status != 0)
                        {
                                fprintf(stderr, "Can't cd to dir %s\n", YearDir);
                                exit(1);
                        }
                }

        /*
```

```
 * Subdirectory for the month, something like, eg. jul2002.
 */
char SubDir[MAXLEN];
sprintf(SubDir, "%s%04d", AsciiMonth[1+pointe->tm_mon], 1900+pointe->tm_year);
Status = chdir(SubDir);
if (Status != 0)
        if ( _mkdir(SubDir) != 0 )
        {
                fprintf(stderr, "Can't create subdir %s\n", SubDir);
                exit(1);
        }
        else
        {
                Status = chdir(SubDir);
                if (Status != 0)
                {
                        fprintf(stderr, "Can't cd to subdir %s\n", SubDir);
                        exit(1);
                }
        }

sprintf(ch, "%02d%02d%02d%02d.ndx",
        1+pointe->tm_mon,
        pointe->tm_mday,
        pointe->tm_hour,
        pointe->tm_min);

listing = fopen(ch, "w");
if (listing == NULL)
{
        fprintf(stderr, "Can't create file %02d%02d%02d%02d.sis.",
        1+pointe->tm_mon,
        pointe->tm_mday,
        pointe->tm_hour,
        pointe->tm_min);
        Fine = false;
        continue;
}

int NbBlocs = quake[i].end-quake[i].begin+1;
/*
 * Fractional part of seconds
 */
int     PremBloc = 1;

if (NbBlocs%BLOCKSIZE == 0)
        NbBlocs /= BLOCKSIZE;
else
{
        NbBlocs /= BLOCKSIZE; ++NbBlocs;
}

float Seconde; int tagada;
tagada = quake[i].begin % current->frequency;
Seconde = (float) tagada;
Seconde /= current->frequency;
Seconde += pointe->tm_sec;

fprintf(listing, "%7sZ%6d%6d%6d%8.1f  %02d:%02d:%04d %02d:%02d:%06.3f\n",
        STATION,
        quake[i].end-quake[i].begin+1,
        PremBloc,
        NbBlocs,
        (float) info[FileIndex].frequency,
        pointe->tm_mday,
        1+pointe->tm_mon,
        1900+pointe->tm_year,
        pointe->tm_hour,
        pointe->tm_min,
        Seconde
);
sprintf(Tamp, "Z COMPONENT - FILE %s\\%s\\%s",
        current->ndir,
        current->nsdir,
        current->filZ);
fprintf(listing, "%60s\n", Tamp);

PremBloc += NbBlocs;
fprintf(listing, "%7sN%6d%6d%6d%8.1f  %02d:%02d:%04d %02d:%02d:%06.3f\n",
        STATION,
        quake[i].end-quake[i].begin+1,
        PremBloc,
        NbBlocs,
        (float) info[FileIndex].frequency,
        pointe->tm_mday,
        1+pointe->tm_mon,
        1900+pointe->tm_year,
        pointe->tm_hour,
        pointe->tm_min,
        Seconde
),
sprintf(Tamp, "N COMPONENT - FILE %s\\%s\\%s",
        current->ndir,
        current->nsdir,
        current->filN);
fprintf(listing, "%60s\n", Tamp);

PremBloc += NbBlocs;
fprintf(listing, "%7sE%6d%6d%6d%8.1f  %02d:%02d:%04d %02d:%02d:%06.3f\n",
        STATION,
        quake[i].end-quake[i].begin+1,
        PremBloc,
        NbBlocs,
        (float) info[FileIndex].frequency,
        pointe->tm_mday,
        1+pointe->tm_mon,
        1900+pointe->tm_year,
        pointe->tm_hour,
        pointe->tm_min,
        Seconde
);
sprintf(Tamp, "E COMPONENT - FILE %s\\%s\\%s",
        current->ndir,
        current->nsdir,
        current->filE);
fprintf(listing, "%60s\n", Tamp);

fclose(listing);
}

/*
 * Generation of earthquake (.sis) files
 */
for (i=0; i<NbQuakes; i++)
{
        /*
         * Pointer to the beginning of earthquake
         */
        tempo = current->begin;
        tempo += quake[i].begin / current->frequency;
        /*
         * Pointer must reflect the actual begin (must substract D_PRE seconds)
         */
        tempo -= (int) D_PRE;
        pointe = localtime(&tempo);

        sprintf(ch, "%02d%02d%02d%02d.sis",
                1+pointe->tm_mon,
                pointe->tm_mday,
                pointe->tm_hour,
                pointe->tm_min);

        int fich = _open( ch, _O_BINARY | _O_WRONLY | _O_CREAT, _S_IREAD | _S_IWRITE ),


        if (fich == -1)
        {
                fprintf(stderr, "Can't create file %02d%02d%02d%02d.sis.",
                1+pointe->tm_mon,
                pointe->tm_mday,
```

```
            pointe->tm_hour,
            pointe->tm_min);
            Fine = false;
            continue;
    }

    int NbBlocs = quake[i].end-quake[i].begin+1;
    /*
     * Fractional part of seconds
     */
    int       PremBloc = 1;

    if (NbBlocs%BLOCKSIZE == 0)
            NbBlocs /= BLOCKSIZE;
    else
    {
            NbBlocs /= BLOCKSIZE; ++NbBlocs;
    }

    int NbOctets = NbBlocs * BLOCKSIZE * sizeof(short);
    int j, k;
    char *Tampon = (char *) malloc(NbOctets);
    short val; char *pt, *ptval;

    if (Tampon == NULL)
    {
            fprintf(stderr, "Can't allocate data for I/O array.\n");
            exit(1);
    }

    pt = Tampon; k = 0;
    for (j=quake[i].begin; j<=quake[i].end; j++, k+=2)
    {
            val = valZ[j]; ptval = (char *) &val;
            *pt++ = *ptval++;
            *pt++ = *ptval++;
    }
    for (; k< NbOctets; k++)
            *pt++ = '\0';

    /*
     * Write binary values of Z-component on disk
     */
    int ecrits = _write( fich, Tampon, NbOctets);
    if (ecrits != NbOctets)
    {
            fprintf(stderr, "Write error.\n");
            exit(1);
    }

    pt = Tampon; k = 0;
    for (j=quake[i].begin; j<=quake[i].end; j++, k+=2)
    {
            val = valN[j]; ptval = (char *) &val;
            *pt++ = *ptval++;
            *pt++ = *ptval++;
    }
    for (; k< NbOctets; k++)
            *pt++ = '\0';

    /*
     * Write binary values of N-component on disk
     */
    ecrits = _write( fich, Tampon, NbOctets);
    if (ecrits != NbOctets)
    {
            fprintf(stderr, "Write error.\n");
            exit(1);
    }

    pt = Tampon; k = 0;
    for (j=quake[i].begin; j<=quake[i].end; j++, k+=2)
    {
            val = valE[j]; ptval = (char *) &val;
            *pt++ = *ptval++;
            *pt++ = *ptval++;
    }
    for (; k< NbOctets; k++)
            *pt++ = '\0';

    /*
     * Write binary values of E-component on disk
     */
    ecrits = _write( fich, Tampon, NbOctets);
    if (ecrits != NbOctets)
    {
            fprintf(stderr, "Write error.\n");
            exit(1);
    }

    free(Tampon);
    _close(fich);
}

/*
 * Génération des fichiers .qak
for (i=0; i<NbQuakes; i++)
{
        tempo = current->begin;
        tempo += quake[i].begin / current->frequency;
        pointe = localtime(&tempo);

        sprintf(ch, "%02d%02d%02d%02d.qak",
                1+pointe->tm_mon,
                pointe->tm_mday,
                pointe->tm_hour,
                pointe->tm_min);

        listing = fopen(ch, "w");
        if (listing == NULL)
        {
                fprintf(stderr, "Impossible de creer le fichier listing!!!");
                exit(1);
        }

        for (int j=quake[i].begin; j<=quake[i].end; j++)
                        fprintf(listing, "%d\t%d\t%d\t%d\n", j, valZ[j], valN[j],
valE[j]);

        fclose(listing);
}
 */

/*
 * Time Stamp is created to avoid redundant detections
 */
if (Fine == true)
{
        Status = chdir(DATA);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", DATA);
                exit(1);
        }
        Status = chdir(current->ndir);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", current->ndir);
                exit(1);
        }
        Status = chdir(current->nsdir);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", current->nsdir);
                exit(1);
        }
        listing = fopen(TIME_STAMP, "w");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't create time stamp.\n");
```

```
                exit(1);
        }
        else
                fclose(listing);
}

switch (NbQuakes)
{
case 0:
        Status = chdir(DATA);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", DATA);
                exit(1);
        }
        Status = chdir(current->ndir);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory. %s \n", current->ndir);
                exit(1);
        }
        Status = chdir(current->nsdir);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory. %s.\n", current->nsdir);
                exit(1);
        }
        listing = fopen(TIME_STAMP, "w");
        if (listing == NULL)
        {
                fprintf(stderr, "CAn't create time stamp.\n");
                exit(1);
        }
        else
                fclose(listing);
        break;
default:
        /*
         * No time stamp written if next component does not yet exist, and last
         * earthquake ends after current compinent
         */
        if (quake[NbQuakes-1].end > NbValues)
                if (next == NULL)
                        break;
        Status = chdir(DATA);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", DATA);
                exit(1);
        }
        Status = chdir(current->ndir);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", current->ndir);
                exit(1);
        }
        Status = chdir(current->nsdir);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to directory: %s.\n", current->nsdir);
                exit(1);
        }
        listing = fopen(TIME_STAMP, "w");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't create time stamp.\n");
                exit(1);
        }
        else
                fclose(listing);
        break;
};

if (valabs != NULL)
        free(valabs);
if (LowPass != NULL)
        free(LowPass);
if (HighPass != NULL)
        free(HighPass);
if (valZ != NULL)
        free(valZ);
if (valE != NULL)
        free(valE);
if (valN != NULL)
        free(valN);

Status = chdir(orig);
if (Status != 0)
{
        fprintf(stderr, "Can't get back to original directory: %s.\n", orig);
        exit(1);
}
}
```

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Constants.h"
#include "procedures.h"
#include <io.h>
#include <direct.h>

int                NbFiles=0;
struct    information info[20];
FILE      *log;

 *
 * Automatic detection of earthquakes.
 *
 * Author  Pierre Lebellegard, Centre IRD de Nouméa, New Caledonia
 *
 * The purpose of this program is to make an automatic detection of earthquakes
 * occurring in continuous input data.
 */
int main(int argc, char* argv[])
{
        struct    _finddata_t        sdir;     long      h_sdir;
        struct    _finddata_t        hour;     long      h_hour;
        char      Directory[MAXLEN];
        char      ch2[MAXLEN];
        int               Status;
        int               i, nbf;
        int               prev_year, prev_day;
        int               curr_year, curr_day;
        struct    tm *ptime;
        time_t    yesterday, today;
        char      filelist[6][MAXLEN];
        FILE      *listing;
        char      *AsciiMonth[13]=
        {
                "",
                "jan",
                "feb",
                "mar",
                "apr",
                "may",
                "jun",
                "jul",
                "aug",
                "sep",
                "oct",
                "nov",
                "dec"
        };
        /*
         * Open log file for writing. Name of log file is made from date: yymmddhhmmss.txt
         */
        Status = chdir(LOG_DIR);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to log file directory: %s.\n", LOG_DIR);
                exit(1);
        }

        time( &today ); yesterday = today - NBDAYS;

        /* Obtain coordinated universal time: */
        ptime = gmtime( &today );

        curr_year = ptime->tm_year; curr_year += 1900;
        curr_day = ptime->tm_yday; ++curr_day;

        sprintf(ch2, "%s%04d.txt",
                AsciiMonth[1+ptime->tm_mon],
                1900+ptime->tm_year
                );
```

```
log = fopen(ch2, "a+");
if (log == NULL)
{
        fprintf(stderr, "Can't create log file: %s.\n", ch2);
        exit(1);
}

/*
 * Start scanning directory
 */
Status = chdir(DATA);
if (Status != 0)
{
        fprintf(stderr, "Can't cd to data directory: %s.\n", DATA);
        exit(1);
}

for (;;)
{
        yesterday += DAY;
        ptime = gmtime( &yesterday );
        prev_year = ptime->tm_year; prev_year += 1900;
        prev_day = ptime->tm_yday; ++prev_day;

        if (prev_year > curr_year)
                break;
        else if ( (prev_year == curr_year)
                &&                (prev_day > curr_day)
                )
                break;
        else
        {
                /*
                 * Scanning each subdirectory
                 */
                sprintf(Directory, "%04d-%03d", prev_year, prev_day);
                Status = chdir(Directory);
                if (Status != 0)
                        continue;

                /*
                 * Two digit names only
                 */
                if ( (h_sdir=_findfirst("??", &sdir)) == -1L )
                {
                        fprintf(stderr, "Empty sub-directory: %s.\n", sdir.name);
                        exit(0);
                }

                else
                {
                        /*
                         * Scanning remaining subdirectories
                         */
                        while ( _findnext(h_sdir, &sdir) == 0)
                        {
                                if (!strcmp(sdir.name, ".."))
                                        continue;
                                /*
                                 * Scanning hourly subdirectories
                                 */
                                Status = chdir(sdir.name);
                                if (Status != 0)
                                {
                                        fprintf(stderr, "Can't cd to: %s.\n",
sdir.name);
                                        exit(1);
                                }

                                /*
                                 * Examine and detection are only made once. To do so,
                                 * existence of (empty) file DETECTE.STP is checked.
                                 */
                                if( (_access( TIME_STAMP, 0 )) != -1 )
```

```
tc  >s  n"  sd.r name);

1L )

subdirectory: %s\\%s.\n", Directory, sdir.name);
subdirectory: %s\\%s.\n", Directory, sdir.name),

(irectory: %s. n", DATA);

directory: %s.·n", Directory);

directory: %s.\n", sdir.name);

Stamp. n");

directory %s.\n", sdir.name);
```

```
{
/*
 * Get back to upper directory
 */
        Status = chdir("..");
        if (Status != 0)
        {
                fprintf(stderr, "Can't get back

                exit(1);
        }
        continue;
}

NbFiles = 0;
/*
 * Only 8 bytes names - 00c as a suffix.
 */
if ( (h_hour=_findfirst("????????.00c", &hour)) == -

{
        fprintf(stderr, "Empty hourly

        fprintf(log, "Empty hourly

        fflush(log);

        Status = chdir(DATA);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to

                exit(1);
        }
        Status = chdir(Directory);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to

                exit(1);
        }
        Status = chdir(sdir.name);
        if (Status != 0)
        {
                fprintf(stderr, "Can't cd to

                exit(1);
        }
        listing = fopen(TIME_STAMP, "w");
        if (listing == NULL)
        {
                fprintf(stderr, "Can't create Time

                exit(1);
        }
        else
                fclose(listing);

        /*
         * Get back to upper directory
         */
        Status = chdir("..");
        if (Status != 0)
        {
                fprintf(stderr, "Can't get back to

                exit(1);
        }
        continue;
}
else
{
        /*
```



```
files (2x3 components)

hour.name);

subdirectory: %s\\%s doesn't contain 6 files.\n",

subdirectory: %s\\%s doesn't contain 6 files.\n",

get back to directory %s.\n", sdir.name);

ame time. Given name: CALcxmmx.00c,

component, and c=3 for E component, mm

hourly subdirectory: %s\\%s don't have the same offset.\n",

sdir.name);

hourly subdirectory: %s\\%s don't have the same offset.\n",

sdir.name);

fopen(TIME_STAMP, "w");

"Can't create Time Stamp.\n");
```

```
 * Each hourly subdirectory must contain 6

 */
nbf = 1;
strcpy(filelist[nbf-1], hour.name);
while ( _findnext(h_hour, &hour) == 0)
{
        if (!strcmp(hour.name, ".."))
                continue;
        if (++nbf < 7)
                strcpy(filelist[nbf-1],

}
_findclose(h_hour);

if (nbf != 6)
{
        fprintf(stderr, "Hourly

                Directory, sdir.name);
        fprintf(log, "Hourly

                Directory, sdir.name);
        fflush(log );

        /*
         * Get back to upper directory
         */
        Status = chdir("..");
        if (Status != 0)
        {
                fprintf(stderr, "Can't

                exit(1);
        }
        continue;
}

/*
 * Three components' files must begin at the

 * with c=1 for Z component, c=2 for N

 * must be equal for the three components.
 */
char buf[10]; int i;
buf[0] = filelist[0][5];
buf[1] = filelist[0][6];
buf[2] = '\0';
int temp = atoi(buf);
for (i=1; i<6; i++)
{
        buf[0] = filelist[i][5];
        buf[1] = filelist[i][6];
        buf[2] = '\0';
        if (atoi(buf) != temp)
        {
                fprintf(stderr, "Files in

                        Directory,

                fprintf(log, "Files in

                        Directory,

                fflush(log);

                listing =

                if (listing == NULL)
                {
                        fprintf(stderr,

                        exit(1);
```

```
        fclose(listing);


"Can't cd to directory: %s.\n", DATA);

chdir(Directory);

"Can't cd to directory: %s.\n", Directory);

chdir(sdir.name);

"Can't cd to directory: %s.\n", sdir.name);


&NbFiles, &(info[0]));

hour.name, &NbFiles, &(info[0]));

comparaison)

&(info[0]));

next:

directory %s.\n", sdir.name);
```

```
                }
                else
                        DATA),

                Status = chdir(DATA);
                if (Status != 0)
                {
                        fprintf(stderr,

                        exit(1);
                }
                Status =

                if (Status != 0)
                {
                        fprintf(stderr,

                        exit(1);
                }
                Status =

                if (Status != 0)
                {
                        fprintf(stderr,

                        exit(1);
                }
                        goto next;
                        }
        }

        h_hour=_findfirst("????????.00c", &hour);
        examine(Directory, sdir.name, hour.name,

        while ( _findnext(h_hour, &hour) == 0)
        {
                if (!strcmp(hour.name, ".."))
                        continue;
                examine(Directory, sdir.name,

        }
        _findclose(h_hour);
}

/*
 * Each directory should contain 2x3 files (2x3
 */
nbf = NbFiles;
for (i=0; i<nbf; i++)
        detect(Directory, sdir.name, i, &NbFiles,

/*
 * Get back to upper directory
 */
Status = chdir("..");
if (Status != 0)
{
        fprintf(stderr, "Can't get back to

        exit(1);
}
_findclose(h_sdir);
}

/*
 * Get back to main directory
 */
Status = chdir("..");
if (Status != 0)
{
        fprintf(stderr, "Can't get back to data directory: %s.\n",

        exit(1);
}
return 0;
}
```