Présentation de TIPE

Étude et optimisation d'un outil d'ingénierie du bâtiment

Thomas CREUSET

numéro de candidat : 11909

Sommaire

Choix du projet

Choix du projet :

La construction en ville : un enjeu pour les ingénieurs

Objectifs de l'ingénieur :

- Stabilité
- Durabilité
- Accessibilité
- Autres

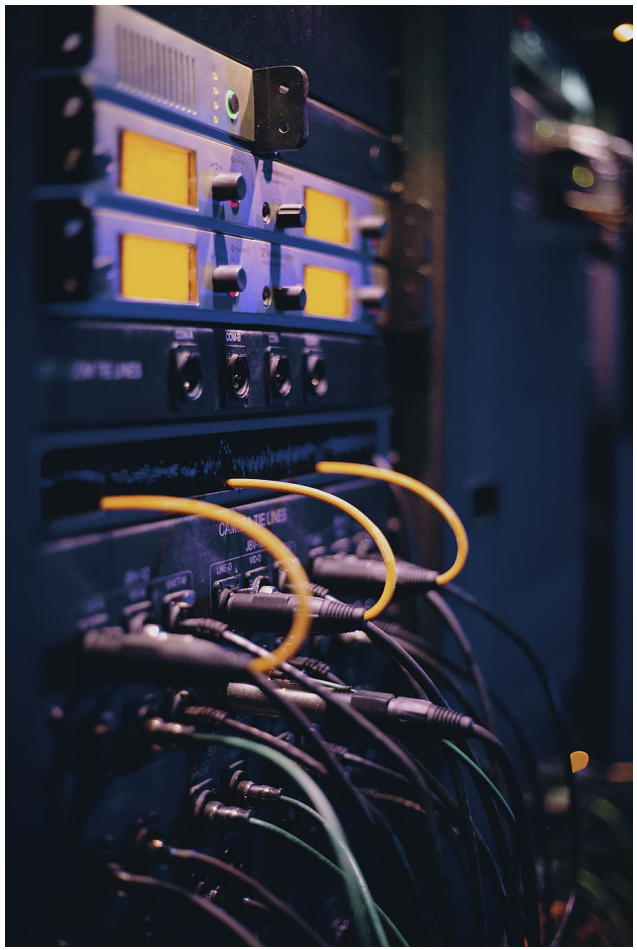
Point d'intérêt pour notre TIPE :

- Stabilité

Source: wallpaperflare

Choix du projet :

Modélisation informatique



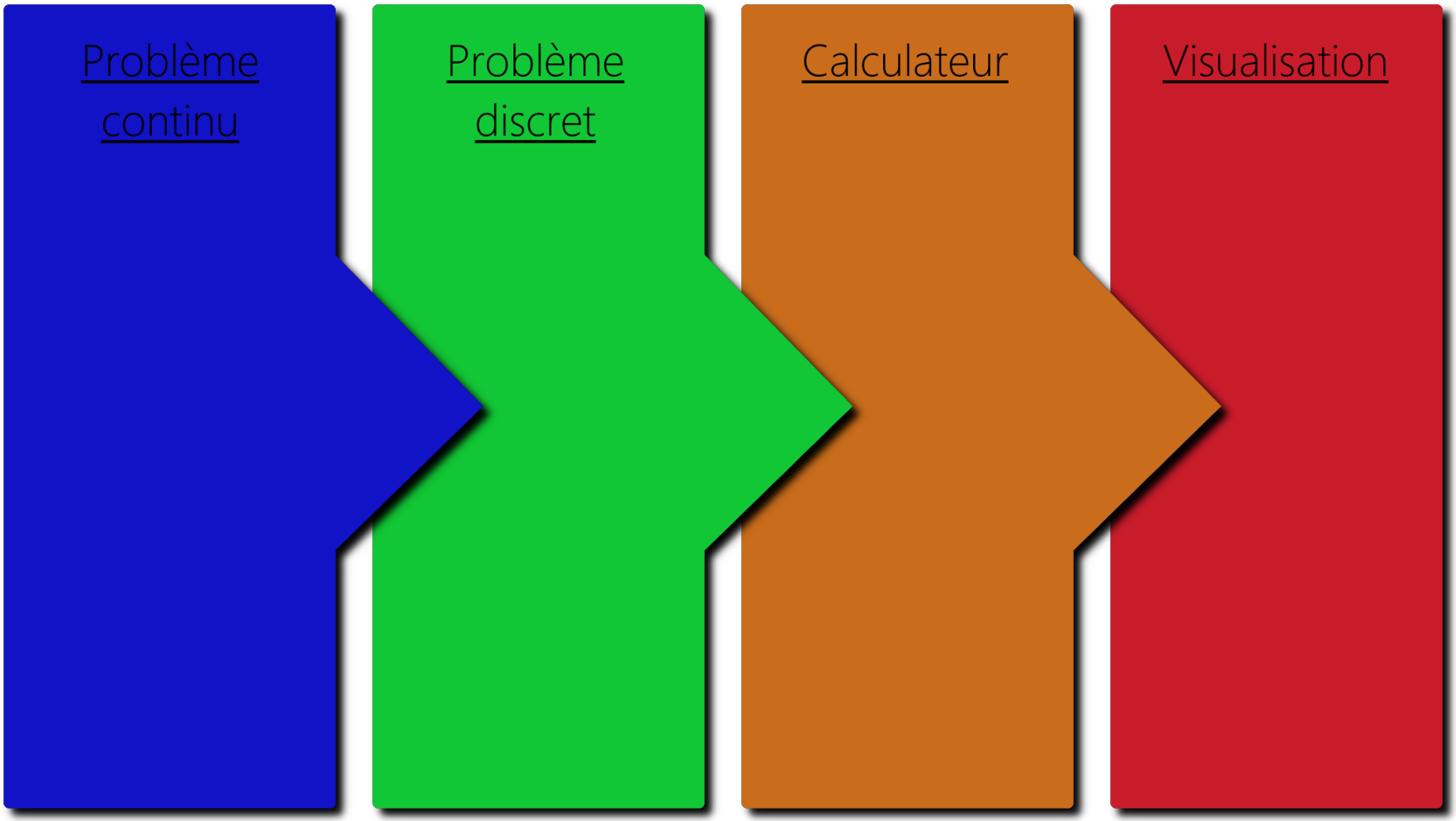
Source: wallpaperflare

Intérêts de la modélisation informatique :

- Coût
- Durée
- Aspects spécifiques
- Paramétrisation précise

Choix du projet :

Méthode des éléments finis



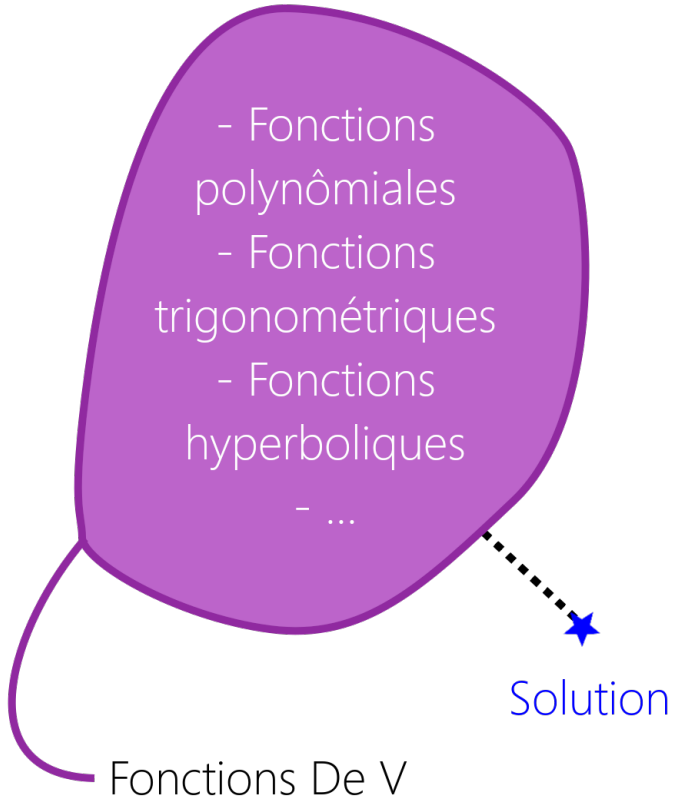
Méthode des éléments finis

Idées sous-jacentes :

- Prendre un sous-ensemble V de l'ensemble des fonctions
- Calculer le projeté de notre solution sur V à l'aide de l'équation différentielle

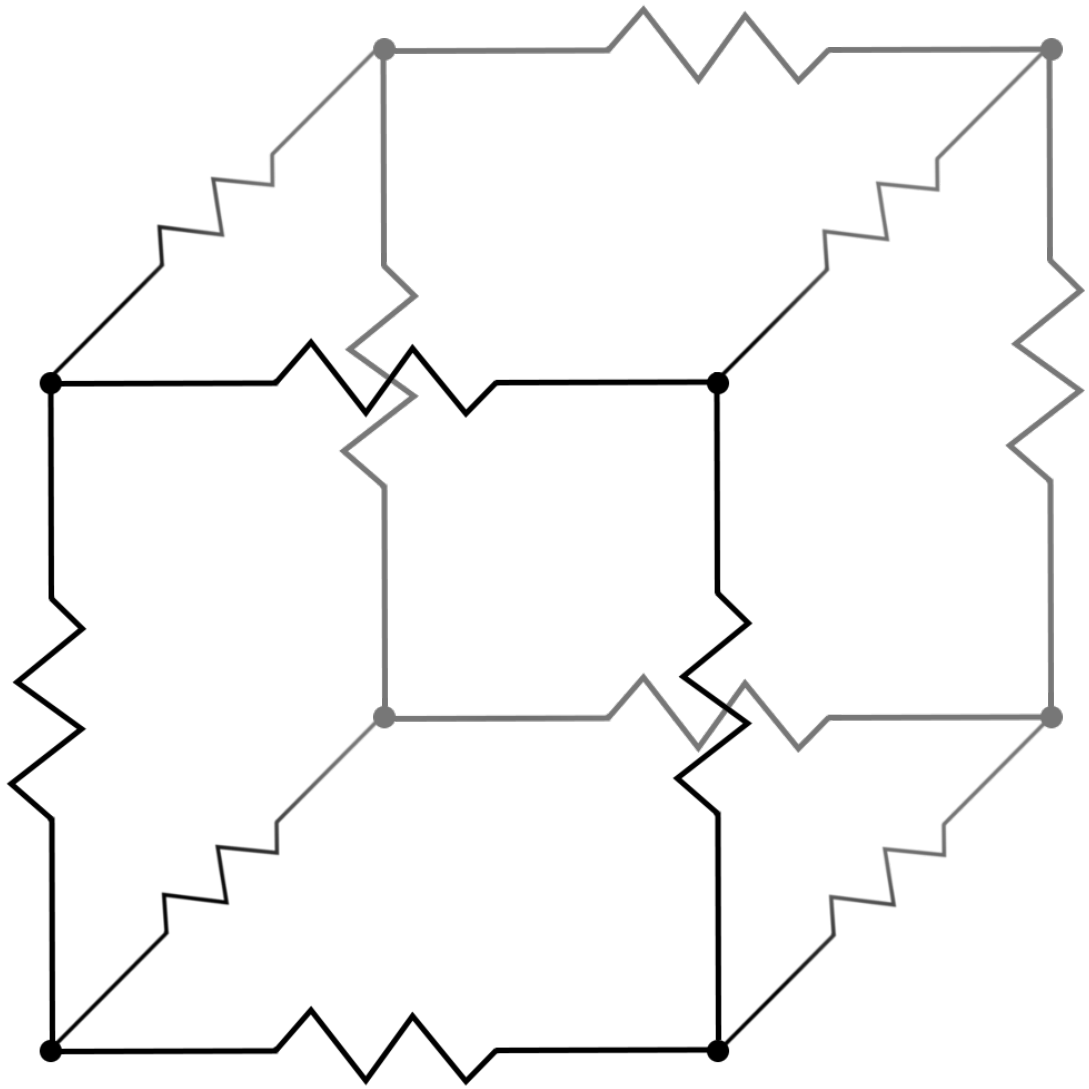
Défauts :

- Connaissances mathématiques en calcul différentiel poussé
- Difficile à généraliser



Méthode des éléments finis :

Méthode des ressorts



Loi de Hooke

$$\vec{\mathcal{F}} = -k \cdot \Delta\ell \cdot \vec{u} \tag{1}$$

Constante de raideur

$$k = \frac{\mathrm{A} \cdot \mathrm{E}}{\mathrm{L}} \tag{2}$$

avec :

- A : l'aire de la section de la poutre
- E : le module de Young du matériel
- L : la longueur de la poutre

Loi de Hooke matricielle

$$F = K \cdot U \tag{3}$$

Forme détaillée

$$\begin{pmatrix} F_c \\ F_i \end{pmatrix} = \begin{pmatrix} K_1 & K_2 \\ K_3 & K_4 \end{pmatrix} \cdot \begin{pmatrix} U_i \\ U_c \end{pmatrix} \tag{4}$$

Conditions aux limites pour la solution d'une équation différentielle :

- Neumann : information sur ses dérivées (forces connues)
- Dirichlet : information sur sa valeur (déplacments connus)

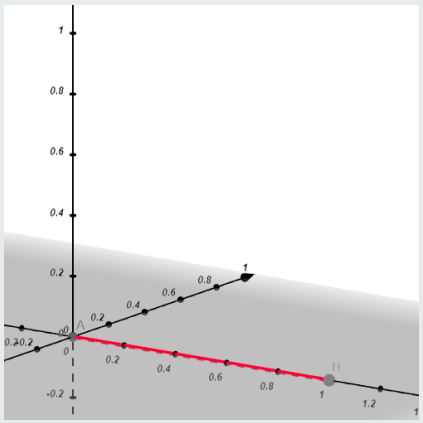
Récupération des forces et des déplacements inconnus

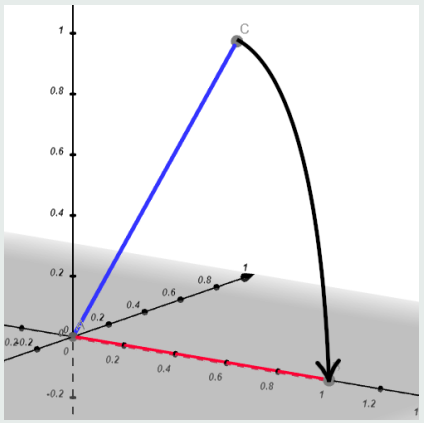
$$F_c = K_1 \cdot U_i + K_2 \cdot U_c \tag{5}$$

$$d'où : \ U_i = K_1^{-1} \cdot (F_c - K_2 \cdot U_c) \tag{6}$$

$$et : \ F_i = K_3 \cdot U_i + K_3 \cdot U_c \tag{7}$$

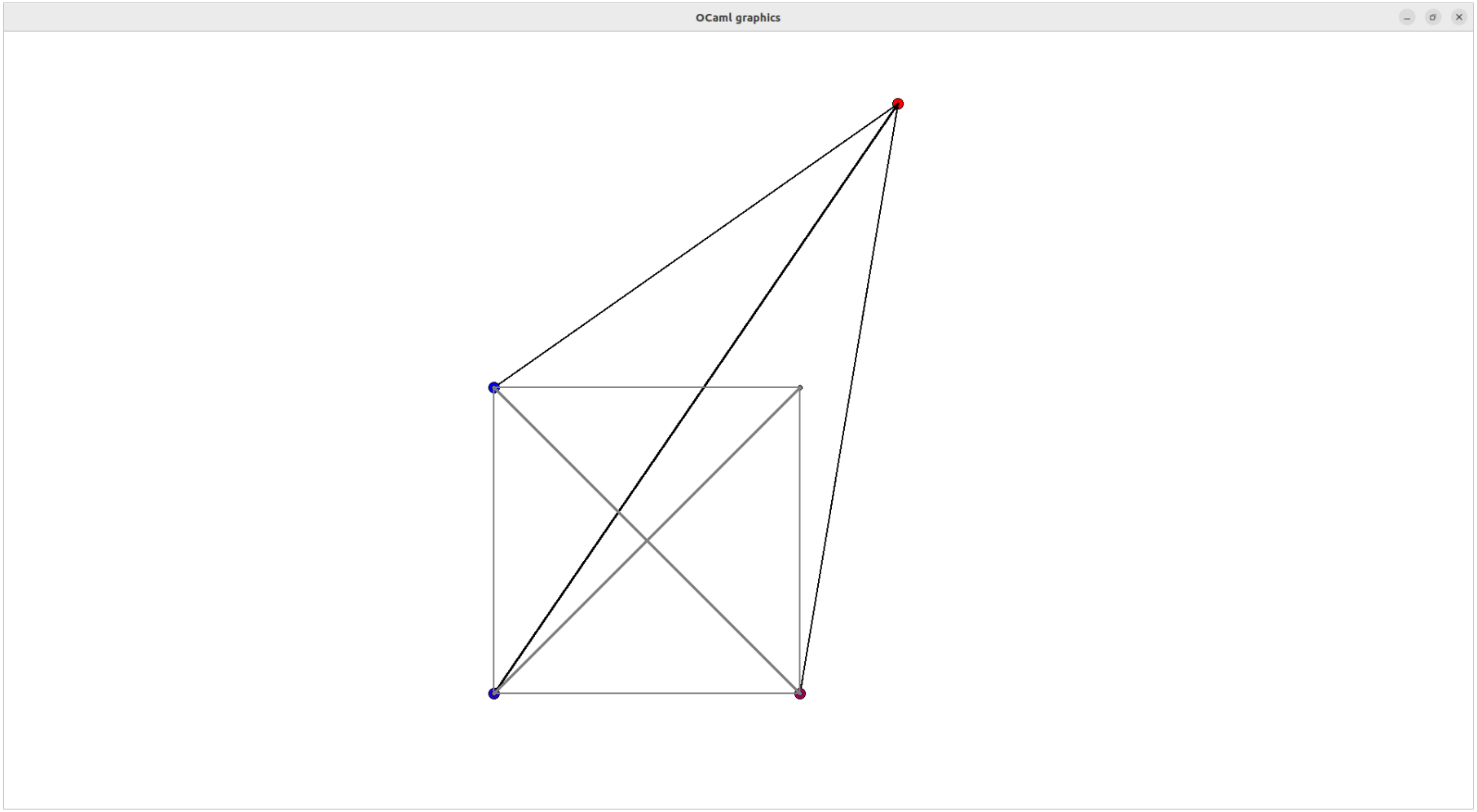
Rotation vers cas par défault





Méthode des éléments finis :

Affichage



Optimisation

Points de divergence des méthode :

- Vitesse
- Coût
- Précision des calculs
- Flexibilité du paramètrage

Optimisation :

Retours expérimentaux

Pas encore fait

La mécanique quantique une solution pour le futur

Algorithme d'inversion HHL (H.arrow H.assidim L.loyd) :

«HHL apporte une amélioration significative, de O(n) à O(log(n)).»[1]

[1] *L'apport des technologies quantiques en intelligence artificielle : vers une acculturation et une compréhension des enjeux du quantique pour l'armée de l'air et de l'espace*, Commandant Campo Marie-Élisabeth, Bureau numérique de l'armée de l'air et de l'espace

Annexes

Annexes :

Code C : standard_lib.h

```c
/*  -Importations-  */

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

Annexes :

Code C : module_ matrice.h

```c
/*  ~Matrices~  */

/*  -Importations-  */

#include "standard_lib.h"


/*  -Types et structures-  */

typedef double valeur;

struct matrice_s {
    int lignes;
    int colonnes;
    valeur** contenu;
};

typedef struct matrice_s matrice;;

/*  -Déclarations fonctions (f) et procédures (p)-  */

matrice* creer_matrice(int lignes, int colonnes);
// f - crée une matrice de dimension 'lignes'x'colonnes' initialisé à 0
    (valeur par default du type 'valeur' à changer si ce dernier change)

```

Annexes :

Code C : module_ matrice.h

```
25 void supprimer_matrice(matrice* matriceEntree);
26 // p - vide la mémoire utilisée par la matrice 'matriceEntree'
27
28 matrice* sous_matrice(matrice* matriceEntree, int ligneDepart, int
       colonneDepart, int nbreLignes, int nbreColonnes);
29 // f - créer la sous matrice comme spécifiée
30
31 matrice* add_matrice(matrice* matriceA, matrice* matriceB);
32 // f - additionne les matrices 'matriceA' et 'matriceB' de manière non
       destructive
33
34 matrice* soustract_matrice(matrice* matriceA, matrice* matriceB);
35 // f - soustrait la matrice 'matriceB' à la matrice 'matriceB' de manièr
       re non destructive
36
37 matrice* mult_matrice(matrice* matriceA, matrice* matriceB);
38 // f - multiplie les matrices 'matriceA' et 'matriceB' de manière non
       destructive
39
40 matrice* transp_matrice(matrice* matriceEntree);
41 // f - transpose la matrice 'matriceEntree' de manière non destructive
42
43 matrice* dilatation_matrice(valeur scalaire, matrice* matriceEntree);
44 // f - dilate la matrice par une scalaire (de type 'valeur') 'scalaire'
45
```

Annexes :

Code C : module_ matrice.h

```
46 valeur det_matrice(matrice* matriceEntree);
47 // f - calcule le déterminant de la matrice carrée 'matriceEntree'
48
49 void echange_ligne(matrice* matriceEntree, int ligne1, int ligne2);
50 // p - échange les lignes d'indice 'ligne1' et 'ligne2' de la matrice '
       matriceEntree' par effet de bord
51
52 void echange_colonne(matrice* matriceEntree, int colonne1, int colonne2)
       ;
53 // p - échange les colonnes d'indice 'colonne1' et 'colonne2' de la
       matrice 'matriceEntree' par effet de bord
54
55 void combinaison_lignes(matrice* matriceEntree, int ligneDest, valeur
       scalaire, int ligneAjout);
56 // p - affecte à la ligne d'indice 'ligneDest' elle-même plus la ligne d
       'indice 'ligneAjout' multipliée par un scalaire 'scalaire' par effet
        de bord
57
58 void combinaison_colonnes(matrice* matriceEntree, int colonneDest,
       valeur scalaire, int colonneAjout);
59 // p - affecte à la colonne d'indice 'colonneDest' elle-même plus la
       colonne d'indice 'colonneAjout' multipliée par un scalaire 'scalaire
       ' par effet de bord
60
```

Annexes :

Code C : module_ matrice.h

```
61 void dilatation_ligne(matrice* matriceEntree, valeur scalaire, int ligne
       );
62 // p - affecte à la ligne d'indice 'ligne' elle-même multipliée par un
       scalaire 'scalaire' non-nul par effet de bord
63
64 void dilatation_colonne(matrice* matriceEntree, valeur scalaire, int
       colonne);
65 // p - affecte à la colonne d'indice 'colonne' elle-même multipliée par
       un scalaire 'scalaire' non-nul par effet de bord
66
67 matrice* inv_matrice(matrice* matriceEntree, bool verifie);
68 // f - calcule la matrice inverse de la matrice carrée inversible '
       matriceEntree' de manière non destructive
69
70 matrice* affichage_matrice(matrice* matriceEntree);
71 // p - affiche la matrice 'matriceEntree'
```

Annexes :

Code C : module_matrice.c

```c
/*  -Fichier entête-  */

#include "module_matrice.h"


/* -Fonctions- */

matrice* creer_matrice(int lignes, int colonnes)
{
    if (lignes <= 0 || colonnes <= 0)
    {
        fprintf(stderr, "création impossible:\n");
        fprintf(stderr, "\t-> les tailles 'lignes' et 'colonnes' doivent
            -être des entiers non nuls.\n");
        fprintf(stderr, "\t\t'lignes': {%d}\n", lignes);
        fprintf(stderr, "\t\t'colonnes': {%d}\n", colonnes);
        exit(EXIT_FAILURE);
    }

    matrice* matriceSortie  = malloc(sizeof(matrice));
    matriceSortie->lignes   = lignes;
    matriceSortie->colonnes = colonnes;
    matriceSortie->contenu  = malloc(sizeof(valeur*)*lignes);

    for (int ligne = 0; ligne < lignes; ++ligne)
```

Annexes :

Code C : module_ matrice.c

```
25      {
26          matriceSortie->contenu[ligne] = malloc(sizeof(valeur)*colonnes);
27
28          for (int colonne = 0; colonne < colonnes; ++colonne)
29          {
30              matriceSortie->contenu[ligne][colonne] = 0.0;
31          }
32      }
33
34      return matriceSortie;
35  }
36
37  void supprimer_matrice(matrice* matriceEntree)
38  {
39      for (int ligne = 0; ligne < matriceEntree->lignes; ++ligne)
40      {
41          free(matriceEntree->contenu[ligne]);
42      }
43
44      free(matriceEntree->contenu);
45      free(matriceEntree);
46  }
47
48  matrice* sous_matrice(matrice* matriceEntree, int ligneDepart, int
        colonneDepart, int nbreLignes, int nbreColonnes)
```

Annexes :

Code C : module_ matrice.c

```
49 {
50     matrice* matriceSortie = creer_matrice(nbreLignes, nbreColonnes);
51
52     for (int i = 0; i < nbreLignes; ++i)
53     {
54         for (int j = 0; j < nbreColonnes; ++j)
55         {
56             matriceSortie->contenu[i][j] = matriceEntree->contenu[i+
                   ligneDepart][j+colonneDepart];
57         }
58     }
59
60     return matriceSortie;
61 }
62
63 matrice* add_matrice(matrice* matriceA, matrice* matriceB)
64 {
65     if (matriceA->lignes != matriceB->lignes || matriceA->colonnes !=
           matriceB->colonnes)
66     {
67         fprintf(stderr, "addition␣impossible:\n");
68         fprintf(stderr, "\t->␣les␣deux␣matrices␣doivent-être␣de␣taille␣
               identique.\n");
69         fprintf(stderr, "\t\t'matriceA':␣{%d}␣lignes\n", matriceA->
               lignes);
```

Annexes :

Code C : module_ matrice.c

```
70          fprintf(stderr, "\t\t'matriceA': {%d} colonnes\n", matriceA->
                colonnes);
71          fprintf(stderr, "\t\t'matriceB': {%d} lignes\n", matriceB->
                lignes);
72          fprintf(stderr, "\t\t'matriceB': {%d} colonnes\n", matriceB->
                colonnes);
73          exit(EXIT_FAILURE);
74      }
75
76      matrice* matriceSortie = creer_matrice(matriceA->lignes, matriceA->
            colonnes);
77
78      for (int ligne = 0; ligne < matriceSortie->lignes; ++ligne)
79      {
80          for (int colonne = 0; colonne < matriceSortie->colonnes; ++
                colonne)
81          {
82              matriceSortie->contenu[ligne][colonne] = matriceA->contenu[
                    ligne][colonne] + matriceB->contenu[ligne][colonne];
83          }
84      }
85
86      return matriceSortie;
87  }
88
```

Annexes :

Code C : module_ matrice.c

```
89  matrice* soustract_matrice(matrice* matriceA, matrice* matriceB)
90  {
91      if (matriceA->lignes != matriceB->lignes || matriceA->colonnes !=
            matriceB->colonnes)
92      {
93          fprintf(stderr, "soustraction impossible:\n");
94          fprintf(stderr, "\t-> les deux matrices doivent-être de taille
                identique.\n");
95          fprintf(stderr, "\t\t'matriceA': {%d} lignes\n", matriceA->
                lignes);
96          fprintf(stderr, "\t\t'matriceA': {%d} colonnes\n", matriceA->
                colonnes);
97          fprintf(stderr, "\t\t'matriceB': {%d} lignes\n", matriceB->
                lignes);
98          fprintf(stderr, "\t\t'matriceB': {%d} colonnes\n", matriceB->
                colonnes);
99          exit(EXIT_FAILURE);
100     }
101
102     matrice* matriceSortie = creer_matrice(matriceA->lignes, matriceA->
            colonnes);
103
104     for (int ligne = 0; ligne < matriceSortie->lignes; ++ligne)
105     {
```

Annexes :

Code C : module_ matrice.c

```
106         for (int colonne = 0; colonne < matriceSortie->colonnes; ++
                colonne)
107         {
108             matriceSortie->contenu[ligne][colonne] = matriceA->contenu[
                    ligne][colonne] - matriceB->contenu[ligne][colonne];
109         }
110     }
111
112     return matriceSortie;
113 }
114
115 matrice* mult_matrice(matrice* matriceA, matrice* matriceB)
116 {
117     if (matriceA->colonnes != matriceB->lignes)
118     {
119         fprintf(stderr, "multiplication␣impossible:\n");
120         fprintf(stderr, "\t->␣la␣matrice␣'matriceA'␣doit␣avoir␣autant␣de
                ␣colonnes␣que␣la␣matrice␣'matriceB'␣a␣de␣lignes.\n");
121         fprintf(stderr, "\t\t'matriceA':␣{%d}␣colonnes\n", matriceA->
                colonnes);
122         fprintf(stderr, "\t\t'matriceB':␣{%d}␣lignes\n", matriceB->
                lignes);
123         exit(EXIT_FAILURE);
124     }
125
```

Annexes :

Code C : module_ matrice.c

```
126     int tailleCommune = matriceA->colonnes;
127     matrice* matriceSortie = creer_matrice(matriceA->lignes, matriceB->
            colonnes);
128
129     for (int ligne = 0; ligne < matriceSortie->lignes; ++ligne)
130     {
131         for (int colonne = 0; colonne < matriceSortie->colonnes; ++
                colonne)
132         {
133             valeur somme = 0;
134
135             for (int k = 0; k < tailleCommune; ++k)
136             {
137                 somme += matriceA->contenu[ligne][k] * matriceB->contenu
                        [k][colonne];
138             }
139
140             matriceSortie->contenu[ligne][colonne] = somme;
141         }
142     }
143
144     return matriceSortie;
145 }
146
147 matrice* transp_matrice(matrice* matriceEntree)
```

Annexes :

Code C : module_ matrice.c

```
148 {
149     matrice* matriceSortie = creer_matrice(matriceEntree->colonnes,
          matriceEntree->lignes);
150
151     for (int ligne = 0; ligne < matriceSortie->lignes; ++ligne)
152     {
153         for (int colonne = 0; colonne < matriceSortie->colonnes; ++
              colonne)
154         {
155             matriceSortie->contenu[ligne][colonne] = matriceEntree->
                  contenu[colonne][ligne];
156         }
157     }
158
159     return matriceSortie;
160 }
161
162 matrice* dilatation_matrice(valeur scalaire, matrice* matriceEntree)
163 {
164     matrice* matriceSortie = creer_matrice(matriceEntree->lignes,
          matriceEntree->colonnes);
165
166     for (int ligne = 0; ligne < matriceSortie->lignes; ++ligne)
167     {
```

Annexes :

Code C : module_ matrice.c

```
168         for (int colonne = 0; colonne < matriceSortie->colonnes; ++
                colonne)
169         {
170             matriceSortie->contenu[ligne][colonne] = scalaire *
                    matriceEntree->contenu[ligne][colonne];
171         }
172     }
173
174     return matriceSortie;
175 }
176
177 valeur det_matrice(matrice* matriceEntree)
178 {
179     if (matriceEntree->colonnes != matriceEntree->lignes)
180     {
181         fprintf(stderr, "calcule du déterminant impossible:\n");
182         fprintf(stderr, "\t-> la matrice 'matriceEntree' doit avoir
                autant de colonnes que de lignes.\n");
183         fprintf(stderr, "\t\t'matriceEntree': {%d} lignes\n",
                matriceEntree->lignes);
184         fprintf(stderr, "\t\t'matriceEntree': {%d} colonnes\n",
                matriceEntree->colonnes);
185         exit(EXIT_FAILURE);
186     }
187
```

Annexes :

Code C : module_matrice.c

```
188     int tailleCommune = matriceEntree->colonnes;
189
190     // cas d'arrêt
191
192     if (tailleCommune == 1)
193     {
194         return matriceEntree->contenu[0][0];
195     }
196
197     // recherche meilleur ligne/colonne (celle possèdant le moins de 0)
198
199     int idMeilleur   = 0;
200     int nbreZeroMax  = 0;
201     int nbreZero     = 0;
202     bool estVertical = false;
203
204     for (int ligne  = 0; ligne < tailleCommune; ++ligne)
205     {
206         nbreZero = 0;
207
208         for (int colonne = 0; colonne < tailleCommune; ++colonne)
209         {
210             if (matriceEntree->contenu[ligne][colonne] == 0.0)
211             {
212                 nbreZero += 1;
```

Annexes :

Code C : module_ matrice.c

```
213             }
214         }
215
216         if (nbreZero > nbreZeroMax)
217         {
218             nbreZeroMax = nbreZero;
219             idMeilleur  = ligne;
220         }
221     }
222
223     for (int colonne  = 0; colonne < tailleCommune; ++colonne)
224     {
225         nbreZero = 0;
226
227         for (int ligne = 0; ligne < tailleCommune; ++ligne)
228         {
229             if (matriceEntree->contenu[ligne][colonne] == 0.0)
230             {
231                 nbreZero += 1;
232             }
233         }
234
235         if (nbreZero > nbreZeroMax)
236         {
237             nbreZeroMax = nbreZero;
```

Annexes :

Code C : module_ matrice.c

```
238             idMeilleur   = colonne;
239             estVertical = true;
240         }
241     }
242
243     //  calcule du déterminant (on se ramène à la transoposé si le
            calcule le plus intéressant est sur une colonne)
244
245     // cas simple
246
247     if (nbreZeroMax == tailleCommune)
248     {
249         return 0.0;
250     }
251
252     // cas général
253
254     valeur det    = 0.0;
255     valeur signe = (idMeilleur % 2 == 0) ? 1.0 : -1.0;
256     int ligneTemp;
257     int colonneTemp;
258
259     if (estVertical)
260     {
261         matriceEntree = transp_matrice(matriceEntree);
```

Annexes :

Code C : module_ matrice.c

```
262     }
263
264     matrice* matriceTemp = creer_matrice(tailleCommune-1, tailleCommune
            -1);
265
266     for (int colonneEnCours = 0; colonneEnCours < tailleCommune; ++
            colonneEnCours)
267     {
268         if (matriceEntree->contenu[idMeilleur][colonneEnCours] != 0.0)
269         {
270             ligneTemp = 0;
271
272             for (int ligne = 0; ligne < tailleCommune; ++ligne)
273             {
274                 colonneTemp = 0;
275
276                 if (ligne != idMeilleur)
277                 {
278                     for(int colonne = 0; colonne < tailleCommune; ++
                            colonne)
279                     {
280                         if (colonne != colonneEnCours)
281                         {
```

Annexes :

Code C : module_ matrice.c

```
282                                 matriceTemp->contenu[ligneTemp][colonneTemp]
                                          = matriceEntree->contenu[ligne][colonne
                                        ];
283                                 colonneTemp++;
284                             }
285                         }
286
287                         ligneTemp++;
288                     }
289                 }
290
291                 det += signe * matriceEntree->contenu[idMeilleur][
                         colonneEnCours] * det_matrice(matriceTemp);
292             }
293
294             signe *= -1.0;
295         }
296
297         if (estVertical)
298         {
299             supprimer_matrice(matriceEntree);
300         }
301         supprimer_matrice(matriceTemp);
302
303         return det;
```

Annexes :

Code C : module_matrice.c

```
304 }
305
306 void echange_ligne(matrice* matriceEntree, int ligne1, int ligne2)
307 {
308     if (ligne1 >= matriceEntree->lignes || ligne1 < 0 || ligne2 >=
          matriceEntree->lignes || ligne2 < 0)
309     {
310         fprintf(stderr, "échange des lignes impossible:\n");
311         fprintf(stderr, "\t-> les lignes 'ligne1' et 'ligne2' doivent
              exister.\n");
312         fprintf(stderr, "\t\t'ligne1': {%d}\n", ligne1);
313         fprintf(stderr, "\t\t'ligne2': {%d}\n", ligne2);
314         exit(EXIT_FAILURE);
315     }
316
317     for (int colonne = 0; colonne < matriceEntree->colonnes; ++colonne)
318     {
319         valeur stockageTemp                          = matriceEntree->contenu
              [ligne1][colonne];
320         matriceEntree->contenu[ligne1][colonne] = matriceEntree->contenu
              [ligne2][colonne];
321         matriceEntree->contenu[ligne2][colonne] = stockageTemp;
322     }
323 }
324
```

Annexes :

Code C : module_matrice.c

```
325 void echange_colonne(matrice* matriceEntree, int colonne1, int colonne2)
326 {
327     if (colonne1 >= matriceEntree->colonnes || colonne1 < 0 || colonne2
          >= matriceEntree->colonnes || colonne2 < 0)
328     {
329         fprintf(stderr, "échange␣des␣colonnes␣impossible:\n");
330         fprintf(stderr, "\t->␣les␣colonnes␣'colonne1'␣et␣'colonne2'␣
              doivent␣exister.\n");
331         fprintf(stderr, "\t\t'colonne1':␣{%d}\n", colonne1);
332         fprintf(stderr, "\t\t'colonne2':␣{%d}\n", colonne2);
333         exit(EXIT_FAILURE);
334     }
335
336     for (int ligne = 0; ligne < matriceEntree->lignes; ++ligne)
337     {
338         valeur stockageTemp                          = matriceEntree->contenu
              [ligne][colonne1];
339         matriceEntree->contenu[ligne][colonne1] = matriceEntree->contenu
              [ligne][colonne2];
340         matriceEntree->contenu[ligne][colonne2] = stockageTemp;
341     }
342 }
343
344 void combinaison_lignes(matrice* matriceEntree, int ligneDest, valeur
      scalaire, int ligneAjout)
```

Annexes :

Code C : module_matrice.c

```
345 {
346     if (ligneDest >= matriceEntree->lignes || ligneDest < 0 ||
            ligneAjout >= matriceEntree->lignes || ligneAjout < 0)
347     {
348         fprintf(stderr, "combinaison␣des␣lignes␣impossible:\n");
349         fprintf(stderr, "\t->␣les␣lignes␣'ligneDest'␣et␣'ligneAjout'␣
                doivent␣exister.\n");
350         fprintf(stderr, "\t\t'ligneDest':␣{%d}\n", ligneDest);
351         fprintf(stderr, "\t\t'ligneAjout':␣{%d}\n", ligneAjout);
352         exit(EXIT_FAILURE);
353     }
354
355     for (int colonne = 0; colonne < matriceEntree->colonnes; ++colonne)
356     {
357         matriceEntree->contenu[ligneDest][colonne] = matriceEntree->
                contenu[ligneDest][colonne] + scalaire * matriceEntree->
                contenu[ligneAjout][colonne];
358     }
359 }
360
361 void combinaison_colonne(matrice* matriceEntree, int colonneDest, valeur
        scalaire, int colonneAjout)
362 {
363     if (colonneDest >= matriceEntree->colonnes || colonneDest < 0 ||
            colonneAjout >= matriceEntree->colonnes || colonneAjout < 0)
```

Annexes :

Code C : module_matrice.c

```
364     {
365         fprintf(stderr, "combinaison des colonnes impossible:\n");
366         fprintf(stderr, "\t-> les colonnes 'colonneDest' et '
                colonneAjout' doivent exister.\n");
367         fprintf(stderr, "\t\t'colonneDest': {%d}\n", colonneDest);
368         fprintf(stderr, "\t\t'colonneAjout': {%d}\n", colonneAjout);
369         exit(EXIT_FAILURE);
370     }
371
372     for (int ligne = 0; ligne < matriceEntree->lignes; ++ligne)
373     {
374         matriceEntree->contenu[ligne][colonneDest] = matriceEntree->
                contenu[ligne][colonneDest] + scalaire * matriceEntree->
                contenu[ligne][colonneAjout];
375     }
376 }
377
378 void dilatation_ligne(matrice* matriceEntree, valeur scalaire, int ligne
        )
379 {
380     if (ligne >= matriceEntree->lignes || ligne < 0)
381     {
382         fprintf(stderr, "dilatation de la ligne impossible:\n");
383         fprintf(stderr, "\t-> la ligne 'ligne' doit exister.\n");
384         fprintf(stderr, "\t\t'ligne': {%d}\n", ligne);
```

Annexes :

Code C : module_ matrice.c

```
385             exit(EXIT_FAILURE);
386         }
387
388         if (scalaire == 0)
389         {
390             fprintf(stderr, "dilatation de la ligne impossible:\n");
391             fprintf(stderr, "\t-> le scalaire 'scalaire' doit être non nul.\
                    n");
392             fprintf(stderr, "\t\t'scalaire': {%f}\n", scalaire); // à
                    modifier si valeur change de type
393             exit(EXIT_FAILURE);
394         }
395
396         for (int colonne = 0; colonne < matriceEntree->colonnes; ++colonne)
397         {
398             matriceEntree->contenu[ligne][colonne] = scalaire *
                    matriceEntree->contenu[ligne][colonne];
399         }
400 }
401
402 void dilatation_colonne(matrice* matriceEntree, valeur scalaire, int
        colonne)
403 {
404         if (colonne >= matriceEntree->colonnes || colonne < 0)
405         {
```

Annexes :

Code C : module_ matrice.c

```
406             fprintf(stderr, "dilatation de la colonne impossible:\n");
407             fprintf(stderr, "\t-> la colonne 'colonne' doit exister.\n");
408             fprintf(stderr, "\t\t'colonne': {%d}\n", colonne);
409             exit(EXIT_FAILURE);
410         }
411
412         if (scalaire == 0)
413         {
414             fprintf(stderr, "dilatation de la colonne impossible:\n");
415             fprintf(stderr, "\t-> le scalaire 'scalaire' doit être non nul.\
                    n");
416             fprintf(stderr, "\t\t'scalaire': {%f}\n", scalaire); // à
                    modifier si valeur change de type
417             exit(EXIT_FAILURE);
418         }
419
420         for (int ligne = 0; ligne < matriceEntree->lignes; ++ligne)
421         {
422             matriceEntree->contenu[ligne][colonne] = scalaire *
                    matriceEntree->contenu[ligne][colonne];
423         }
424     }
425
426     matrice* inv_matrice(matrice* matriceEntree, bool verifie)
427     {
```

Annexes :

Code C : module_ matrice.c

```
428     if (matriceEntree->colonnes != matriceEntree->lignes)
429     {
430         fprintf(stderr, "calcule de l'inverse impossible:\n");
431         fprintf(stderr, "\t-> la matrice 'matriceEntree' doit avoir
                autant de colonnes que de lignes.\n");
432         fprintf(stderr, "\t\t'matriceEntree': {%d} lignes\n",
                matriceEntree->lignes);
433         fprintf(stderr, "\t\t'matriceEntree': {%d} colonnes\n",
                matriceEntree->colonnes);
434         exit(EXIT_FAILURE);
435     }
436
437     if (verifie)
438     {
439         valeur det = det_matrice(matriceEntree);
440         printf("%f\n", det);
441
442         if (det == 0)
443         {
444             fprintf(stderr, "calcule de l'inverse impossible:\n");
445             fprintf(stderr, "\t-> la matrice 'matriceEntree' est de dé
                    terminant nul.\n");
446             fprintf(stderr, "\t\tdéterminant: {%f}\n", det); // à
                    modifier si valeur change de type
447             exit(EXIT_FAILURE);
```

Annexes :

Code C : module_ matrice.c

```
448
449         }
450     }
451
452     // mise en place
453
454     int tailleCommune = matriceEntree->colonnes;
455     matrice* matriceTemp = creer_matrice(tailleCommune, 2*tailleCommune)
            ;
456
457     for (int ligne = 0; ligne < tailleCommune; ++ligne)
458     {
459         for (int colonne = 0; colonne < tailleCommune; ++ colonne)
460         {
461             matriceTemp->contenu[ligne][colonne] = matriceEntree->
                    contenu[ligne][colonne];
462
463             if (ligne == colonne)
464             {
465                 matriceTemp->contenu[ligne][colonne+tailleCommune] = 1;
466             }
467             else
468             {
469                 matriceTemp->contenu[ligne][colonne+tailleCommune] = 0;
470             }
```

Annexes :

Code C : module_ matrice.c

```
471         }
472     }
473
474     // algorithme de Gauss-Jordan
475
476     int lignePivot   = -1;
477
478     for (int colonne = 0; colonne < tailleCommune; ++colonne)
479     {
480         int ligneMax = lignePivot+1;
481         int maximum  = matriceTemp->contenu[lignePivot+1][colonne];
482
483         for (int ligne = lignePivot+2; ligne < tailleCommune; ++ligne)
484         {
485             if (matriceTemp->contenu[ligne][colonne] > maximum)
486             {
487                 maximum = matriceTemp->contenu[ligne][colonne];
488                 ligneMax = ligne;
489             }
490         }
491
492         if (matriceTemp->contenu[ligneMax][colonne] != 0)
493         {
494             lignePivot += 1;
```

Annexes :

Code C : module_ matrice.c

```
495             dilatation_ligne(matriceTemp, 1/matriceTemp->contenu[
                    ligneMax][colonne], ligneMax);
496
497             if (ligneMax != lignePivot)
498             {
499                 echange_ligne(matriceTemp, lignePivot, ligneMax);
500             }
501
502             for (int ligne = 0; ligne < tailleCommune; ++ligne)
503             {
504                 if (ligne != lignePivot)
505                 {
506                     combinaison_lignes(matriceTemp, ligne, (-1)*
                            matriceTemp->contenu[ligne][colonne], lignePivot
                            );
507                 }
508             }
509         }
510     }
511
512     // recopie de la matrice inverse
513
514     matrice* matriceSortie = creer_matrice(tailleCommune, tailleCommune)
            ;
515     for (int ligne = 0; ligne < tailleCommune; ++ligne)
```

Annexes :

Code C : module_ matrice.c

```
516     {
517         for (int colonne = 0; colonne < tailleCommune; ++colonne)
518         {
519             matriceSortie->contenu[ligne][colonne] = matriceTemp->
                    contenu[ligne][colonne+tailleCommune];
520         }
521     }
522
523     supprimer_matrice(matriceTemp);
524     return matriceSortie;
525 }
526
527 matrice* affichage_matrice(matrice* matriceEntree)
528 {
529     printf("Affichage:\n");
530
531     for (int ligne = 0; ligne < matriceEntree->lignes; ++ligne)
532     {
533         printf("|");
534         for (int colonne = 0; colonne < matriceEntree->colonnes; ++
                colonne)
535         {
536             printf("␣{%f}␣", matriceEntree->contenu[ligne][colonne]);
537         }
538         printf("|\n");
```

Annexes :

Code C : module_ matrice.c

```
539         }
540 }
```

Annexes :

Code C : main.c

```
1  /*
2  Documentation:
3
4      - Types de conditions limites (toujours l'une des deux)
5
6      {-1} Dirichlet : contrainte de position
7      {1}  Neumann : contrainte de force
8
9      - Forme final du problème initiale
10
11     [Fc]    [K1   K2]    [Ui]
12     [  ] = [       ] x [  ]
13     [Fi]    [K3   K4]    [Uc]
14 , où les indices i correspondent aux inconnus et c au connus pour U (dé
       placements) et F (forces)
15
16     K1 : degree_de_liberte x degree_de_liberte
17     K4 : degree_de_contrainte x degree_de_contrainte
18
19 */
20
21
22 /* - Imports - */
23
24 #include <stdio.h>
```

Annexes :

Code C : main.c

```c
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include "module_matrice.h"


/* - Constantes - */

#define Dimension 3 // s.u.
#define NoeudsParElement 2 // s.u.


/* - Structures et types - */

typedef struct
{
    matrice* position;
    matrice* deplacement;
    matrice* force;
    int* typeConstraite;
    int* indicesK;
} noeud_t;

```

Annexes :

Code C : main.c

```c
typedef struct
{
    int* indices;
    double e; // Pa module de Young
    double a; // m^2 section
} element_t;

typedef struct
{
    noeud_t* noeuds;
    element_t* elements;
    int nbreNoeuds;
    int nbreElements;
    int degreeDeLiberte;
    int degreeDeContrainte;
} probleme_t;


/* - Décalrations - */

probleme_t* lecture_donnees(char* lien);
/* f - récupère les données à l'adresse fournie */

void ecrit_resultat(char* lien, probleme_t* probleme);
/* f - écrit les données traités à l'adresse fournie */
```

Annexes :

Code C : main.c

```
75
76 void supprime_probleme(probleme_t* probleme);
77 /* f - delete la structure du problème */
78
79 matrice* raideur_element(probleme_t* probleme, int i);
80 /* f - créer la matrice de raideur associée à l'élément d'indice i élé
       ment */
81
82 matrice* creation_matrice_raideur(probleme_t* probleme);
83 /* f - créer la matrice de raideur pour le problème */
84
85 matrice* recuperation_forces_connues(probleme_t* probleme);
86 /* f - récupération du vecteur colonne des forces connues */
87
88 matrice* recuperation_deplacements_connus(probleme_t* probleme);
89 /* f - récupération du vecteur colonne des déplacements connus */
90
91 void applique_elements_finis(char* lienDonnees, char* lienSortie);
92 /* f - effectue la méthode des éléments finis */
93
94
95 /* - Fonctions - */
96
97 probleme_t* lecture_donnees(char* lien)
98 {
```

Annexes :

Code C : main.c

```
 99      // Initialisation du problème
100
101      probleme_t* probleme = malloc(sizeof(probleme_t));
102
103      probleme->degreeDeLiberte    = 0;
104      probleme->degreeDeContrainte = 0;
105
106      // Ouverture du fichier
107
108      FILE* fichier = NULL;
109
110      fichier = fopen(lien, "r");
111
112      // Lecture des informations primaires
113
114      fscanf(fichier, "%d;%d\n", &(probleme->nbreNoeuds), &(probleme->
             nbreElements));
115
116      // Lecture des noeuds
117
118      probleme->noeuds = malloc(sizeof(noeud_t) * probleme->nbreNoeuds);
119
120      double valeur;
121
122      for (int i = 0; i < probleme->nbreNoeuds; ++i)
```

Annexes :

Code C : main.c

```
123     {
124         // Initialisation du noeud
125 
126         probleme->noeuds[i].position        = creer_matrice(Dimension, 1)
                ;
127         probleme->noeuds[i].deplacement     = creer_matrice(Dimension, 1)
                ;
128         probleme->noeuds[i].force           = creer_matrice(Dimension, 1)
                ;
129         probleme->noeuds[i].typeConstraite = malloc(sizeof(int) *
                Dimension);
130         probleme->noeuds[i].indicesK        = malloc(sizeof(int) *
                Dimension);
131 
132         // Lecture du noeud
133 
134         for (int d = 0; d < Dimension; ++d)
135         {
136             fscanf(fichier, "%lf;%d;%lf\n", &(probleme->noeuds[i].
                    position->contenu[d][0]), &(probleme->noeuds[i].
                    typeConstraite[d]), &valeur);
137 
138             if (probleme->noeuds[i].typeConstraite[d] == 1)
139             {
140                 ++(probleme->degreeDeLiberte);
```

Annexes :

Code C : main.c

```
141                 probleme->noeuds[i].force->contenu[d][0]        = valeur;
142                 probleme->noeuds[i].deplacement->contenu[d][0] = 0.0;
143                 probleme->noeuds[i].indicesK[d]                 =
                        probleme->degreeDeLiberte;
144             }
145             else
146             {
147                 --(probleme->degreeDeContrainte);
148                 probleme->noeuds[i].deplacement->contenu[d][0] = valeur;
149                 probleme->noeuds[i].force->contenu[d][0]        = 0.0;
150                 probleme->noeuds[i].indicesK[d]                 =
                        probleme->degreeDeContrainte;
151             }
152         }
153     };
154
155     // Formatage des indices dans K
156
157     for (int i = 0; i < probleme->nbreNoeuds; ++i)
158     {
159         for (int d = 0; d < Dimension; ++d)
160         {
161             if (probleme->noeuds[i].indicesK[d] < 0)
162             {
```

Annexes :

Code C : main.c

```
163                 probleme->noeuds[i].indicesK[d] = abs(probleme->noeuds[i
                        ].indicesK[d]) + (probleme->degreeDeLiberte) - 1;
164             }
165             else
166             {
167                 probleme->noeuds[i].indicesK[d] = probleme->noeuds[i].
                        indicesK[d] - 1;
168             }
169         }
170     }
171
172     probleme->degreeDeContrainte = abs(probleme->degreeDeContrainte);
173
174     // Lecture des elements
175
176     probleme->elements = malloc(sizeof(element_t) * probleme->
            nbreElements);
177
178     for (int i = 0; i < probleme->nbreElements; ++i)
179     {
180         // Initialisation de l'élément
181
182         probleme->elements[i].indices = malloc(sizeof(int) *
                NoeudsParElement);
183
```

Annexes :

Code C : main.c

```
184         // Lecture de l'élément
185
186         fscanf(fichier, "%d;%d;%lf;%lf\n", &(probleme->elements[i].
                indices[0]), &(probleme->elements[i].indices[1]), &(probleme
                ->elements[i].e), &(probleme->elements[i].a));
187     }
188
189     fclose(fichier);
190
191     return probleme;
192 }
193
194 void ecrit_resultat(char* lien, probleme_t* probleme)
195 {
196     // Ouverture du fichier
197
198     FILE* fichier = NULL;
199
200     fichier = fopen(lien, "w");
201
202     // Ecriture des informations primaires
203
204     fprintf(fichier, "%d;%d\n", probleme->nbreNoeuds, probleme->
            nbreElements);
205
```

Annexes :

Code C : main.c

```
206     // Ecriture des noeuds
207
208     for (int i = 0; i < probleme->nbreNoeuds; ++i)
209     {
210         for (int d = 0; d < Dimension; ++d)
211         {
212             fprintf(fichier, "%lf;%lf;%lf\n", probleme->noeuds[i].
                    position->contenu[d][0], probleme->noeuds[i].deplacement
                    ->contenu[d][0], probleme->noeuds[i].force->contenu[d
                    ][0]);
213         }
214     }
215
216     // Ecriture des elements
217
218     for (int i = 0; i < probleme->nbreElements; ++i)
219     {
220         fprintf(fichier, "%d;%d;%lf;%lf\n", probleme->elements[i].
                indices[0], probleme->elements[i].indices[1], probleme->
                elements[i].e, probleme->elements[i].a);
221     }
222
223     fclose(fichier);
224 }
225
```

Annexes :

Code C : main.c

```
226 void supprime_probleme(probleme_t* probleme)
227 {
228     // free des noeuds
229 
230     for (int i = 0; i < probleme->nbreNoeuds; ++i)
231     {
232         supprimer_matrice(probleme->noeuds[i].position);
233         supprimer_matrice(probleme->noeuds[i].deplacement);
234         supprimer_matrice(probleme->noeuds[i].force);
235         free(probleme->noeuds[i].typeConstraite);
236         free(probleme->noeuds[i].indicesK);
237     }
238 
239     free(probleme->noeuds);
240 
241     // free des elements
242 
243     for (int i = 0; i < probleme->nbreElements; ++i)
244     {
245         free(probleme->elements[i].indices);
246     }
247 
248     free(probleme->elements);
249 
250     // free du probleme
```

Annexes :

Code C : main.c

```
251
252     free(probleme);
253
254     return;
255 }
256
257 matrice* raideur_element(probleme_t* probleme, int i)
258 {
259     // Calcul de la rotation selon Oz
260
261     int indice1 = probleme->elements[i].indices[0];
262     int indice2 = probleme->elements[i].indices[1];
263
264     double x1 = probleme->noeuds[indice1].position->contenu[0][0];
265     double y1 = probleme->noeuds[indice1].position->contenu[1][0];
266     double x2 = probleme->noeuds[indice2].position->contenu[0][0];
267     double y2 = probleme->noeuds[indice2].position->contenu[1][0];
268
269     double deltax = x2 - x1;
270     double deltay = y2 - y1;
271
272     double longeurProj = sqrt(deltax * deltax + deltay * deltay);
273
274     double cosinus;
275     double sinus;
```

Annexes :

Code C : main.c

```
276
277     if (sqrt(longeurProj * longeurProj) <= 0.001) // on évite la
            division par 0
278     {
279         cosinus = 1.0;
280         sinus   = 0.0;
281     }
282     else
283     {
284         cosinus = deltax / longeurProj;
285         sinus   = deltay / longeurProj;
286     }
287
288     // - création de la matrice de rotation R(-angle) (A)
289
290     matrice* mat_A = creer_matrice(Dimension * NoeudsParElement,
            Dimension * NoeudsParElement);
291
292     for (int n = 0; n < NoeudsParElement; ++n)
293     {
294         mat_A->contenu[0 + Dimension * n][0 + Dimension * n] = cosinus;
295         mat_A->contenu[0 + Dimension * n][1 + Dimension * n] = sinus;
296         mat_A->contenu[1 + Dimension * n][0 + Dimension * n] = -sinus;
297         mat_A->contenu[1 + Dimension * n][1 + Dimension * n] = cosinus;
298         mat_A->contenu[2 + Dimension * n][2 + Dimension * n] = 1.0;
```

Annexes :

Code C : main.c

```
299     }
300
301     // - calcule des positions après la première rotation
302
303     matrice* positions = creer_matrice(Dimension * 2, 1);
304     for (int d = 0; d < Dimension; ++d)
305     {
306         positions->contenu[d][0]           = probleme->noeuds[indice1].
                position->contenu[d][0];
307         positions->contenu[Dimension+d][0] = probleme->noeuds[indice2].
                position->contenu[d][0];
308     }
309     matrice* nouvellePositions = mult_matrice(mat_A, positions);
310
311     // Calcul de la rotation selon Oy
312
313     x1         = nouvellePositions->contenu[0][0];
314     double z1 = nouvellePositions->contenu[2][0];
315     x2         = nouvellePositions->contenu[3][0];
316     double z2 = nouvellePositions->contenu[5][0];
317
318     deltax         = x2 - x1;
319     double deltaz = z2 - z1;
320
```

Annexes :

Code C : main.c

```
321     double longeur = sqrt(deltax * deltax + deltaz * deltaz); // conserv
            ée par rotation (et plus de composante selon y)
322
323     if (sqrt(longeur * longeur) <= 0.001) // on évite la division par 0
324     {
325         cosinus = 1.0;
326         sinus   = 0.0;
327     }
328     else
329     {
330         cosinus = deltax / longeur;
331         sinus   = deltaz / longeur;
332     }
333
334     // - création de la matrice de rotation R(-angle) (B)
335
336     matrice* mat_B = creer_matrice(Dimension * NoeudsParElement,
            Dimension * NoeudsParElement);
337
338     for (int n = 0; n < NoeudsParElement; ++n)
339     {
340         mat_B->contenu[0 + Dimension * n][0 + Dimension * n] = cosinus;
341         mat_B->contenu[0 + Dimension * n][2 + Dimension * n] = -sinus;
342         mat_B->contenu[1 + Dimension * n][1 + Dimension * n] = 1.0;
343         mat_B->contenu[2 + Dimension * n][0 + Dimension * n] = sinus;
```

Annexes :

Code C : main.c

```
344         mat_B->contenu[2 + Dimension * n][2 + Dimension * n] = cosinus;
345     }
346
347     // Calcul de la matrice de raideur de l'élément
348
349     // - création de la matrice dans la base canonique (K)
350
351     matrice* mat_K = creer_matrice(Dimension * NoeudsParElement,
            Dimension * NoeudsParElement);
352
353     double constante = (probleme->elements[i].e) * (probleme->elements[i
            ].a) / longeur;
354
355     mat_K->contenu[0][0]                     = constante;
356     mat_K->contenu[0][Dimension]             = -constante;
357     mat_K->contenu[Dimension][0]             = -constante;
358     mat_K->contenu[Dimension][Dimension] = constante;
359
360     // - calcule de la matrice dans la base tournée A x B x K x tB x tA
            = AB x K x t(AB)
361
362     matrice* mat_AB  = mult_matrice(mat_A, mat_B);
363     matrice* mat_tAB = transp_matrice(mat_AB);
364
365     matrice* mat_ABK       = mult_matrice(mat_AB, mat_K);
```

Annexes :

Code C : main.c

```
366     matrice* mat_K_finale = mult_matrice(mat_ABK, mat_tAB);
367
368     // free des matrices intermédiaires et retour
369
370     supprimer_matrice(mat_A);
371     supprimer_matrice(positions);
372     supprimer_matrice(nouvellePositions);
373     supprimer_matrice(mat_K);
374     supprimer_matrice(mat_AB);
375     supprimer_matrice(mat_tAB);
376     supprimer_matrice(mat_ABK);
377
378     return mat_K_finale;
379 }
380
381 matrice* creation_matrice_raideur(probleme_t* probleme)
382 {
383     // Initialisation de la matrice
384
385     matrice* matriceRaideur = creer_matrice((probleme->nbreNoeuds) *
            Dimension, (probleme->nbreNoeuds) * Dimension);
386
387     // Assemblage de la matrice
388
389     for (int i = 0; i < probleme->nbreElements; ++i)
```

Annexes :

Code C : main.c

```
390     {
391         matrice* matriceElement = raideur_element(probleme, i);
392 
393         // Intération sur toutes les conbinaisons des noeuds
394 
395         for (int noeud1 = 0; noeud1 < NoeudsParElement; ++noeud1)
396         {
397             for (int direction1 = 0; direction1 < Dimension; ++
                    direction1)
398             {
399                 for (int noeud2 = 0; noeud2 < NoeudsParElement; ++noeud2
                        )
400                 {
401                     for (int direction2 = 0; direction2 < Dimension; ++
                            direction2)
402                     {
403                         int ligne      = probleme->noeuds[probleme->
                                elements[i].indices[noeud1]].indicesK[
                                direction1];
404                         int colonne    = probleme->noeuds[probleme->
                                elements[i].indices[noeud2]].indicesK[
                                direction2];
```

Annexes :

Code C : main.c

```
405                             matriceRaideur->contenu[ligne][colonne] +=
                                    matriceElement->contenu[noeud1 * Dimension +
                                     direction1][noeud2 * Dimension + direction2
                                    ];
406                         }
407                     }
408                 }
409             }
410
411             supprimer_matrice(matriceElement);
412         }
413
414         return matriceRaideur;
415 }
416
417 matrice* recuperation_forces_connues(probleme_t* probleme)
418 {
419         matrice* forces_connues = creer_matrice(probleme->degreeDeLiberte,
                1); // vecteur colonne
420
421         int indice = 0;
422
423         // la création de la matrice est fortement liée au tri choisi
424
425         for (int i = 0; i < probleme->nbreNoeuds; ++i)
```

Annexes :

Code C : main.c

```
426     {
427         for (int d = 0; d < Dimension; ++d)
428         {
429             if (probleme->noeuds[i].typeConstraite[d] == 1)
430             {
431                 forces_connues->contenu[indice][0] = probleme->noeuds[i
                        ].force->contenu[d][0];
432                 ++indice;
433             }
434         }
435     }
436
437     return forces_connues;
438 }
439
440 matrice* recuperation_deplacements_connus(probleme_t* probleme)
441 {
442     matrice* deplacements_connus = creer_matrice(probleme->
            degreeDeContrainte, 1); // vecteur colonne
443
444     int indice = 0;
445
446     // la création de la matrice est fortement liée au tri choisi
447
448     for (int i = 0; i < probleme->nbreNoeuds; ++i)
```

Annexes :

Code C : main.c

```
449     {
450         for (int d = 0; d < Dimension; ++d)
451         {
452             if (probleme->noeuds[i].typeConstraite[d] == -1)
453             {
454                 deplacements_connus->contenu[indice][0] = probleme->
                        noeuds[i].deplacement->contenu[d][0];
455                 ++indice;
456             }
457         }
458     }
459
460     return deplacements_connus;
461 }
462
463 void applique_elements_finis(char* lienDonnees, char* lienSortie)
464 {
465     // précalculs
466
467     probleme_t* probleme = lecture_donnees(lienDonnees);
468
469     printf("donnees lues\n");
470
471     matrice* matriceRaideur = creation_matrice_raideur(probleme);
472
```

Annexes :

Code C : main.c

```
473     printf("matrice␣créée\n");
474
475     // récupération des matrices
476
477     matrice* forces_connues       = recuperation_forces_connues(probleme)
            ;
478     matrice* deplacements_connus = recuperation_deplacements_connus(
            probleme);
479     matrice* k1                   = sous_matrice(matriceRaideur, 0, 0,
            probleme->degreeDeLiberte, probleme->degreeDeLiberte);
480     matrice* k2                   = sous_matrice(matriceRaideur, 0,
            probleme->degreeDeLiberte, probleme->degreeDeLiberte, probleme->
            degreeDeContrainte);
481     matrice* k3                   = sous_matrice(matriceRaideur, probleme
            ->degreeDeLiberte, 0, probleme->degreeDeContrainte, probleme->
            degreeDeLiberte);
482     matrice* k4                   = sous_matrice(matriceRaideur, probleme
            ->degreeDeLiberte, probleme->degreeDeLiberte, probleme->
            degreeDeContrainte, probleme->degreeDeContrainte);
483
484     // calculs matriciels
485
486     // A / F = Fc - K2 x Uc
487     matrice* k2_x_uc = mult_matrice(k2, deplacements_connus);
488
```

Annexes :

Code C : main.c

```
489     matrice* force_temp = soustract_matrice(forces_connues, k2_x_uc);
490
491     // B / Ui = K1^-1 x F
492
493     matrice* invK1 = inv_matrice(k1, false);
494
495     matrice* deplacements_inconnus = mult_matrice(invK1, force_temp);
496
497     // C / Fi = K3 x Ui + K4 x Up
498
499     matrice* mult1 =  mult_matrice(k3, deplacements_inconnus);
500     matrice* mult2 =  mult_matrice(k4, deplacements_connus);
501
502     matrice* forces_inconnues = add_matrice(mult1, mult2);
503
504     // mise à jour des noeuds
505
506     int idTampDepl = 0;
507     int idTampForc = 0;
508
509     for (int i = 0; i < probleme->nbreNoeuds; ++i)
510     {
511         for (int d = 0; d < Dimension; ++d)
512         {
513             if (probleme->noeuds[i].typeContraite[d] == 1)
```

Annexes :

Code C : main.c

```
514                 {
515                     probleme->noeuds[i].deplacement->contenu[d][0] =
                            deplacements_inconnus->contenu[idTampDepl][0];
516                     ++idTampDepl;
517                 }
518                 else
519                 {
520                     probleme->noeuds[i].force->contenu[d][0] =
                            forces_inconnues->contenu[idTampForc][0];
521                     ++idTampForc;
522                 }
523             }
524         }
525
526         // free et retour
527
528         ecrit_resultat(lienSortie, probleme);
529
530         supprimer_matrice(matriceRaideur);
531         supprimer_matrice(forces_connues);
532         supprimer_matrice(deplacements_connus);
533         supprimer_matrice(k1);
534         supprimer_matrice(k2);
535         supprimer_matrice(k3);
536         supprimer_matrice(k4);
```

Annexes :

Code C : main.c

```
537     supprimer_matrice(k2_x_uc);
538     supprimer_matrice(force_temp);
539     supprimer_matrice(invK1);
540     supprimer_matrice(deplacements_inconnus);
541     supprimer_matrice(mult1);
542     supprimer_matrice(mult2);
543     supprimer_matrice(forces_inconnues);
544 
545     supprime_probleme(probleme);
546 
547     return;
548 }
549 
550 
551 
552 /* - Main (exemple) - */
553 
554 int main()
555 {
556     srand(time(NULL));
557 
558     applique_elements_finis("donnees.txt", "resultat.txt");
559 
560     printf("Termine.\n");
561     return 0;
```

Annexes :

Code C : main.c

```
562 }
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
1
2  (*Pour Windows*)
3  (*
4  #load "graphics.cma";;
5  Graphics.open_graph "800x600";;
6  open Graphics;;
7  open_graph "720x1280";;
8  *)
9
10 (*Pour Linux*)
11 #use "topfind";;
12 #require "graphics";;
13 open Graphics;;
14 let hauteur=720 and largeur=1280;;
15 open_graph "";;
16 resize_window largeur hauteur;;
17
18 (*------------Types,variables et fonctions outils--------*)
19
20
21 type point    = {x: float;  y: float;  z: float};;
22 type vecteur = {vx: float; vy: float; vz: float};;
23
24
25 let x0    = ref (float_of_int (size_x()/2))
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
26 and y0    = ref (float_of_int (size_y()/2))
27 and zoom = ref 150.;;
28
29
30 let base = ref ({vx = 1.; vy = 0.; vz = 0.},
31 {vx = 0.; vy = 1.; vz = 0.},
32 {vx = 0.; vy = 0.; vz = 1.});;
33
34
35 let vecteur pt1 pt2 = {vx = (pt2.x -. pt1.x); vy = (pt2.y -. pt1.y); vz
       = (pt2.z -. pt1.z)};;
36
37
38 let produit_scalaire vct1 vct2 = vct1.vx *. vct2.vx +. vct1.vy *. vct2.
       vy +. vct1.vz *. vct2.vz;;
39
40
41 let norme vct = sqrt(vct.vx**2. +. vct.vy**2. +. vct.vz**2.);;
42
43
44 let unitaire vct = {vx = (vct.vx /. (norme vct));
45                                                  vy = (vct.vy /. (norme
                                                         vct));
46                                                  vz = (vct.vz /. (norme
                                                         vct))};;
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
47
48
49 let produit_vectoriel vct1 vct2 =
50         {vx = (vct1.vy *. vct2.vz -. vct1.vz *. vct2.vy);
51          vy = (vct1.vz *. vct2.vx -. vct1.vx *. vct2.vz);
52          vz = (vct1.vx *. vct2.vy -. vct1.vy *. vct2.vx)};;
53
54 let dans_base pt bse = let vctb1, vctb2, vctb3 = bse and origine = {x =
       0.; y = 0.; z = 0.} in
55         {x = (produit_scalaire (vecteur origine pt) vctb1);
56          y = (produit_scalaire (vecteur origine pt) vctb2);
57          z = (produit_scalaire (vecteur origine pt) vctb3)};;
58
59
60 let rotation_x vct theta =
61         {vx = vct.vx;
62          vy = vct.vy *. cos theta -. vct.vz *. sin theta;
63          vz = vct.vy *. sin theta +. vct.vz *. cos theta};;
64
65 let rotation_y vct theta =
66         {vx = vct.vx *. cos theta +. vct.vz *. sin theta;
67          vy = vct.vy;
68          vz = vct.vz *. cos theta -. vct.vx *. sin theta};;
69
70 let rotation_base_x theta = let vct1, vct2, vct3 = !base in
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
71          base := ((rotation_x vct1 theta),
72                                  (rotation_x vct2 theta),
73                                  (rotation_x vct3 theta));;
74
75  let rotation_base_y theta = let vct1, vct2, vct3 = !base in
76          base := ((rotation_y vct1 theta),
77                                  (rotation_y vct2 theta),
78                                  (rotation_y vct3 theta))
79  ;;
80
81  let projette pt = (int_of_float (!x0 +. !zoom *. pt.x), int_of_float (!
        y0 +. !zoom *. pt.y));;
82
83  let make_point (x,y,z) = { x = x; y=y; z=z;};;
84
85  type element = int * int * float * float;; (*indice du noeud1, indice
        noeud2, module young, section*)
86
87  let make_element indice_noeud1 indice_noeud2 mod_young section = (
        indice_noeud1,indice_noeud2,mod_young,section);;
88
89  type item_affichable = Arete of (point*point*int*int) | Noeud of (point*
        int*int);; (*Arete(point de départ, point d'arrivée, epaisseur,
        couleur) et Noeud(point, rayon, couleur)*)
90
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
91
92  (*COULEURS ET EPAISSEUR DES ITEMS*)
93
94  (*Fonction pour resize des intervalles (proportionnalité)*)
95  let map debut1 fin1 debut2 fin2 x =
96    let t = (x -. debut1)/.(fin1 -. debut1) in
97    (1. -. t)*. debut2 +. t*.fin2
98  ;;
99
100
101 (*Détermine la couleur d'un noeud à l'aide de la norme de la force
        appliquée en ce noeud*)
102 let couleurs_noeuds noeuds forces =
103         let normes_forces = Array.map norme forces in
104         (*let max_norme_force force1 force2 = max (norme force1) (norme
              force2) in
105         let min_norme_force force1 force2 = min (norme force1) (norme
              force2) in *)
106   let max_force  = Array.fold_left max normes_forces.(0) normes_forces
107   and min_force = Array.fold_left min normes_forces.(0) normes_forces
108 in
109 let tab_couleurs = Array.map (fun norme_force -> let teinte =
        int_of_float (map min_force max_force 0. 255. norme_force) in rgb
        teinte 0 (255 - teinte) ) normes_forces in
110 tab_couleurs;; (*Censé faire un dégradé du bleu au rouge*)
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
111
112 (*Calcule l'emplacement des noeuds après application de la force,
        ajoutant les déplacements*)
113 let noeuds_deplaces noeuds deplacements =
114                 Array.map2 (fun point vecteur -> make_point ((point.x +.
                        vecteur.vx),(point.y +. vecteur.vy),(point.y +.
                        vecteur.vy)) ) noeuds deplacements
115 ;;
116
117 (*Calcule l'épaisseur à afficher des aretes. Attention, renvoie le max
        et le min des sections (unité d'origine)*)
118 let epaisseurs_elements elements =
119         let sections = Array.map (fun (i1,i2,young,section) -> section)
                elements in
120 let max_section = Array.fold_left (fun section accu_section -> max
        section accu_section) 0. sections
121 and min_section = Array.fold_left (fun section accu_section -> min
        section accu_section) infinity sections
122 and min_epaisseur = 2. (*Constantes d'épaisseurs des traits*)
123 in let max_epaisseur = (max_section/. min_section) *. min_epaisseur
124 in
125 let tab_epaisseurs = Array.map (fun section -> int_of_float (map
        min_section max_section min_epaisseur max_epaisseur section) )
        sections in
126
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
127 tab_epaisseurs,min_section,max_section
128 ;;
129
130 (*Crée un tableau des items à afficher (noeuds et arêtes), qui sera trié
        par la cote moyenne ou la cote en fonction de si c'est un poin ou
    une arete.
131         Contient les aretes et noeuds avant et après application des
              forces*)
132 let make_items_affichables elements noeuds forces deplacements=
133         let epaisseurs,_,_ = epaisseurs_elements elements in
134         let couleurs = couleurs_noeuds noeuds forces in
135         let noeuds_depl = noeuds_deplaces noeuds deplacements in
136         let tab_aretes_originelles = Array.map2 (fun (i1,i2,young,
              section) epaisseur -> Arete(noeuds.(i1),noeuds.(i2),
              epaisseur,rgb 127 127 127) ) elements epaisseurs
137         and tab_noeuds_originels = Array.map (fun point -> Noeud(point
              ,5,rgb 127 127 127)) noeuds
138         and tab_aretes_deplacees = Array.map2 (fun (i1,i2,young,section)
               epaisseur -> Arete(noeuds_depl.(i1),noeuds_depl.(i2),
              epaisseur,black) ) elements epaisseurs
139         and tab_noeuds_deplaces = Array.map2 (fun point couleur -> Noeud
              (point,7,couleur)) noeuds_depl couleurs
140
141         in
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
142         Array.concat [tab_aretes_originelles;tab_noeuds_originels;
                tab_aretes_deplacees;tab_noeuds_deplaces]
143 ;;
144
145 (*Fonction auxiliaire pour tracer une arete*)
146 let trace_arete point1 point2 epaisseur couleur =
147         let pt1 = dans_base point1 !base
148         and pt2 = dans_base point2 !base
149   in
150         let x1,y1 = projette pt1
151         and x2,y2 = projette pt2 in
152         set_color couleur;
153         set_line_width epaisseur;
154         moveto x1 y1;
155         lineto x2 y2;
156 ;;
157
158 (*Fonction auxiliaire pour tracer un noeud*)
159 let trace_noeud point rayon couleur =
160         let epaisseur_trait = max 1 (int_of_float(float_of_int (rayon)
                *. 0.2)) in
161         let pt = dans_base point !base in
162         let x,y = projette pt in
163         set_color couleur;
164         fill_circle x y rayon;
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
165         set_color black;
166
167         set_line_width epaisseur_trait;
168         draw_circle x y rayon
169 ;;
170
171 (*------Algorithme du Peintre--------------*)
172
173
174 (*Profondeur d'un point dans la direction z*)
175 let cote pt = let proj = dans_base pt !base in proj.z;;
176
177 (*Profondeur pour une arete*)
178 let cote_moyenne (point1, point2) = (cote point1 +. cote point2)/. 2.;;
179
180 (*Tri des items pour l'algo du peintre*)
181 let tri tab clef = let taille = (Array.length tab) - 1 in
182         for i = 1 to taille do
183                 let j = ref i and check = clef tab.(i) and temp = tab.(i
                        ) in
184                 while !j > 0 && clef (tab.(!j-1)) > check do
185                         tab.(!j) <- tab.(!j-1);
186                         j := !j - 1;
187                 done;
188                 tab.(!j) <- temp;
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
189         done
190 ;;
191
192 let tri_items items_a_afficher =
193         let clef_tri item = match item with
194                 |Arete(point1,point2,epaisseur,couleur_arete) ->
                        cote_moyenne (point1,point2)
195                 |Noeud(p,rayon,couleur_noeud) -> cote p
196         in
197         tri items_a_afficher clef_tri
198 ;;
199
200
201 (*Affichage des items dans le bon ordre*)
202 let peintre_items items_a_afficher = let taille = Array.length
        items_a_afficher in
203         tri_items items_a_afficher;
204         for i = taille-1 downto 0 do
205                 (*print_string "Traçage de l'item n° : ";print_int i;
                        print_newline();*)
206                 let item= items_a_afficher.(i) in
207                 match item with
208                 | Noeud(point,rayon,couleur) -> trace_noeud point rayon
                        couleur
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
209                     | Arete(point1,point2,epaisseur,couleur) -> trace_arete
                            point1 point2 epaisseur couleur
210         done
211 ;;
212
213
214 (*-----Récuperation des données dans un fichier extérieur----------*)
215
216
217 (*Type tableau dynamique pour faciliter la récupération des données*)
218 type 'a tableau_dynamique = {mutable support: 'a array;
219                                                                    mut
220
221 let make_td element = {
222         support = Array.make 16 element; taille = 0};;
223
224 let ajoute td valeur =
225         if td.taille <> Array.length td.support then
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
226                 begin
227                 td.support.(td.taille) <- valeur;
228                 td.taille <- td.taille + 1;
229                 end
230         else
231                 begin
232                 let new_support = Array.make (td.taille*2) valeur in
233                 for i = 0 to (td.taille-1) do
234                         new_support.(i) <- td.support.(i);
235                 done;
236                 td.support <- new_support;
237                 td.taille <- td.taille + 1;
238                 end;;
239
240
241 (*Fonction qui lit le fichier contenant les données et qui renvoie les
        tableaux contenant :
242         -les noeuds (indicés par i)
243         -le déplacement des noeuds (deplacement du noeud i à l'indice i
                )
244         -les forces appliquées au noeud i
245         -les elements, ie (indice_noeud1,indice_noeud2,module_young,
                section)*)
246 let lecture_fichier nomFichier =
247         let fichier = open_in nomFichier in
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
248         let point_generique = make_point (0.,0.,0.) in
249         let element_generique = make_element 0 0 0. 0.  in
250         let deplacement_generique = vecteur point_generique
                point_generique in
251         let force_generique = vecteur point_generique point_generique in
252         let noeuds  = make_td point_generique in
253         let deplacements = make_td deplacement_generique in
254         let forces = make_td force_generique in
255         let elements = make_td element_generique in
256
257         let ligne = ref (input_line fichier) in
258         let nb_noeuds, nb_elements = Scanf.sscanf !ligne "%d;%d" (fun n1
                n2 -> (n1,n2)) in
259
260         for i = 0 to nb_noeuds-1 do
261             ligne:= input_line fichier;
262             let x,dx,fx = Scanf.sscanf !ligne "%f;%f;%f" (fun x dx
                    fx-> (x,dx,fx)) in
263             ligne:= input_line fichier;
264             let y,dy,fy = Scanf.sscanf !ligne "%f;%f;%f" (fun y dy
                    fy -> (y,dy,fy)) in
265             ligne:= input_line fichier;
266             let z,dz,fz = Scanf.sscanf !ligne "%f;%f;%f" (fun z dz
                    fz-> (z,dz,fz)) in
267
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
268                         ajoute noeuds (make_point (x,y,z));
269                         ajoute deplacements (vecteur point_generique (
                                make_point (dx,dy,dz)));
270                         ajoute forces (vecteur point_generique (
                                make_point (fx,fy,fz)));
271         done;
272         for i = 0 to nb_elements-1 do
273                 ligne:=input_line fichier;
274                 let element = Scanf.sscanf !ligne "%d;%d;%f;%f" (fun i1
                        i2 module_young section -> (i1,i2,module_young,
                        section)) in
275                         ajoute elements element;
276         done;
277         close_in fichier;
278         let coupe_tableau_dyn tab = Array.sub (tab.support) 0 (tab.
                taille)  in
279   coupe_tableau_dyn noeuds,coupe_tableau_dyn deplacements,
          coupe_tableau_dyn forces ,coupe_tableau_dyn elements
280 ;;
281
282
283
284 (*Boucle pour afficher la structure et la faire tourner à l'aide du
        clavier*)
285 let en_sync_items items_a_afficher =
```

Annexes :

Code Ocaml : Affichage_ FEM_ 3D.ml

```
286         auto_synchronize false;
287         display_mode false;
288         peintre_items items_a_afficher;
289
290                 while true do
291                         let event = wait_next_event [Key_pressed] in let
                                key = event.key in
292                         if key = 's' then y0 := !y0 -. 5.;
293                         if key = 'z' then y0 := !y0 +. 5.;
294                         if key = 'q' then x0 := !x0 -. 5.;
295                         if key = 'd' then x0 := !x0 +. 5.;
296                         if key = 'o' then rotation_base_y (0.05);
297                         if key = 'l' then rotation_base_y (-0.05);
298                         if key = 'k' then rotation_base_x (-0.05);
299                         if key = 'm' then rotation_base_x (0.05);
300                         if key = 'a' then zoom := !zoom +. 5.;
301                         if key = 'e' then zoom := !zoom -. 5.;
302                         clear_graph ();
303                         peintre_items items_a_afficher;
304                         synchronize ();
305                 done
306 ;;
307
308
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
309 (*Fonction main : récupere les tableaux et lance la fonction
        en_sync_items.*)
310 let main () =
311 let noeuds,deplacements,forces,elements = lecture_fichier "resultat.txt"
        in
312 print_string "Nombre d'éléments : ";
313 print_int (Array.length elements); print_newline();
314
315 (*let noeuds2 = noeuds_deplaces noeuds deplacements in
316 let tous_noeuds = Array.append noeuds noeuds2 in
317 let tous_elements = Array.append elements elements in
318 *)
319 let items_a_afficher = make_items_affichables elements noeuds forces
        deplacements in
320 (*affiche_aretes_elements elements noeuds;*)
321 (*set_line_width 10;
322 lineto (size_x()/2) (size_y()/2); *)
323 synchronize ();
324 en_sync_items items_a_afficher;
325 ;;
326
327 main();;
```