

Présentation de TIPE

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

Thomas CREUSET

numéro de candidat : 11909

- 1 Choix du projet
 - La construction en ville: un enjeu pour les ingénieurs
 - Modélisation informatique
 - Méthode des éléments finis
- 2 Méthode des éléments finis
 - Méthode de Galerkin
 - Méthode des ressorts
 - Loi de Hooke
 - Généralisation, matrice de raideur
 - Deux astuces
 - Affichage
- 3 Optimisation
 - Analogique et numérique, défauts et qualités
 - Retours expérimentaux
 - Une solution inattendue et conclusion
- 4 Annexes
 - Code C
 - Code Ocaml

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 :

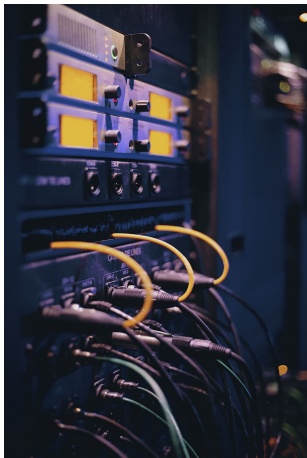
- Stabilité



Source: wallpaperflare

Choix du projet :

Modélisation informatique



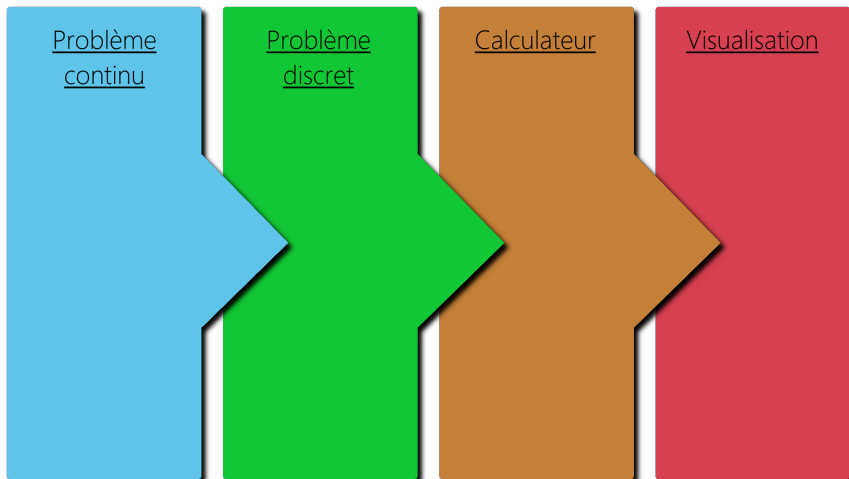
Source: wallpaperflare

Intérêts de la modélisation informatique :

- Coût
- Durée
- Aspects spécifiques
- Paramétrisation précise

Choix du projet :

Méthode des éléments finis



Méthode des éléments finis

Méthode des éléments finis :

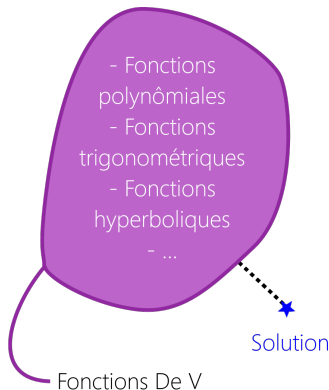
Méthode de Galerkin

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

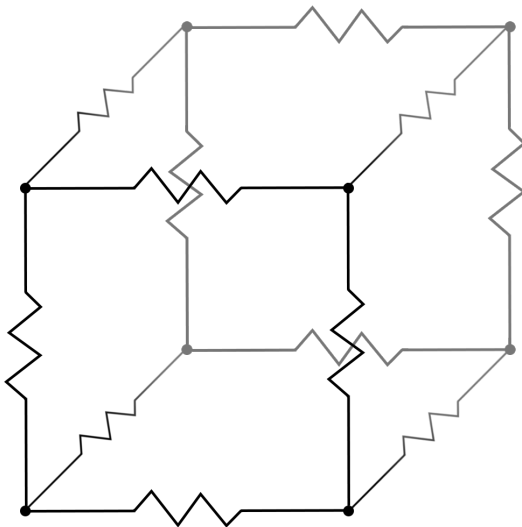
Défauts :

- Connaissances mathématiques en calcul différentiel poussé
- Difficile à généraliser



Méthode des éléments finis :

Méthode des ressorts



Méthode des éléments finis :

Loi de Hooke

Loi de Hooke

$$\vec{F} = -k \cdot \Delta \ell \cdot \vec{u} \quad (1)$$

Constante de raideur

$$k = \frac{A \cdot E}{L} \quad (2)$$

avec :

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

Méthode des éléments finis :

Généralisation, matrice de raideur

Loi de Hooke matricielle

$$F = K \cdot U \quad (3)$$

Forme détaillée

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

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

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

Méthode des éléments finis :

Deux astuces

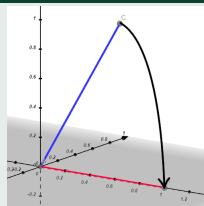
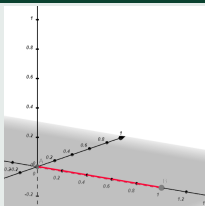
Récupération des forces et des déplacements inconnus

$$F_c = K_1 \cdot U_i + K_2 \cdot U_c \quad (5)$$

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

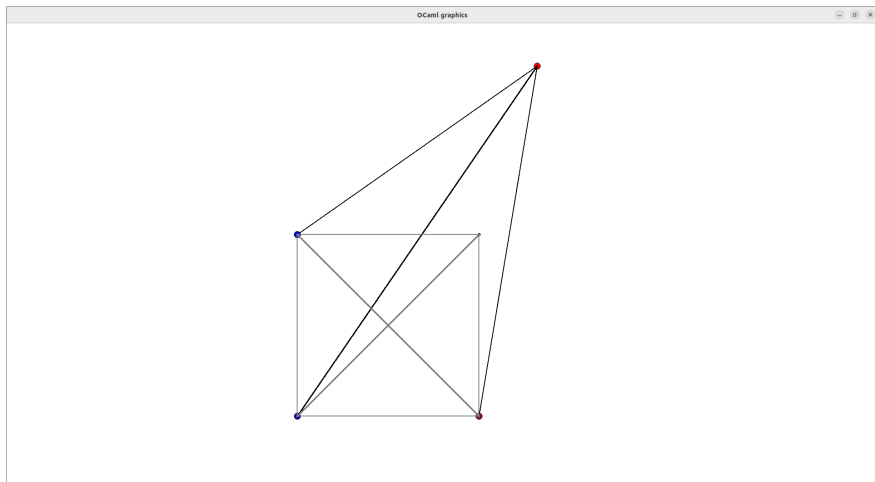
$$et : F_i = K_3 \cdot U_i + K_4 \cdot U_c \quad (7)$$

Rotation vers cas par défaut



Méthode des éléments finis :

Affichage



Optimisation

Optimisation :

Analogique et numérique, défauts et qualités

Points de divergence des méthodes :

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

Optimisation :

Retours expérimentaux

Pas encore fait

La mécanique quantique une solution pour le futur

Algorithme d'inversion HHL (Harrow, Hassidim, Lloyd) :

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

¹L'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`

```
1  /* -Importations - */
2
3  #include <stdio.h>
4  #include <stdbool.h>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <math.h>
```

Annexes :

Code C : module `_matrice.h`

```
1  /* ~Matrices~ */
2
3  /* -Importations- */
4
5  #include "standard_lib.h"
6
7
8  /* -Types et structures- */
9
10 typedef double valeur;
11
12 struct matrice_s {
13     int lignes;
14     int colonnes;
15     valeur** contenu;
16 };
17
18 typedef struct matrice_s matrice;;
19
20 /* -Déclarations fonctions (f) et procédures (p)- */
21
22 matrice* creer_matrice(int lignes, int colonnes);
23 // f - crée une matrice de dimension 'lignes'x'colonnes' initialisé à 0
24 // (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* soustrait_matrice(matrice* matriceA, matrice* matriceB);
35 // f - soustrait la matrice 'matriceB' à la matrice 'matriceB' de maniè
    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

```
1  /* -Fichier entête- */
2
3  #include "module_matrice.h"
4
5
6  /* -Fonctions- */
7
8  matrice* creer_matrice(int lignes, int colonnes)
9  {
10     if (lignes <= 0 || colonnes <= 0)
11     {
12         fprintf(stderr, "création impossible:\n");
13         fprintf(stderr, "\t-> les tailles 'lignes' et 'colonnes' doivent
14             -être des entiers non nuls.\n");
15         fprintf(stderr, "\t\t'lignes': %d\n", lignes);
16         fprintf(stderr, "\t\t'colonnes': %d\n", colonnes);
17         exit(EXIT_FAILURE);
18     }
19
20     matrice* matriceSortie = malloc(sizeof(matrice));
21     matriceSortie->lignes = lignes;
22     matriceSortie->colonnes = colonnes;
23     matriceSortie->contenu = malloc(sizeof(valeur*)*lignes);
24
25     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* soustraire_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; ++
107         {
108             matriceSortie->contenu[ligne][colonne] = matriceA->contenu[
109                 ligne][colonne] - matriceB->contenu[ligne][colonne];
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
121             \t colonnes que la matrice 'matriceB' a de lignes.\n");
122         fprintf(stderr, "\t\t 'matriceA' : %d colonnes\n", matriceA->
123             colonnes);
124         fprintf(stderr, "\t\t 'matriceB' : %d lignes\n", matriceB->
125             lignes);
126         exit(EXIT_FAILURE);
127     }
128 }
```

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; ++
169         colonne)
170     {
171         matriceSortie->contenu[ligne][colonne] = scalaire *
172             matriceEntree->contenu[ligne][colonne];
173     }
174     return matriceSortie;
175 }
176
177 valeur det_matrice(matrice* matriceEntree)
178 {
179     if (matriceEntree->colonnes != matriceEntree->lignes)
180     {
181         fprintf(stderr, "calculé du déterminant impossible:\n");
182         fprintf(stderr, "\t-> la matrice 'matriceEntree' doit avoir
183             autant de colonnes que de lignes.\n");
184         fprintf(stderr, "\t\t'matriceEntree': %d lignes\n",
185             matriceEntree->lignes);
186         fprintf(stderr, "\t\t'matriceEntree': %d colonnes\n",
187             matriceEntree->colonnes);
188         exit(EXIT_FAILURE);
189     }
190 }
```


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 possédant 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 transposé 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\t'ligne1': %d\n", ligne1);
313         fprintf(stderr, "\t\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
328         >= matriceEntree->colonnes || colonne2 < 0)
329     {
330         fprintf(stderr, "échange des colonnes impossible:\n");
331         fprintf(stderr, "\t-> les colonnes 'colonne1' et 'colonne2'
332             doivent exister.\n");
333         fprintf(stderr, "\t\t'colonne1': %d\n", colonne1);
334         fprintf(stderr, "\t\t'colonne2': %d\n", colonne2);
335         exit(EXIT_FAILURE);
336     }
337     for (int ligne = 0; ligne < matriceEntree->lignes; ++ligne)
338     {
339         valeur stockageTemp = matriceEntree->contenu
340             [ligne][colonne1];
341         matriceEntree->contenu[ligne][colonne1] = matriceEntree->contenu
342             [ligne][colonne2];
343         matriceEntree->contenu[ligne][colonne2] = stockageTemp;
344     }
345 }
346 void combinaison_lignes(matrice* matriceEntree, int ligneDest, valeur
347     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);
385     }
386 }
```

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, "calcul 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, "calcul de l'inverse impossible:\n");
445         fprintf(stderr, "\t-> la matrice 'matriceEntree' est de dé
             terminant nul.\n");
446         fprintf(stderr, "\t\t'déterminant': %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         }
471     }
472 }
```

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

```
25 #include <stdbool.h>
26 #include <stdlib.h>
27 #include <time.h>
28 #include <math.h>
29 #include <string.h>
30 #include "module_matrice.h"
31
32
33 /* - Constantes - */
34
35 #define Dimension 3 // s.u.
36 #define NoeudsParElement 2 // s.u.
37
38
39 /* - Structures et types - */
40
41 typedef struct
42 {
43     matrice* position;
44     matrice* déplacement;
45     matrice* force;
46     int* typeConstraite;
47     int* indicesK;
48 } noeud_t;
49
```

Annexes :

Code C : main.c

```
50 typedef struct
51 {
52     int* indices;
53     double e; // Pa module de Young
54     double a; // m^2 section
55 } element_t;
56
57 typedef struct
58 {
59     noeud_t* noeuds;
60     element_t* elements;
61     int nbreNoeuds;
62     int nbreElements;
63     int degreeDeLiberte;
64     int degreeDeContrainte;
65 } probleme_t;
66
67
68 /* - Décalrations - */
69
70 probleme_t* lecture_donnees(char* lien);
71 /* f - récupère les données à l'adresse fournie */
72
73 void ecrit_resultat(char* lien, probleme_t* probleme);
74 /* 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->degreedeccontrainte = 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     ;
128     probleme->noeuds[i].deplacement   = creer_matrice(Dimension, 1)
129     ;
130     probleme->noeuds[i].force          = creer_matrice(Dimension, 1)
131     ;
132     probleme->noeuds[i].typeConstraite = malloc(sizeof(int) *
133     Dimension);
134     probleme->noeuds[i].indicesK      = malloc(sizeof(int) *
135     Dimension);
136
137     // Lecture du noeud
138
139     for (int d = 0; d < Dimension; ++d)
140     {
141         fscanf(fichier, "%lf;%d;%lf\n", &(probleme->noeuds[i].
142         position->contenu[d][0]), &(probleme->noeuds[i].
143         typeConstraite[d]), &valeur);
144
145         if (probleme->noeuds[i].typeConstraite[d] == 1)
146         {
147             ++(probleme->degreDeLiberte);
148         }
149     }
150 }
```

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

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 supprimer_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 longueurProj = sqrt(deltax * deltax + deltay * deltay);
273
274     double cosinus;
275     double sinus;
```

Annexes :

Code C : main.c

```
276
277     if (sqrt(longueurProj * longueurProj) <= 0.001) // on évite la
        division par 0
278     {
279         cosinus = 1.0;
280         sinus   = 0.0;
281     }
282     else
283     {
284         cosinus = deltax / longueurProj;
285         sinus   = deltax / longueurProj;
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 printf("%lf□%lf□%lf,□%lf□%lf□%lf", x1, nouvellePositions->contenu
        [1][0], z1, x2, nouvellePositions->contenu[4][0], z2);
319
320 deltax = x2 - x1;
```

Annexes :

Code C : main.c

```
321 double deltaz = z2 - z1;
322
323 double longueur = sqrt(deltax * deltax + deltaz * deltaz); // conserv
    ée par rotation (et plus de composante selon y)
324
325 if (sqrt(longueur * longueur) <= 0.001) // on évite la division par 0
326 {
327     cosinus = 1.0;
328     sinus   = 0.0;
329 }
330 else
331 {
332     cosinus = deltax / longueur;
333     sinus   = deltaz / longueur;
334 }
335
336 // - création de la matrice de rotation R(-angle) (B)
337
338 matrice* mat_B = creer_matrice(Dimension * NoeudsParElement,
    Dimension * NoeudsParElement);
339
340 for (int n = 0; n < NoeudsParElement; ++n)
341 {
342     mat_B->contenu[0 + Dimension * n][0 + Dimension * n] = cosinus;
343     mat_B->contenu[0 + Dimension * n][2 + Dimension * n] = -sinus;
```


Annexes :

Code C : main.c

```
344     mat_B->contenu[1 + Dimension * n][1 + Dimension * n] = 1.0;
345     mat_B->contenu[2 + Dimension * n][0 + Dimension * n] = sinus;
346     mat_B->contenu[2 + Dimension * n][2 + Dimension * n] = cosinus;
347 }
348
349 // Calcul de la matrice de raideur de l'élément
350
351 // - création de la matrice dans la base canonique (K)
352
353 matrice* mat_K = creer_matrice(Dimension * NoeudsParElement,
    Dimension * NoeudsParElement);
354
355 double constante = (probleme->elements[i].e) * (probleme->elements[i
    ].a) / longueur;
356
357 mat_K->contenu[0][0] = constante;
358 mat_K->contenu[0][Dimension] = -constante;
359 mat_K->contenu[Dimension][0] = -constante;
360 mat_K->contenu[Dimension][Dimension] = constante;
361
362 // - calcule de la matrice dans la base tournée tA x tB x K x B x A
    = t(BA) x K x BA
363
364 matrice* mat_BA = mult_matrice(mat_B, mat_A);
365 matrice* mat_tBA = transp_matrice(mat_BA);
```

Annexes :

Code C : main.c

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

Annexes :

Code C : main.c

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

Annexes :

Code C : main.c

```
407         matriceRaideur->contenu[ligne][colonne] +=
            matriceElement->contenu[noeud1 * Dimension +
                direction1][noeud2 * Dimension + direction2
            ];
408     }
409 }
410 }
411 }
412
413     supprimer_matrice(matriceElement);
414 }
415
416     return matriceRaideur;
417 }
418
419 matrice* recuperation_forces_connues(probleme_t* probleme)
420 {
421     matrice* forces_connues = creer_matrice(probleme->degreDeLiberte,
        1); // vecteur colonne
422
423     int indice = 0;
424
425     // la création de la matrice est fortement liée au tri choisi
426
427     for (int i = 0; i < probleme->nbreNoeuds; ++i)
```

Annexes :

Code C : main.c

```
428 {
429     for (int d = 0; d < Dimension; ++d)
430     {
431         if (probleme->noeuds[i].typeConstraite[d] == 1)
432         {
433             forces_connues->contenu[indice][0] = probleme->noeuds[i]
434                 .force->contenu[d][0];
435             ++indice;
436         }
437     }
438
439     return forces_connues;
440 }
441
442 matrice* recuperation_deplacements_connus(probleme_t* probleme)
443 {
444     matrice* deplacements_connus = creer_matrice(probleme->
445         degreeDeContrainte, 1); // vecteur colonne
446
447     int indice = 0;
448
449     // la création de la matrice est fortement liée au tri choisi
450
451     for (int i = 0; i < probleme->nbreNoeuds; ++i)
```

Annexes :

Code C : main.c

```
451 {
452     for (int d = 0; d < Dimension; ++d)
453     {
454         if (probleme->noeuds[i].typeConstraite[d] == -1)
455         {
456             déplacements_connus->contenu[indice][0] = probleme->
457                 noeuds[i].deplacement->contenu[d][0];
458             ++indice;
459         }
460     }
461
462     return déplacements_connus;
463 }
464
465 void applique_elements_finis(char* lienDonnees, char* lienSortie)
466 {
467     // précalculs
468
469     probleme_t* probleme = lecture_donnees(lienDonnees);
470
471     printf("donnees_lues\n");
472
473     matrice* matriceRaideur = creation_matrice_raideur(probleme);
474 }
```

Annexes :

Code C : main.c

```
475 printf("matrice créée\n");
476
477 // récupération des matrices
478
479 matrice* forces_connues      = recuperation_forces_connues(probleme)
    ;
480 matrice* déplacements_connus = recuperation_deplacements_connus(
    probleme);
481 matrice* k1                  = sous_matrice(matriceRaideur, 0, 0,
    probleme->degreDeLiberte, probleme->degreDeLiberte);
482 matrice* k2                  = sous_matrice(matriceRaideur, 0,
    probleme->degreDeLiberte, probleme->degreDeLiberte, probleme->
    degreDeContrainte);
483 matrice* k3                  = sous_matrice(matriceRaideur, probleme
    ->degreDeLiberte, 0, probleme->degreDeContrainte, probleme->
    degreDeLiberte);
484 matrice* k4                  = sous_matrice(matriceRaideur, probleme
    ->degreDeLiberte, probleme->degreDeLiberte, probleme->
    degreDeContrainte, probleme->degreDeContrainte);
485
486 // calculs matriciels
487
488 //  $A / F = F_c - K_2 \times U_c$ 
489 matrice* k2_x_uc = mult_matrice(k2, déplacements_connus);
490
```

Annexes :

Code C : main.c

```
491 matrice* force_temp = soustrait_matrice(forces_connues, k2_x_uc);
492
493 // B / U_i = K1^-1 x F
494
495 matrice* invK1 = inv_matrice(k1, false);
496
497 matrice* déplacements_inconnus = mult_matrice(invK1, force_temp);
498
499 // C / F_i = K3 x U_i + K4 x U_p
500
501 matrice* mult1 = mult_matrice(k3, déplacements_inconnus);
502 matrice* mult2 = mult_matrice(k4, déplacements_connus);
503
504 matrice* forces_inconnues = add_matrice(mult1, mult2);
505
506 // mise à jour des noeuds
507
508 int idTampDepl = 0;
509 int idTampForc = 0;
510
511 for (int i = 0; i < probleme->nbreNoeuds; ++i)
512 {
513     for (int d = 0; d < Dimension; ++d)
514     {
515         if (probleme->noeuds[i].typeConstraite[d] == 1)
```


Annexes :

Code C : main.c

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

Annexes :

Code C : main.c

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

Annexes :

Code C : main.c

564 }

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
1 (*Pour Windows*)
2 #load "graphics.cma";;
3 Graphics.open_graph "800x600";;
4 open Graphics;;
5 open_graph "720x1280";;
6
7
8 (*Pour Linux
9 #use "topfind";;
10 #require "graphics";;
11 open Graphics;;
12 let hauteur=720 and largeur=1280;;
13 open_graph "";;
14 resize_window largeur hauteur;*)
15
16
17 (*-----Types,variables et fonctions outils-----*)
18
19 type point    = {x: float; y: float; z: float};;
20 type vecteur  = {vx: float; vy: float; vz: float};;
21
22 let x0    = ref (float_of_int (size_x()/2))
23 and y0    = ref (float_of_int (size_y()/2))
24 and zoom  = ref 150.;;
25
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
26 let base = ref ({vx = 1.; vy = 0.; vz = 0.},
27 {vx = 0.; vy = 1.; vz = 0.},
28 {vx = 0.; vy = 0.; vz = 1.});;
29
30 let vecteur pt1 pt2 = {vx = (pt2.x -. pt1.x); vy = (pt2.y -. pt1.y); vz
    = (pt2.z -. pt1.z)};;
31
32 let produit_scalaire vct1 vct2 = vct1.vx *. vct2.vx +. vct1.vy *. vct2.
    vy +. vct1.vz *. vct2.vz;;
33
34 let norme vct = sqrt(vct.vx**2. +. vct.vy**2. +. vct.vz**2.);;
35
36 let unitaire vct = {vx = (vct.vx /. (norme vct));
37                                     vy = (vct.vy /. (norme
38                                     vct));
39                                     vz = (vct.vz /. (norme
40                                     vct))};;
41
42 let produit_vectoriel vct1 vct2 =
43     {vx = (vct1.vy *. vct2.vz -. vct1.vz *. vct2.vy);
44       vy = (vct1.vz *. vct2.vx -. vct1.vx *. vct2.vz);
45       vz = (vct1.vx *. vct2.vy -. vct1.vy *. vct2.vx)};;
46
47 let dans_base pt bse = let vctb1, vctb2, vctb3 = bse and origine = {x =
    0.; y = 0.; z = 0.} in
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

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

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
70 let projette pt = (int_of_float (!x0 +. !zoom *. pt.x), int_of_float (!
    y0 +. !zoom *. pt.y));;
71
72 let make_point (x,y,z) = { x = x; y=y; z=z};;
73
74 type element = int * int * float * float;; (*indice du noeud1, indice
    noeud2, module young, section*)
75
76 let make_element indice_noeud1 indice_noeud2 mod_young section = (
    indice_noeud1,indice_noeud2,mod_young,section);;
77
78 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)*)
79
80
81 (*COULEURS ET EPAISSEUR DES ITEMS*)
82
83 (*Fonction pour resize des intervalles (proportionnalité)*)
84 let map debut1 fin1 debut2 fin2 x =
85     let t = (x -. debut1)/.(fin1 -. debut1) in
86     (1. -. t)*. debut2 +. t*.fin2;;
87
88 (*Détermine la couleur d'un noeud à l'aide de la norme de la force
    appliquée en ce noeud*)
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
89 let couleurs_noeuds noeuds forces =
90     let normes_forces = Array.map norme forces in
91     (*let max_norme_force force1 force2 = max (norme force1) (norme
        force2) in
92     let min_norme_force force1 force2 = min (norme force1) (norme
        force2) in *)
93     let max_force = Array.fold_left max normes_forces.(0) normes_forces
94     and min_force = Array.fold_left min normes_forces.(0) normes_forces
95 in
96 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
97 tab_couleurs;; (*Censé faire un dégradé du bleu au rouge*)
98
99 (*Calcule l'emplacement des noeuds après application de la force,
        ajoutant les déplacements*)
100 let noeuds_deplaces noeuds déplacements =
101     Array.map2 (fun point vecteur ->
102         print_string "Coordonnées␣␣";print_float point.x;
            print_string "␣"; print_float point.y;print_string "
            ␣"; print_float point.z;
103         print_newline();
104         print_string "Déplacement␣␣";print_float vecteur.vx;
            print_string "␣"; print_float vecteur.vy;
            print_string "␣"; print_float vecteur.vz;
```


Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
105         print_newline();
106         print_newline();
107         make_point ((point.x +. vecteur.vx),(point.y +. vecteur.
            vy),(point.z +. vecteur.vz)) ) noeuds déplacements
            ;;
108
109 (*Calcule l'épaisseur à afficher des aretes. Attention, renvoie le max
    et le min des sections (unité d'origine)*)
110 let epaisseurs_elements elements =
111     let sections = Array.map (fun (i1,i2,young,section) -> section)
        elements in
112     let max_section = Array.fold_left (fun section accu_section -> max
        section accu_section) 0. sections
113     and min_section = Array.fold_left (fun section accu_section -> min
        section accu_section) infinity sections
114     and min_epaisseur = 2. (*Constantes d'épaisseurs des traits*)
115     in let max_epaisseur = (max_section/. min_section) *. min_epaisseur
116     in
117     let tab_epaisseurs = Array.map (fun section -> int_of_float (map
        min_section max_section min_epaisseur max_epaisseur section) )
        sections in
118     tab_epaisseurs,min_section,max_section;;
119
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
120 (*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.
121     Contient les aretes et noeuds avant et après application des
        forces*)
122 let make_items_affichables elements noeuds forces déplacements=
123     let epaisseurs,_,_ = epaisseurs_elements elements in
124     let couleurs = couleurs_noeuds noeuds forces in
125     let noeuds_depl = noeuds_deplaces noeuds déplacements in
126     let tab_aretes_originelles = Array.map2 (fun (i1,i2,young,
        section) epaisseur -> Arete(noeuds.(i1),noeuds.(i2),
        epaisseur,rgb 127 127 127) ) elements epaisseurs
127     and tab_noeuds_originels = Array.map (fun point -> Noeud(point
        ,3,rgb 127 127 127)) noeuds
128     and tab_aretes_deplacees = Array.map2 (fun (i1,i2,young,section)
        epaisseur -> Arete(noeuds_depl.(i1),noeuds_depl.(i2),
        epaisseur,black) ) elements epaisseurs
129     and tab_noeuds_deplaces = Array.map2 (fun point couleur -> Noeud
        (point,7,couleur)) noeuds_depl couleurs
130     in
131     Array.concat [tab_aretes_originelles;tab_noeuds_originels;
        tab_aretes_deplacees;tab_noeuds_deplaces];;
132
133 (*Fonction auxiliaire pour tracer une arete*)
134 let trace_arete point1 point2 epaisseur couleur =
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
135     let pt1 = dans_base point1 !base
136     and pt2 = dans_base point2 !base
137   in
138     let x1,y1 = projette pt1
139     and x2,y2 = projette pt2 in
140     set_color couleur;
141     set_line_width epaisseur;
142     moveto x1 y1;
143     lineto x2 y2;;
144
145   (*Fonction auxiliaire pour tracer un noeud*)
146   let trace_noeud point rayon couleur =
147     let epaisseur_trait = max 1 (int_of_float(float_of_int (rayon)
148       *. 0.2)) in
149     let pt = dans_base point !base in
150     let x,y = projette pt in
151     set_color couleur;
152     fill_circle x y rayon;
153     set_color black;
154
155     set_line_width epaisseur_trait;
156     draw_circle x y rayon;;
157
158   (*-----Algorithme du Peintre-----*)
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
159
160 (*Profondeur d'un point dans la direction z*)
161 let cote pt = let proj = dans_base pt !base in proj.z;;
162
163 (*Profondeur pour une arete*)
164 let cote_moyenne (point1, point2) = (cote point1 +. cote point2)/. 2.;;
165
166 (*Tri des items pour l'algo du peintre*)
167 let tri tab clef = let taille = (Array.length tab) - 1 in
168   for i = 1 to taille do
169     let j = ref i and check = clef tab.(i) and temp = tab.(i)
170       in
171     while !j > 0 && clef (tab.(!j-1)) > check do
172       tab.(!j) <- tab.(!j-1);
173       j := !j - 1;
174     done;
175     tab.(!j) <- temp;
176   done
177 ;;
178 let tri_items items_a_afficher =
179   let clef_tri item = match item with
180     | Arete(point1,point2,epaisseur,couleur_arete) ->
181       cote_moyenne (point1,point2)
182     | Noeud(p,rayon,couleur_noeud) -> cote p
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
182         in
183         tri items_a_afficher clef_tri;;
184
185 (*Affichage des items dans le bon ordre*)
186 let peindre_items items_a_afficher = let taille = Array.length
    items_a_afficher in
187     tri_items items_a_afficher;
188     for i = taille-1 downto 0 do
189         (*print_string "Traçage de l'item n° : ";print_int i;
190          print_newline();*)
191         let item= items_a_afficher.(i) in
192         match item with
193         | Noeud(point,rayon,couleur) -> trace_noeud point rayon
194           couleur
195         | Arete(point1,point2,epaisseur,couleur) -> trace_arete
196           point1 point2 epaisseur couleur
197
198     done;;
199
200 (*-----Récupération des données dans un fichier extérieur-----*)
201 (*Type tableau dynamique pour faciliter la récupération des données*)
202 type 'a tableau_dynamique = {mutable support: 'a array;
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

mut

201

202

```
203 let make_td element = {
204     support = Array.make 16 element; taille = 0};;
205
206 let ajoute td valeur =
207     if td.taille <> Array.length td.support then
208         begin
209             td.support.(td.taille) <- valeur;
210             td.taille <- td.taille + 1;
211         end
212     else
213         begin
214             let new_support = Array.make (td.taille*2) valeur in
215             for i = 0 to (td.taille-1) do
216                 new_support.(i) <- td.support.(i);
217             done;
218             td.support <- new_support;
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
219         td.taille <- td.taille + 1;
220     end;;
221
222 (*Fonction qui lit le fichier contenant les données et qui renvoie les
    tableaux contenant :
223     -les noeuds (indités par i)
224     -le déplacement des noeuds (deplacement du noeud i à l'indice i
    )
225     -les forces appliquées au noeud i
226     -les elements, ie (indice_noeud1,indice_noeud2,module_young,
    section)*)
227 let lecture_fichier nomFichier =
228     let fichier = open_in nomFichier in
229     let point_generique = make_point (0.,0.,0.) in
230     let element_generique = make_element 0 0 0. 0. in
231     let deplacement_generique = vecteur point_generique
    point_generique in
232     let force_generique = vecteur point_generique point_generique in
233     let noeuds = make_td point_generique in
234     let deplacements = make_td deplacement_generique in
235     let forces = make_td force_generique in
236     let elements = make_td element_generique in
237
238     let ligne = ref (input_line fichier) in
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
239     let nb_noeuds, nb_elements = Scanf.sscanf !ligne "%d;%d" (fun n1
240         n2 -> (n1,n2)) in
241
242     for i = 0 to nb_noeuds-1 do
243         ligne:= input_line fichier;
244         let x,dx,fx = Scanf.sscanf !ligne "%f;%f;%f" (fun x dx
245             fx-> (x,dx,fx)) in
246         ligne:= input_line fichier;
247         let y,dy,fy = Scanf.sscanf !ligne "%f;%f;%f" (fun y dy
248             fy -> (y,dy,fy)) in
249         ligne:= input_line fichier;
250         let z,dz,fz = Scanf.sscanf !ligne "%f;%f;%f" (fun z dz
251             fz-> (z,dz,fz)) in
252
253         ajoute noeuds (make_point (x,y,z));
254         ajoute déplacements (vecteur point_generique (
255             make_point (dx,dy,dz)));
256         ajoute forces (vecteur point_generique (
257             make_point (fx,fy,fz)));
258     done;
259     for i = 0 to nb_elements-1 do
260         ligne:=input_line fichier;
261         let element = Scanf.sscanf !ligne "%d;%d;%f;%f" (fun i1
262             i2 module_young section -> (i1,i2,module_young,
263             section)) in
```


Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
256             ajoute elements element;
257         done;
258         close_in fichier;
259         let coupe_tableau_dyn tab = Array.sub (tab.support) 0 (tab.
            taille) in
260         coupe_tableau_dyn noeuds,coupe_tableau_dyn déplacements,
            coupe_tableau_dyn forces ,coupe_tableau_dyn elements;;
261
262 (*Boucle pour afficher la structure et la faire tourner à l'aide du
            clavier*)
263 let en_sync_items items_a_afficher =
264     auto_synchronize false;
265     display_mode false;
266     peindre_items items_a_afficher;
267
268     while true do
269         let event = wait_next_event [Key_pressed] in let
            key = event.key in
270         if key = 's' then y0 := !y0 -. 5.;
271         if key = 'z' then y0 := !y0 +. 5.;
272         if key = 'q' then x0 := !x0 -. 5.;
273         if key = 'd' then x0 := !x0 +. 5.;
274         if key = 'o' then rotation_base_y (0.05);
275         if key = 'l' then rotation_base_y (-0.05);
276         if key = 'k' then rotation_base_x (-0.05);
```

Annexes :

Code Ocaml : Affichage_FEM_3D.ml

```
277         if key = 'm' then rotation_base_x (0.05);
278         if key = 'a' then zoom := !zoom +. 5.;
279         if key = 'e' then zoom := !zoom -. 5.;
280         clear_graph ();
281         peintre_items items_a_afficher;
282         synchronize ();
283     done;;
284
285 (*Fonction main : récupere les tableaux et lance la fonction
    en_sync_items.*)
286 let main () =
287 let noeuds,deplacements,forces,elements = lecture_fichier "resultat.txt"
    in
288 print_string "Nombre d'éléments : ";
289 print_int (Array.length elements); print_newline();
290
291 (*let noeuds2 = noeuds_deplaces noeuds deplacements in
292 let tous_noeuds = Array.append noeuds noeuds2 in
293 let tous_elements = Array.append elements elements in
294 *)
295 let items_a_afficher = make_items_affichables elements noeuds forces
    deplacements in
296 (*affiche_aretes_elements elements noeuds;*)
297 (*set_line_width 10;
298 lineto (size_x()/2) (size_y()/2); *)
```

Annexes :

Code Ocaml : *Affichage_FEM_3D.ml*

```
299 synchronize ();  
300 en_sync_items items_a_afficher;;  
301  
302 main();;
```