Mise en place du programme de lecture de données readData

Comme nous avons vu dans les documents relatifs à l'auto codage, nous avions besoin d'un programme qui puisse lire les données d'entrées récupérées depuis Matlab. Nous avons donc créé un programme appelé readData. Nous expliquerons dans la suite son fonctionnement.

Le projet C++ relatif au test de ce programme se situe dans le dossier $TestReadData$.

I – Création du projet

Nous créons via le logiciel code blocks, un nouveau projet appelé $TestReadData$. Ce projet contiendra 5 fichiers :

- $main.cpp$, le programme principal du projet.
- $saveData.h$, le $header\ file$ du programme de sauvegarde de données déjà réalisé.
- $saveData.cpp$, le $source\ file$ du programme de sauvegarde de données déjà réalisé.
- $readData.cpp$, le $header\ file$ du programme de lecture de données.
- $readData.h$, le $header\ file$ du programme de lecture de données.

II – Détail des nouveaux fichiers

II.1 - $readData.h$

Voici le $header\ file$ du programme de lecture de données :

```
#ifndef READDATA_H_INCLUDED
#define READDATA_H_INCLUDED
#include <string>
#include <iostream> //Pour afficher les messages (cout)
#include <fstream> //Pour la lecture des fichier en C++
#include <sstream>
#define TAILLEMAX 30000

using namespace std;

float* readData(long double tableau[TAILLEMAX], string filename_save);

#endif // READDATA_H_INCLUDED
```

Les $\#include$ servent respectivement à :

- $string$: charger la librairie relative à la manipulation de données de type string.
- $iostream$: charger la libraire relative à l'affichage des messages (« cin » ; « cout » ; etc).
- $fstream$: charger la librairie relative à la lecture de fichiers.
- $sstream$: charger la librairie pour convertir des données d'un type A à un type B (non utilisé dans la forme finale du programme).

On définit également une variable globale $TAILLEMAX$ qui correspond à la taille maximale des tableaux que l'on va utiliser, en effet travailler avec des tableaux de taille variable (allocations dynamique de mémoire) aurait été beaucoup plus long à mettre en place. De plus cela n'aurait rien changé dans la finalité du programme, nous avons donc choisi de faire une allocation fixe de taille

30000. Au niveau du matlab les entrée et les sorties sont des tableaux de $tfin * hev_{dt} + 1 = 20001$. Les simulations durent $200s$ avec une pas de $hev_{dt} = 0.01s$. Nous avons donc choisi arbitrairement 3000 pour que les tableaux rentrent bien dedans.

Ensuite on charge la bibliothèque standard pour les espaces de noms.

On déclare finalement notre fonction, on a donc une fonction qui renvoie un $float$, nous aurions également pu choisir comme type de sortie $void$ puisque nous ne retournerons rien après la fonction, la fonction modifiera seulement un tableau en interne.

Les arguments de la fonction sont les suivants :

- `long double tableau[TAILLEMAX]` : tableau d'entrée pouvant avoir une taille maximale de $TAILLEMAX$, le format choisi est $long\ double$ pour avoir un maximum de précision.
- `string filename_save` : nom du fichier à lire, avec le format "$nom\_fichier.txt$".

II.2 - *readData.cpp*

Voici le $source\ file$ du programme de lecture de données :

```
#include "readData.h"
#include <iostream> //Pour afficher les messages (cout)
#include <fstream> //Pour la lecture des fichier en C++
#include <string>
#include <sstream>

using namespace std;

float* readData(long double tableau[TAILLEMAX], string filename_save){

    ifstream fichierlu(filename_save.c_str());

    if(fichierlu)
    {
    //L'ouverture s'est bien passée, on peut donc lire

        int i = 0;
        long double nombre;
        while(fichierlu >> nombre) //Tant qu'on n'est pas à  la fin, on lit
        {
            tableau[i] = nombre;
            i++;
        }
    }
    else
    {
        cout << "ERREUR: Impossible d'ouvrir le fichier en lecture." <<
endl;
    }
return 0;
}
```

Le début du fichier est le même que pour le $header\ file$ jusqu'à la définition de la fonction.

Dans un premier temps nous chargeons le fichier à lire via la commande $ifstream$. Une fois le fichier chargé on vérifie que cela a bien fonctionné grâce au if, si le fichier n'a pas été bien chargé, on envoie un message d'erreur. Si le fichier a bien été ouvert, on charge dans une variable au format $long\ double$. Ensuite on parcourt tout le fichier en prenant chaque mot (un mot correspondra ici à un nombre à virgule), et on ajoute ce nombre à notre tableau d'entrée, et on recommence jusqu'à ce que tout le fichier soit parcouru. Une fois cela fait on quitte la fonction.

II.3 - $main.cpp$

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "saveData.h"
#include "readData.h"

using namespace std;

int main()
{
    int taille = 20001;
    long double test_tab[TAILLEMAX];

    readData(test_tab, "testprojet.txt");
    saveData(taille ,test_tab,"sauvegardetestprojet.txt");
    cout << "Hello World";
return 0;
}
```

Dans ce $main$, on définit la taille du futur tableau à savoir 20001 comme calculé précédemment, ensuite on initialise notre tableau. Après on appelle la fonction de lecture qui va lire un fichier créé au préalable et une fois le fichier lu on l'enregistre avec la fonction $saveData$.

Attention, des modifications ont aussi été effectuées sur les fichiers $saveData$. Les modifications sont faites uniquement au niveau des formats des tableaux qui passent en $long\ double$.

III – Résultats

Voici les résultats obtenus, à gauche le fichier d'entrée et à droite le fichier de sortie :

3.70152166732726e-11	3.70152e-011
3.70263846832323e-11	3.70264e-011
3.70412781087925e-11	3.70413e-011
3.70561767765955e-11	3.70562e-011
3.70710814370609e-11	3.70711e-011
3.70859920924487e-11	3.7086e-011
3.71009087451702e-11	3.71009e-011
3.71158313976375e-11	3.71158e-011
3.71307600522640e-11	3.71308e-011
3.71456947114636e-11	3.71457e-011
3.71606353776518e-11	3.71606e-011
3.71755820532444e-11	3.71756e-011
3.71905347406586e-11	3.71905e-011
3.72054934423126e-11	3.72055e-011
3.72204581606253e-11	3.72205e-011
3.72354288980167e-11	3.72354e-011

Figure 1 - Résultats obtenus

On remarque clairement une différence entre les deux valeurs, ceci est dû au format des données récupérées qui est plus petits que le format sous Matlab. Quand nous lirons les données

d’entrée lors des autos codages nous aurons donc cette troncature qui sera faite. Ce qui n’est pas très grave au vu du niveau de troncature qui est fait au bout d’un nombre assez conséquents de chiffres après la virgule.