

**ENSTA-Bretagne**

---

Méthodes Ensemblistes pour la Robotique  
Projet commun: Création d'un logiciel

---

**AUTOMATIQUE ET SYSTEMES DE NAVIGATION**

**IASE & ESSE 2011**



# Sommaire

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Contexte de l'étude et partage du travail</b>	<b>7</b>
<b>3</b>	<b>Partage des tâches</b>	<b>8</b>
<b>4</b>	<b>Scanner &amp; Parser ESTIM</b>	<b>9</b>
4.1	Grammaire ESTIM (BNF) . . . . .	10
4.2	Analyseur Lexical & Syntaxique . . . . .	10
4.2.1	Bison . . . . .	10
4.2.2	Flex . . . . .	11
4.2.3	Gestion des erreurs . . . . .	11
4.2.4	Tests Unitaires . . . . .	13
4.3	Analyseur sémantique . . . . .	16
4.4	Résumé parseur . . . . .	16
4.5	Amélioration . . . . .	16
4.6	Ajout de la règle . . . . .	16
4.6.1	Références . . . . .	17
<b>5</b>	<b>Interfaces</b>	<b>18</b>
5.1	Choix des couleurs : Noir et blanc ou couleurs . . . . .	19
5.1.1	Modification du window.ccp . . . . .	19
5.2	Menu bar et status bar . . . . .	19
5.2.1	Modification de l'héritage des fichiers . . . . .	20
5.2.2	Ajout des menus dans la menu bar . . . . .	20
5.2.3	Connexion des sous-menus open, save et save as . . . . .	21
5.2.4	Status bar . . . . .	21
5.3	Enlever le bouton exit . . . . .	21
5.4	code . . . . .	21
5.4.1	Amélioration . . . . .	22
<b>6</b>	<b>Création d'une interface permettant la visualisation du résultat</b>	<b>23</b>
6.1	Prise en main de l'interface graphique . . . . .	23
6.2	Le drag and zoom . . . . .	23
6.2.1	résumé Zoom et implémentation . . . . .	27

6.3	Dezoom . . . . .	28
6.4	Amélioration du zoom . . . . .	28
<b>7</b>	<b>Utilisation d'Ibex</b>	<b>29</b>
7.1	Rappel de théorie et définitions . . . . .	29
7.1.1	Intervalle . . . . .	29
7.1.2	Contracteur . . . . .	29
7.1.3	Relaxation . . . . .	31
7.1.4	SIVIA . . . . .	32
7.2	Présentation de l'implémentation de l'analyse par intervalles . . . . .	33
7.2.1	IBEX et contracteurs . . . . .	33
7.2.2	Sivia.cpp . . . . .	34
7.3	Difficultés rencontrés et solutions apportées . . . . .	39
7.3.1	Affichage des pavés . . . . .	39
7.3.2	Utilisation de IBEX/QInter . . . . .	40
7.3.3	Utilisation des objets IBEX et mémoire . . . . .	40
<b>8</b>	<b>Méthode de Monte Carlo</b>	<b>41</b>
8.1	Intérêt et Principe . . . . .	41
8.2	Implémentation . . . . .	41
8.2.1	classe montecarlo . . . . .	41
8.2.2	window . . . . .	42
8.2.3	syntaxtree . . . . .	42
8.3	Exemple de résultat . . . . .	42
<b>9</b>	<b>Modélisation</b>	<b>43</b>
9.1	Importation de Qt à Rhapsody . . . . .	43
9.1.1	Problèmes rencontrés . . . . .	43
9.2	Utilisation du plugin Qt sous Eclipse . . . . .	44
9.3	Génération de code C++ sous Rhapsody . . . . .	45
9.3.1	MinGW et MSYS . . . . .	45
9.4	Perspectives . . . . .	46
9.4.1	Modèle global . . . . .	46
<b>10</b>	<b>Intégration</b>	<b>49</b>
10.1	Intégration du code . . . . .	49

10.1.1 Qt . . . . .	49
10.1.2 Travail d'équipe . . . . .	49
10.2 Intégration du rapport . . . . .	50
10.3 Gestion du projet . . . . .	50
<b>A Autres travaux</b>	<b>52</b>
A.1 Implémentation de la fonction d'ouverture de fichiers . . . . .	52
A.2 Plage d'affichage des segments . . . . .	52

# 1 Introduction

1

Ce projet collectif a consisté en la réalisation d'un programme d'estimation de solutions. Du fait de la complexité d'une telle tâche, nous nous sommes donc séparés en équipes de travail.

Afin de travailler en autonomie, les objectifs de chaque équipes étaient relativement décorrelés. Pour coordonner le tout, et pour faciliter l'intégration, nous avons mis en place un système de Projet GOOGLE permettant à chaque équipe de poster ses fichiers.

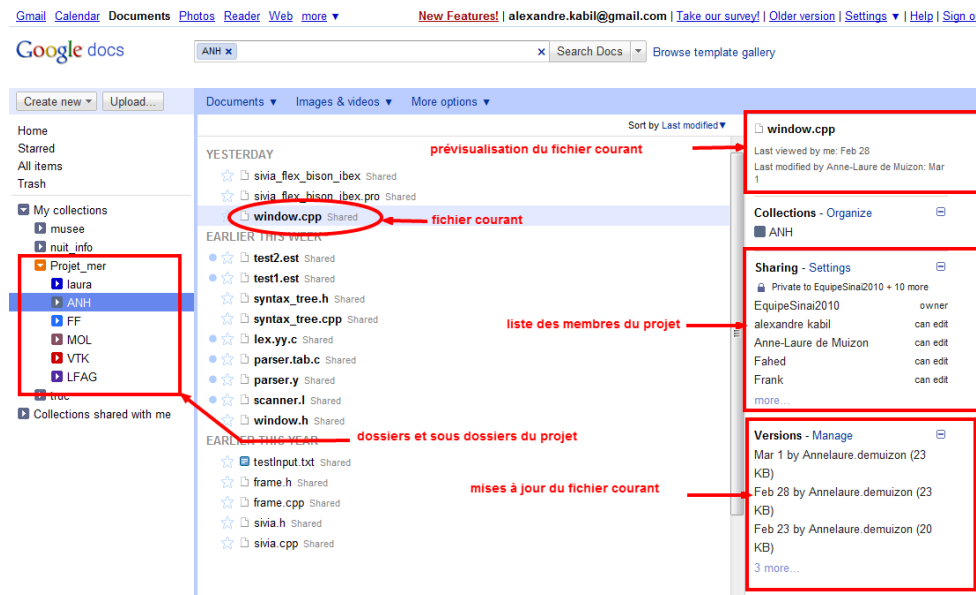


Figure 1 – *vue du projet sous Google documents*

Ce système nous a permis de garder une trace des mises à jour des divers fichiers et surtout de ne pas les mélanger. La communication intra et inter équipes fut l'une des tâches les plus difficiles à coordonner. Nous reviendrons sur l'aspect 'gestion' dans la partie 'Intégration'.

L'essentiel du projet s'est effectué sous Qt, une Interface de Développement multi-langages.

<sup>1</sup>Equipe UML

## 2 Contexte de l'étude et partage du travail

<sup>2</sup> Le partage du travail initial fut le suivant:

- **Création d'un arbre abstrait de données permettant l'évaluation de formules et conditions d'entrées**

Il s'agit du point d'entrée du programme, à savoir la lecture d'une équation et des différents paramètres que l'on souhaite estimer.

- **Création d'une interface permettant la visualisation du résultat**

Cette partie concerne toute la partie visible du programme, à savoir le zoom, l'application de différentes méthodes et la visualisation du résultat.

- **Utilisation d' IBEX et de la relaxation d'intervalles**

Certaines solutions ne sont pas facilement atteignables, et une équipe a donc du travailler sur la contraction d'intervalles.

- **Programmation de la méthode de Monte-carlo**

Afin d'apporter plus de flexibilité, une seconde méthode d'estimation a été implémentée.

- **Modélisation du programme sous UML et intégration**

Afin de structurer le programme et les différents travaux, la modélisation exécutable fut souhaitée. L'intégration est une tâche à part entière lors d'un projet collectif, et elle est l'aboutissement du projet.

- **Tests et mise en situation du programme**

Avant de proposer le programme, celui-ci doit subir des tests de *Benchmark*, à savoir des mises en situation .

---

<sup>2</sup>Equipe UML

### 3 Partage des tâches

<sup>3</sup> Au cours du projet, nous nous sommes rendus compte d'un déséquilibre entre les différentes tâches à accomplir. De fait, le rôle de la plupart des groupes a changé au cours du temps, et l'on pourrait établir le partage des tâches final:

- **Groupe Bison-Flex:** Création de l'arbre syntaxique, résolution des erreurs de parsing et participation à l'intégration.
- **Groupe interface graphique:** Réalisation de l'interface globale, intégration de différentes fonctions et de différentes parties du code.
- **Groupe fonctions d'interfaces graphiques:** Implémentation de fonctions visuelles tels le zoom, amélioration des fonctions existantes.
- **Groupe IBEX:** Implémentation d' IBEX, et amélioration de l'efficacité de l'algorithme Sivia.
- **Groupe Monte-Carlo:** Implémentation de l'algorithme de Monte Carlo et intégration du code.
- **Groupe UML:** Création et tests sur le diagramme de classes, intégration, 'gestion' du projet.

Bien entendu, à la fin du projet, tout le monde a travaillé sur différentes parties afin de combler les retards imprévus.

---

<sup>3</sup>Equipe UML



## 4 Scanner & Parser ESTIM

<sup>4</sup> Le logiciel ESTIM prend en entrée des problèmes d'estimation décrits dans un fichier texte ou dans l'éditeur du logiciel. La structure choisie pour exprimer ces problèmes est la suivante :

**Model** :  $2 * \cos(x1) * \sin(x2 + x1) + 1;$

**Relaxed data** : 2 ;

**Data** : (1, [2, 3])(4, [5, 6])(7, [8, 9]);

Où l'expression après " Model " donne la contrainte du problème ; " Relaxed data " le nombre de relaxations (noté q); " Data " les couples de valeurs intervalles. Notre travail consista à formaliser à l'aide d'une grammaire le langage que nous nommerons " ESTIM ", puis à réaliser un parseur qui permette d'analyser lexicalement et syntaxiquement le texte en entrée, enfin à effectuer une analyse sémantique permettant au code applicatif de manipuler les données issues de l'arbre syntaxique.

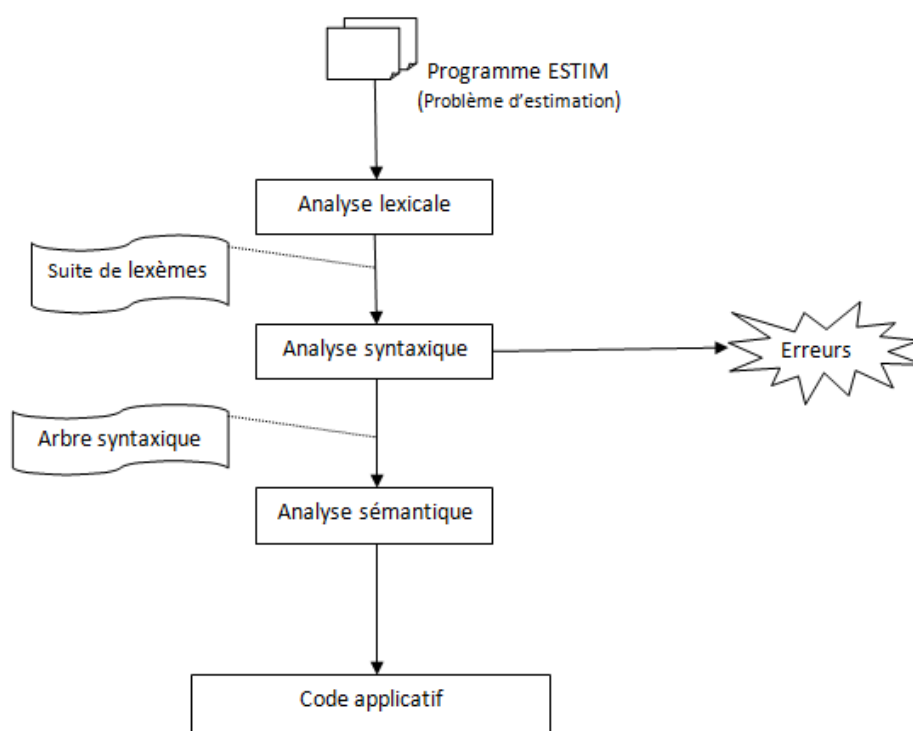


Figure 2 – Structure du parseur ESTIM

Pour réaliser ce parseur, nous avons choisi d'utiliser du code C/C++ pour l'analyseur sémantique. Pour les analyseurs lexicaux et syntaxiques, nous utilisons Flex et Bison : des outils largement répandus par le projet GNU

---

<sup>4</sup>Equipe Bison

## 4.1 Grammaire ESTIM (BNF)

Pour réaliser le parseur associé au langage ESTIM, nous avons tout d'abord défini la grammaire du langage. La BNF de ESTIM est la suivante :

```

<Input>          ::= < txt>

<txt>            ::= Model ':' < expr> ';' <constraint>

<constraint>     ::= Relaxed data ':' <int> ';' Data ':' <listdata> ';'

<listdata>       ::= <listdata> <data>
                    | <data>

<data >          ::= '(' <dataVal> ',' <dataVal> ')'

<dataVal>        ::= REAL
                    | INTEGER
                    | '[' <dataVal> ',' <dataVal> ']'

<expr>           ::= VARIABLE
                    | REAL
                    | INTEGER
                    | <expr> + <expr>
                    | <expr> '+' <expr>
                    | <expr> '*' <expr>
                    | <expr> '/' <expr>
                    | tk_function '(' expr ') '
                    | '(' < expr> ') '

Int              ::= INTEGER

DIGIT            ::= [0-9]
INTEGER          ::= { DIGIT }+
REAL             ::= {INTEGER} ( '.' {INTEGER} )
LETTER           ::= [A-Za-z]
VARIABLE         ::= {LETTER} ( {LETTER} | {DIGIT} ) *

```

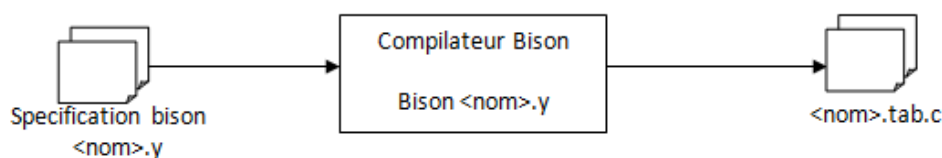
## 4.2 Analyseur Lexical & Syntaxique

Nous avons utilisé Bison pour réaliser la partie analyseur syntaxique, et Flex pour la partie analyseur lexical.

### 4.2.1 Bison

Bison est un outil bâti pour la construction des analyseurs syntaxiques à partir de leur grammaire. Il accepte en entrée la description du langage décrit sous forme de règles de productions et

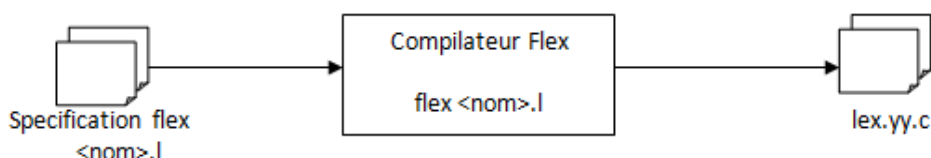
produit un programme écrit en langage C qui une fois compilé, reconnaît les phrases du langage en question.



Après avoir décrit la grammaire du langage ESTIM conformément aux règles Bison, nous avons obtenu le fichier suivant : **Commande de compilation dans un terminal** : `> bison nom_fichier.y` fichier `parser.y` contient le programme Bison du langage ESTIM. `Parser.y` utilise les fonctions du fichier `syntaxe_tree.c` pour créer l'arbre syntaxique sous forme d'une chaîne de nJud.

## 4.2.2 Flex

L'outil Flex accepte en entrée des spécifications d'unités lexicales sous forme de définition régulière et produit un programme en langage C.



**Commande de compilation dans un terminal** : `> flex nom_fichier.l` L'exécutable obtenu à partir du fichier `lex.yy.c` permet de lire le texte d'entrée caractère par caractère jusqu'à ce qu'il trouve le plus long préfixe du texte d'entrée qui corresponde à l'une des expressions régulières. Le fichier `scanner.l` contient le programme Flex du langage ESTIM.

## 4.2.3 Gestion des erreurs

Il s'agit ici de détecter et d'informer l'utilisateur le plus précisément possible sur les erreurs de syntaxe rencontrées lors du parsing. La gestion des erreurs se fait par la fonction `yerror` dans le fichier `parser.y`.

```

int yerror(char *s) {
    sprintf(msgErrParser,"Error: %s near %s    -> ligne : %d \n", s , yytext, no_line);
    printf(msgErrParser);
    return(0);
}
  
```

Lorsque le parseur rencontre une erreur de syntaxe, la fonction `yerror` est exécutée. Cette fonction renvoie 0 lorsque le parsing se passe bien, et renvoie 1 sinon. En cas d'erreur elle écrit dans la variable globale `msgErrParser` la ligne (`no_line`) et le mot (`yytext`) incriminés. `yytext` est une variable prédéfinie dans Flex. C'est un tableau de `Char` qui contient la chaîne d'entrée qui a été acceptée. `no_line` une variable entière que nous avons définis pour dans le fichier `scanner.l`. Cette variable nous permet de calculer le numéro de la ligne courante.

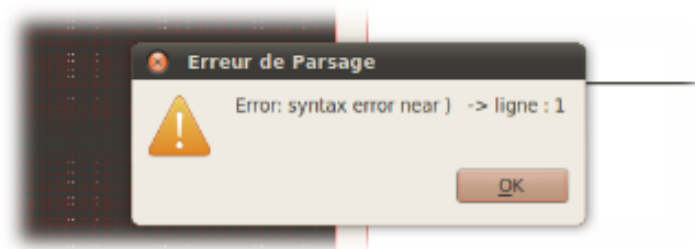
```

15 %%
16 {INTEGER}          { yy1val.name=QString(yytext); return(tk_integer);}
17 {REAL}             { yy1val.name=QString(yytext); return(tk_real);}
18 [-+*/(),\[::]     { yy1val.name=QString(yytext); return *yytext;}
19 "sin"              { yy1val.name="sin" ; return(tk_function);}
20 "cos"              { yy1val.name="cos" ; return(tk_function);}
21 "exp"              { yy1val.name="exp" ; return(tk_function);}
22 "Model"|"model"|"MODEL"      { yy1val.name="model" ; return(tk_model);}
23 "Relaxed data"|"Relaxed Data"|"relaxed data"|"RELAXED DATA" { yy1val.name="constraint" ; return(t
24 "Data"|"data"|"DATA"      { yy1val.name="data" ; return(tk_data);}
25 {VARIABLE} {yy1val.name=QString(yytext); return(tk_variable);}
26
27 [ \t]+             { /* skipping spaces */ }
28 "\n"               { ++no_line; /* counting CR */ }
29
30 %%

```

Figure 3 – *Mise à jour de scanner.l*

Comme le montre la figure ci-dessus, la variable `no_line` est mise à jour chaque fois qu'on rencontre le symbole du saut de ligne (`\n`). La variable globale `msgErrParser` est utilisée par l'interface graphique pour afficher le message d'erreur dans une message box.

Figure 4 – *Message Box de retour d'erreur*

Pour éviter des retours d'erreur intempestifs, nous avons surchargé les mots réservés du langage en gérant la casse (majuscule) dans le fichier `scanner.l`.

```

22 "Model"|"model"|"MODEL"      { yy1val.name="model" ; return
23 "Relaxed data"|"Relaxed Data"|"relaxed data"|"RELAXED DATA"
24 "Data"|"data"|"DATA"      { yy1val.name="data" ; return(tk_d
25 {VARIABLE} {yy1val.name=QString(yytext); return(tk_variable);}
26

```

Figure 5 – *gestion de la casse*

Le parseur acceptera le terme " Model " ou " model " ou encore " MODEL ".

#### 4.2.4 Tests Unitaires

Pour vérifier le bon fonctionnement du parseur, nous avons généré dans un fichier texte (info.txt) l'arbre syntaxique du problème test suivant :

**Model** :  $2 * \cos(x1)$ ;

**Relaxed data** : 2 ;

**Data** :  $(1, [2, 3])(4, [5, 6])([7, 8], [9, 10])$ ;

On obtient le contenu du fichier texte suivant :

```

Start computing
  Table corresponding to the abstract syntax tree
0,   name: 2   left:-1   right:-1
1,   name: x1  left:-1   right:-1
2,   name: cos left:1    right:-1
3,   name: *   left:0    right:2
4,   name: 2   left:-1   right:-1
5,   name: 1   left:-1   right:-1
6,   name: 2   left:-1   right:-1
7,   name: 3   left:-1   right:-1
8,   name: interval left:6   right:7
9,   name: dataval left:5    right:8
10,  name: 4   left:-1   right:-1
11,  name: 5   left:-1   right:-1
12,  name: 6   left:-1   right:-1
13,  name: interval left:11  right:12
14,  name: dataval left:10   right:13
15,  name: listdata left:9   right:14
16,  name: 7   left:-1   right:-1
17,  name: 8   left:-1   right:-1
18,  name: interval left:16  right:17
19,  name: 9   left:-1   right:-1
20,  name: 10  left:-1   right:-1
21,  name: interval left:19  right:20
22,  name: dataval left:18   right:21
23,  name: listdata left:15  right:22
24,  name: constraint left:4   right:23
25,  name: model left:3     right:24
-----

```

Figure 6 – Tests : arbre syntaxique généré (info.txt)

sous forme d'arbre:

Dans le cas particulier où l'utilisateur ne met qu'une seule Data, listdata n'existe pas et on a directement un dataval pour référencer l'unique information transmise.

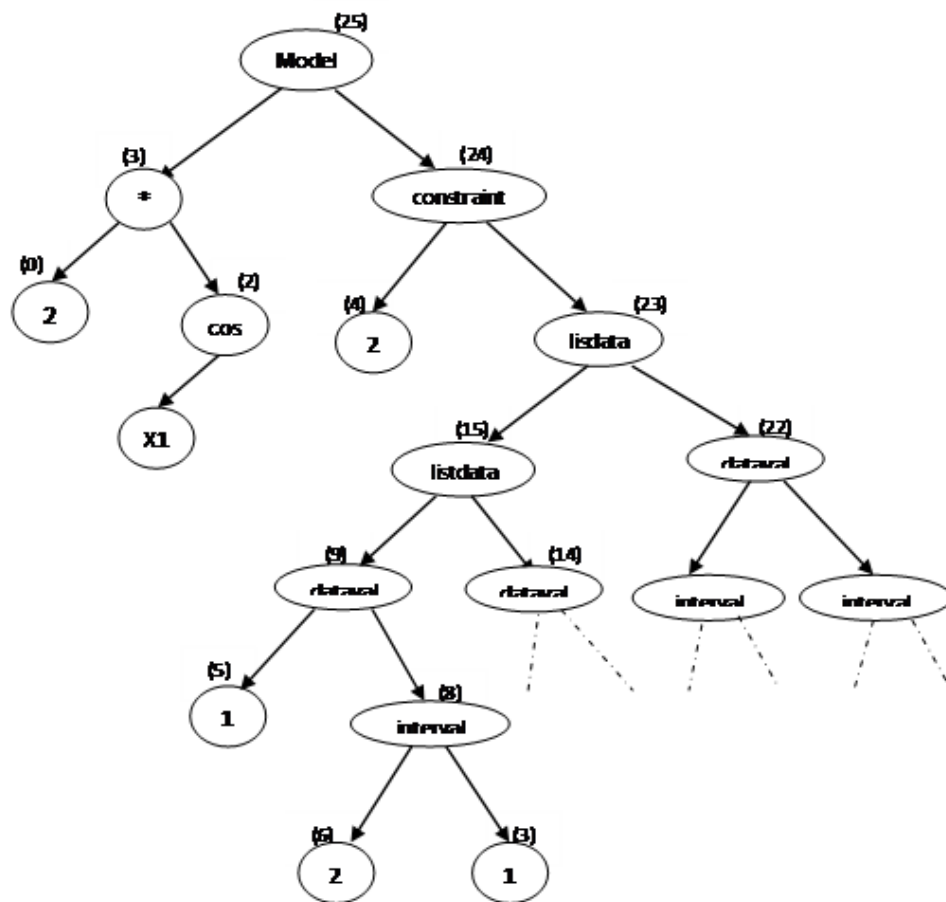


Figure 7 – arbre syntaxique

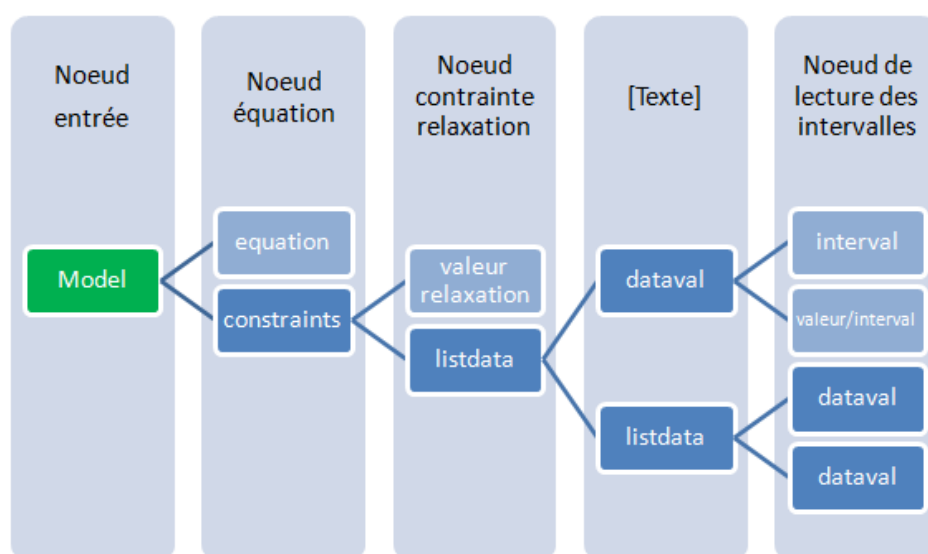


Table corresponding to the abstract syntax tree

0,	name: 2	left:-1	right:-1
1,	name: x1	left:-1	right:-1
2,	name: cos	left:1	right:-1
3,	name: *	left:0	right:2
4,	name: x2	left:-1	right:-1
5,	name: x1	left:-1	right:-1
6,	name: +	left:4	right:5
7,	name: sin	left:6	right:-1
8,	name: *	left:3	right:7
9,	name: 1	left:-1	right:-1
10,	name: +	left:8	right:9
11,	name: 8	left:-1	right:-1
12,	name: 7	left:-1	right:-1
13,	name: 8	left:-1	right:-1
14,	name: interval	left:12	right:13
15,	name: 9	left:-1	right:-1
16,	name: 10	left:-1	right:-1
17,	name: interval	left:15	right:16
18,	name: dataval	left:14	right:17
19,	name: constraint	left:11	right:18
20,	name: model	left:10	right:19

### 4.3 Analyseur sémantique

Les exigences clients spécifient que l'on doit entrer une chaîne de caractères 'normalisée' et que le parseur doit séparer les différents types d'informations. Comme vu précédemment, certains mots spécifient un type d'information :

**Model** pour l'équation

**Relaxed Data** pour le coefficient de relaxation

**Data** pour la Matrice d'intervalles

Des méthodes ont donc été créées pour accéder à chacune de ces données. Les variables associées sont mises en extern pour faciliter le partage d'information. Il faut donc garantir que les méthodes ont été appelées avant de se servir d'une variable. `INTERVAL Eval(int i, INTERVAL_VECTOR X)` pour évaluer l'équation `void Get_q(int i)` pour mettre à jour le coefficient de relaxation `void Get_data(int i, int index, bool parse)` pour mettre à jour la matrice d'intervalles.

### 4.4 Résumé parseur

5

Un message d'erreur apparaît si le parseur détecte une erreur. Techniquement, la fonction `MsgErreur` affiche la `QMessageBox` si la fonction `create_syntax_tree`, généré automatique par Bison, ne renvoie pas 0. C'est aussi le fait que cette fonction soit générée qui impose que le message à afficher soit stocké dans une variable globale (`msgErrParser`). `MsgErreur` est donc appelée à deux endroits:

- directement dans la fonction `load` pour signaler les erreurs à l'utilisateur quand il ouvre un fichier.
- dans `compute` de `Window` si jamais il clique sur `GO`.

D'une manière générale cette fonction doit être systématiquement appelé après `create_syntax_tree`.

### 4.5 Amélioration

Actuellement, même si il y a des erreurs lors du parsing, un arbre est créé, le message erreur affiché mais le calcul est tout de même effectué. Peut être serait-il préférable de ne pas continuer la procédure (affichage) et imposer ainsi à l'utilisateur qu'il n'ait plus d'erreur dans le code au lieu d'afficher un arbre potentiellement faux ?

### 4.6 Ajout de la règle

La règle devrait permettre de fournir une aide à l'utilisateur dans son repérage dans la fenêtre. Pour l'ajout de la règle, il a fallu procéder à la création d'un bloc appelé `Pavage` englobant le `frame pavage1` ainsi que deux `QLabel` contenant les règles. Les règles sont donc des `Strings`.

---

<sup>5</sup>Guillaume



#### 4.6.1 Références

LIENS VERS LA DOCUMENTATION OFFICIELLE : [http://www.esiee.fr / couppriem/thlc.htm](http://www.esiee.fr/couppriem/thlc.htm)

## 5 Interfaces

<sup>6</sup> Lors du projet ASN, le travail était réparti en différentes activités. Nous avons choisi de nous consacrer à l'interface graphique du projet. Le projet avait déjà initialement une interface très simplifiée qui permettait d'observer la figure de modifier la fonction d'observation et relancer le programme. Différentes fonctions ont donc été ajoutées :

- Le choix des couleurs sur la visualisation (Noir/Blanc ou couleurs).
- L'ajout d'une status bar et d'un menu pour sauvegarder ou ouvrir des fichiers (\*.est) contenant une fonction et ses intervalles.
- La fonction de Monte-Carlo.
- Le zoom.

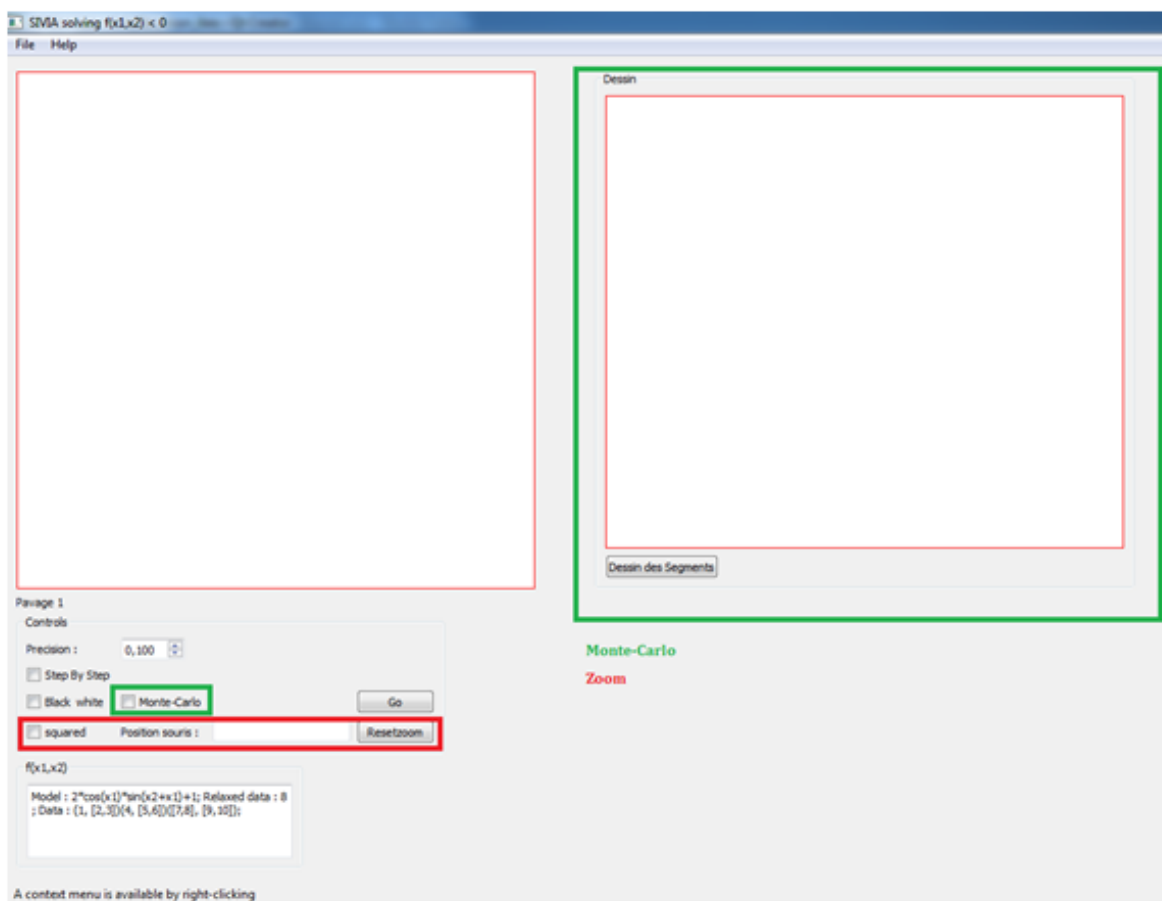


Figure 8 – *vue de l'interface graphique*

---

<sup>6</sup>Equipe Interfaces

## 5.1 Choix des couleurs : Noir et blanc ou couleurs

L'objectif de cette fonction est de pouvoir modifier la coloration de l'image : en niveau de gris (3 niveaux) ou en couleur. Cette action sera régie par ce qu'on appelle une " QCheckBox ", autrement dit un élément graphique booléen qui est soit coché (" true "), soit décoché (" false "). Si cette fonction est décochée, alors la coloration sera en couleur. Comment ajouter la fonction noir et blanc ? Il y a deux étapes. La première étape consiste à modifier l'interface graphique pour créer la " QCheckBox " et la deuxième consiste à modifier les fonctions qui s'exécutent lorsqu'on active cette nouvelle boîte.

### 5.1.1 Modification du window.cpp

1. Il s'agit dans un premier temps de créer la " QCheckBox " dans le constructeur du fichier `window.cpp` :

```
BlackWhite = new QCheckBox("Black & white");
```

2. Nous l'ajoutons dans l'organisation de la fenêtre en ajoutant toujours dans ce constructeur une ligne qui le place dans la fenêtre :

```
controlsLayout->addWidget(BlackWhite,2, 0, Qt::AlignLeft);
```

3. La connexion entre l'interface graphique est déjà indirectement réalisée par la fonction " `createConnexion()` ", en revanche il nous reste à modifier l'action qui aura lieu quand on sélectionnera le bouton " GO ". Pour cela nous modifions donc la fonction " `compute()` " qui est appelée par l'action d'exécution de " GO ".

Dans cette fonction, la " QCheckBox " va créer un booléen " BW " qui permettra de créer soit un tableau " `Color[3]` " en noir et blanc ou en couleur :

```
if(!BW){
    color[0]=QColor("Yellow");
    color[1]=QColor("Red");
    color[2]=QColor("Cyan");
}
else{
    color[0]=QColor("Grey");
    color[1]=QColor("Black");
    color[2]=QColor("White");
}
```

Ainsi la fonction `sivia.cpp` qui gère l'affichage de la fonction recevra un tableau contenant la gamme de couleur à utiliser.

## 5.2 Menu bar et status bar

Pour ajouter une menu bar et une status bar respectivement en haut et en bas de l'interface graphique, nous avons réalisé des modifications sur les fichiers " `window` ". Par la suite, chaque

menu sera associé à sa fonction correspondant à son application. Par exemple, pour le menu " save as ", il faudra réaliser toute la partie de sauvegarde qui s'appliquera lors du clic sur " save as " dans le menu. Ici, nous allons seulement expliquer les actions réalisées pour la création graphique du menu.

### 5.2.1 Modification de l'héritage des fichiers

Nos fichiers " window " héritent de " Qwidget " ce qui permet d'afficher notre fenêtre graphique. Mais ce dernier ne prend pas en compte les fonctionnalités d'une barre de menu ainsi que d'une status bar. Il faut donc modifier cet héritage pour permettre d'ajouter les fonctions voulues. La solution est donc d'hériter de " QMainWindow " qui hérite également de " Qwidget ". Par ailleurs, " QMainWindow " contient également toutes les fonctionnalités pour les barres de menu et la status bar.

```
#include <QMainWindow>
```

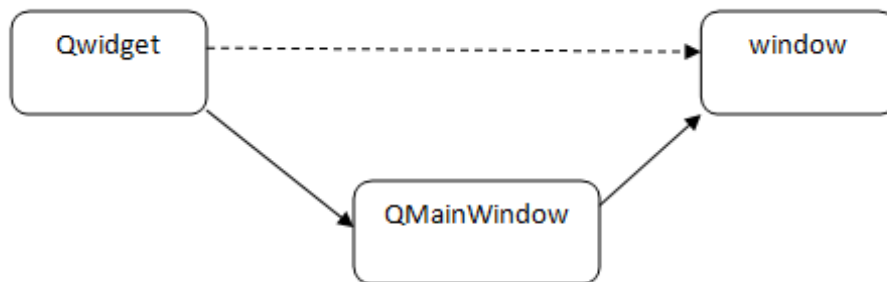


Figure 9 – schéma de l'héritage

En modifiant cet héritage, nous obtenons donc une fenêtre qui par défaut possède déjà une menu bar et une status bar.

### 5.2.2 Ajout des menus dans la menu bar

Afin d'ajouter les menus, nous avons donc créé une nouvelle méthode " createMenus() " dans laquelle les deux menus (" File " et " Help ") ont été intégrés. Des actions correspondantes aux sous-menus ont par la suite été ajoutées à ces menus comme présenté ci-dessous :

```

void Window::createMenus()
{ Menu File
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(openAct);
    fileMenu->addAction(saveAct);
    fileMenu->addAction(saveAsAct);
    fileMenu->addSeparator();
    fileMenu->addAction(exitAct);

// Menu Help

```

```

    helpMenu = menuBar()->addMenu(tr("&Help"));
    helpMenu->addAction(aboutAct);
    helpMenu->addAction(aboutQtAct);
}

```

### 5.2.3 Connexion des sous-menus open, save et save as

Enfin, ces actions ont été connectées à des slots ou méthodes que nous avons développées et ajoutées dans les fichiers " window ". Plus exactement, 3 sous-menus : open, save et save as. Le sous-menu " open " ouvre une fenêtre de dialogue où l'on sélectionne un fichier \*.est. Puis, la méthode " loadfile " est appelée avec en paramètre ce fichier. Cette méthode a pour fonction de charger le fichier et d'afficher dans l'interface graphique ce qu'elle a lu. Puis, le nom du fichier est enregistré dans une variable nommée " currentfile ". Enfin, l'utilisateur devra appuyer sur le bouton GO pour lancer l'application. Le nom du fichier qui vient d'être ouvert sera affiché dans le titre de la fenêtre de l'interface graphique. Le sous-menu " save " fait le contraire du sous-menu " open ". Il écrit dans le fichier de " currentfile ", ce que l'utilisateur a écrit dans l'interface graphique. Si la variable " currentfile " est vide alors la méthode " saveas " est appelée. Le sous-menu " saveas " effectue la même action que le sous-menu " save " à l'exception qu'une fenêtre de sélection de fichier est ouverte en premier. L'utilisateur sélectionne ou rentre un nouveau nom de fichier. Ce nom est enregistré dans la variable " currentfile " et le fichier est écrit. Le nouveau fichier apparaît ensuite dans le titre de la fenêtre de l'interface graphique.

### 5.2.4 Status bar

La status bar fait partie des méthodes par défaut de QMainWindow. Nous l'avons donc initialisée dans le constructeur. Puis les valeurs qu'elle doit présenter ont été mises lors de la définition des actions des sous-menus.

## 5.3 Enlever le bouton exit

Cette opération est relativement simple. En effet, il s'agit tout simplement de retirer les lignes de code qui font apparaître ce bouton ainsi que la connexion à son action. Ces lignes de codes sont présentes dans le fichier " window.cpp " ; Il s'agit aussi de retirer la ligne de code qui instancie ce bouton dans le fichier " window.h " associé. Ce bouton a été remplacé par le menu " Exit ".

## 5.4 code

<sup>7</sup> Il se place naturellement dans Window.h et Window.cpp.

**Window.h:** Les QLabel VertScale et HorizScale sont déclaré, ainsi que la GridLayout PavageLayout et la QGroupBox pavageGroup. La méthode createPavage est aussi déclarée.

**Window.cpp:** Pavagegroup ainsi que ses éléments sont instanciés à l'instanciation de la Window grâce à la méthode createPavage, qui fixe aussi la mise en page.

**Problème :** Il n'y a malheureusement pas de QWidget dédié aux règles. La String pourrait être remplacée par un fichier graphique, mais le résultat est encore plus incertain.

---

<sup>7</sup>guillaume

### 5.4.1 Amélioration

L'échelle pourrait être affichée en plus de la règle, mais cela implique que deux autres `QGroupBox` soient créés et imbriqués dans le bloc `Pavage`, car l'échelle devra bien sûr être adaptable aux valeurs actuelles du pavé, ces derniers n'étant pas systématiquement  $[-10 ; 10]$  grâce à la fonction `zoom`. Malheureusement l'imbriication des `GroupBox` alourdit significativement le code.

## 6 Création d'une interface permettant la visualisation du résultat

Dans le cadre du logiciel commun devant être conçu par le groupe, nous avons en charge l'implémentation de certaines fonctionnalités devant être intégrées dans l'interface graphique. Nous partageons cette tâche avec deux autres binômes, ce qui impliquait une communication continue et ce dès les phases de spécification et de conception entre les groupes qui étaient en charge de la modification de l'interface. Par la suite une fois la partie technique réalisée, l'objectif était de réussir l'intégration globale avec les autres membres du groupe de travail. Cette partie du rapport propose de présenter les nouvelles fonctionnalités que nous avons ajoutées au code préexistant et leurs intégrations au reste du code.

### 6.1 Prise en main de l'interface graphique

Comme cela se fait souvent dans le monde industriel, nous sommes partis d'un code préexistant, auquel nous devons rajouter de nouvelles fonctions afin de satisfaire les spécifications du client.

L'interface graphique est en effet constituée de deux parties, un window, instance de la classe window, qui contient un objet frame, instance de la classe frame. C'est dans la frame que l'affichage du calcul de Sivia s'effectue. Notre objectif primaire est de pouvoir réaliser un " drag et zoom " dans la frame. Pour cela nous avons besoin de certains pré-requis comme avoir continuellement les coordonnées de la position de la souris. Dans un premier temps nous avons implémenté une nouvelle fonctionnalité dans la classe frame, qui puisse nous permettre d'avoir en temps réel les coordonnées de la souris et même de l'afficher dans une " QLineEdit ". Ceci en surchargeant la fonction suivante :

```
void Window::mouseMoveEvent(QMouseEvent *event)
{
    affichage_souris->setText(QString::number(event->pos().x()) + ", " + /
    QString::number(event->pos().y()));
}
```

Dans un premier temps les coordonnées de la souris ont été calculées en prenant comme origine le sommet de " l'objet window " en haut à gauche. Par la suite nous nous sommes rendu compte que cela faussait les calculs et que l'origine du repère devait se situer en haut à gauche de " l'objet frame ", contenu dans le window.

Une fois cette opération préliminaire effectuée, nous pouvions alors poursuivre vers la partie qui nous intéressait plus particulièrement ici, à savoir le " drag et zoom " avec un re-calcule de Sivia à chaque fois.

### 6.2 Le drag and zoom

C'est l'une des techniques graphique les plus usitées aujourd'hui. Le drag et zoom est en effet quasi inévitable dès que l'on veut travailler sur des systèmes à interface graphique relativement " grand public ". Pour mener à bien cette mission, nous avons procédé comme suit :

- Sauvegarde de la taille de la fenêtre initiale : nous créons deux variables de type INTERVAL où nous stockons les dimensions de la fenêtre initiale.

- de la fenêtre de zoom : grâce à un `mousePressEvent` et à un `mouseReleaseEvent`, tout deux implémentés dans la classe `frame`, nous enregistrons les coordonnées de la fenêtre sur laquelle l'utilisateur désire zoomer. Deux la classe `frame.h` nous avons définie deux variables de type `QPointF` dans lesquels les coordonnées sont sauvegardées.
- Implémentation de la fonction `zoom`: dans la classe `window`, et en utilisant l'objet `pavage`, nous avons réalisé la fonction `zoom`. De prime abord, il faut avoir en tête une donnée qui est souvent source d'erreur quand on travaille sur des interfaces graphiques. En effet l'origine du repère se situe en haut à gauche de notre fenêtre, mais il doit être représenté de la sorte : Il faut absolument avoir en tête que le repère est figuré de la sorte, afin de rectifier

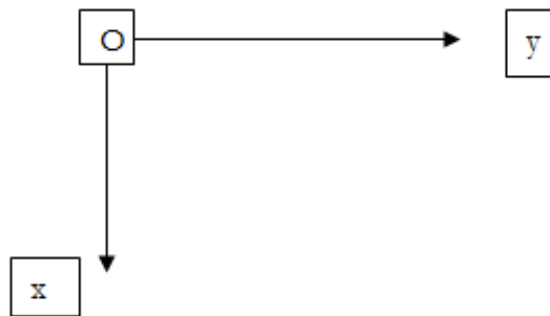


Figure 10 – Repère utilisé pour les interfaces graphiques

nos calculs théoriques sur l'axe `y` et obtenir les résultats prévus. Durant le codage de la fonctionnalité `drag` et `zoom`, cela a constitué une source d'erreur et d'incompréhension qui nous a bloqués durant quelques heures. Une fois ce réglage effectué, le problème majeur auquel on était confronté était la " conversion " des coordonnées obtenues en pixel sous forme d'intervalle. En effet les points obtenus avec la méthode implantée précédemment nous étaient fournis en pixel, alors que nous travaillions essentiellement avec des intervalles. Aussi en appliquant la formule suivante, nous obtenions les bornes inférieures et supérieures de nos nouveaux intervalles :

```
double i1inf = Inf(I1)+ Diam(pavage1->Inter1)
               *pavage1->pressclick.x()/(pavage1->width());
double i1sup= Inf(I1) + Diam(pavage1->Inter1)
               *pavage1->releaseclick.x()/(pavage1->width());
double i2inf= Inf(I2) + Diam(pavage1->Inter2)
               *(pavage1->height() - pavage1->pressclick.y()/(pavage1->height()));
double i2sup=Inf(I2) + Diam(pavage1->Inter2)
               *(pavage1->height() - pavage1->releaseclick.y()/(pavage1->height()));
```

`I1inf` et `I1sup` représentent les bornes inférieures et supérieures de la largeur de la nouvelle fenêtre. `I2inf` et `I2sup` celles de la longueur. En bleu, nous avons utilisé la fonction `Diam` qui donne en sortie la longueur d'un intervalle donné et en vert, `width`, qui nous donne la taille de la fenêtre en pixel. Avec ce calcul on arrive à constituer le nouvel intervalle de la fenêtre zoomée. Par ailleurs on peut remarquer pour le calcul des bornes de la hauteur un terme correctif a été rajouté pour être conforme au repère de l'interface (voir encadré rouge).



- Appel de la fonction Sivia : avec les nouveaux intervalles calculé, on appelle la méthode Compute. Ainsi on effectue le zoom voulu. Aussi nous avons rajouté une fonctionnalité qui permet de modifier la précision automatiquement avec le zoom que l'on effectue. Le calcul est le suivant : `double sauv1= Diam (pavage1->Inter1); precisionSpinBox->setValue( Diam (pavage1->Inter1)/sauv1);` Ainsi la précision évolue continuellement avec les zooms successifs.
- Implémentation de la fonction dézoomer : nous avons aussi introduit un nouveau bouton qui permet à l'utilisateur de revenir à la fenêtre d'origine.

```
void Window::dezoom()
{
    pavage1->Inter1=INTERVAL (-10,10);
    pavage1->Inter2=INTERVAL (-10,10);
    precisionSpinBox->setValue( 0.1);
    compute();
}
```

Ainsi le " drag and zoom " fut implémenté dans la classe window. Il ne restait plus qu'à établir les signaux et les slots nécessaires afin de faire communiquer les objets pavages et window entre eux.

- Utilisation des signaux : Le signal release, déclaré dans frame.h, est émis par l'objet pavage1 de type frame, au moment du relâché de la souris :

```
void Frame::mouseReleaseEvent( QMouseEvent *event )
{
    releaseclick.setX(event->pos().x()) ;
    releaseclick.setY(event->pos().y()) ;
    emit released();
}
```

La position du clic est préalablement récupérée par la fonction :

```
void Frame::mousePressEvent( QMouseEvent *event )
{
    pressclick.setX(event->pos().x() ) ;
    pressclick.setY(event->pos().y()) ;
}
```

Ceci permet de récupérer dans l'objet de type window les coordonnées de clic et de dé clic de la souris sur l'objet pavage1. Ceci se fait en établissant les connections de la façon suivante :

```
void Window::createConnections()
{
    [É]
    connect(pavage1,SIGNAL(released()),this,SLOT(zoom()));
    connect(bdezoom, SIGNAL(clicked()),this, SLOT(dezoom()));
}
```

La fonction zoom s'exécute alors lors de ce dé clic. On voit que le même procédé est utilisé pour que la fonction dezoom s'exécute au clic sur le bouton correspondant.

- Fonctionnalité orthogonalité : Nous avons effectué ces opérations et les tests se sont montrés globalement concluants. Nous avons juste remarqué que lors des zooms successifs, l'orthogonalité des figures n'était pas maintenue. Nous avons par la suite créé une nouvelle fonctionnalité qui évite lors de plusieurs zooms que le graphe ne soit déformé. Cette fonctionnalité, lorsqu'elle est activée en cochant la checkbox correspondante, consiste à rétablir les proportions entre le rapport largeur-hauteur des coordonnées de la fenêtre, et le rapport donné par la résolution en pixels de la fenêtre. Le côté à redimensionner est toujours celui qui entraîne le plus petit zoom, afin de ne pas rogner la zone à zoomer. Activer cette option restaure directement l'orthogonalité de l'affichage, tandis que la désactiver ne prend effet qu'au zoom suivant, en donnant toute liberté au choix des nouvelles coordonnées. Un booléen squared est initialisé faux. Il est associé au checkbox squaredCheck, défini dans la fonction createControls. Le code suivant est inséré dans la fonction zoom :

```

if (squared)
{
double coeff = (Diam(pavage1->Inter1)*pavage1->height())/ (Diam(pavage1->Inter2)*pavage1->width());
if (coeff > 1)
{
double mid2 = Mid(pavage1->Inter2);
double inf2 = mid2-Diam(pavage1->Inter2)/2*coeff;
double sup2 = mid2+Diam(pavage1->Inter2)/2*coeff;
pavage1->Inter2= INTERVAL(inf2,sup2);
}
if (coeff < 1)
{
double mid1 = Mid(pavage1->Inter1);
double inf1 = mid1-Diam(pavage1->Inter1)/2/coeff;
double sup1 = mid1+Diam(pavage1->Inter1)/2/coeff;
pavage1->Inter1= INTERVAL(inf1,sup1);
}
}

```

Ainsi, on établit si le rapport entre les proportions de l'écran et de la fenêtre est le même (coeff = 1). Si ce n'est pas le cas, on redimensionne, dans les conditions que nous venons de définir. Ce schéma représente la différence entre un zoom classique et un zoom orthogonal :

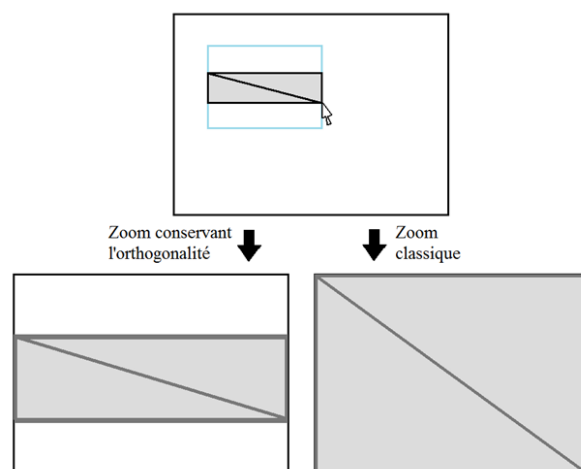


Figure 11 – schéma zoom classique-orthogonal

### 6.2.1 résumé Zoom et implémentation

8

Le zoom se déclenche lorsqu'un bouton de la souris est relâché dans le cadre. Le principe du zoom est simple :

- Les coordonnées des points de clique et de déclique sont enregistrées dans deux QPointF nommée pressclick et releaseclick
- Un signal appelé released() est ensuite envoyé à window qui déclenche alors la fonction zoom.
- Zoom détermine alors en fonction des points les deux nouveaux intervalles (fonctions rectifiées par le groupe chargé des fonctions au sein de l'interface)
- Si l'option squared est activée, le zoom est proportionnel à la plus petite des deux distances [fonctionnalité du groupe précédent], sinon, le rectangle défini par l'utilisateur est défini la nouvelle fenêtre [dont l'affichage est carré].
- La précision est ensuite modifiée de telle manière que le carré de précision ait la même taille à l'écran que le carré de précision avant le zoom.
- La fonction compute est ensuite lancée, elle recalcule et affiche le résultat.

A noter que c'est la fonction zoom qui a [la première] demandé une unique fonction compute, car elle avait été dupliquée avec l'option MonteCarlo, et ainsi les options BlackAndWhite et MonteCarlo ont dû être implémentées en variable et non en appelle de fonction différents.

---

<sup>8</sup>guillaume

L'implémentation de cette fonction a nécessité la création d'un slot (`released()`) car il n'y a pas de liens vers la classe `window` à partir de la classe `frame`, alors que l'événement déclenchant le zoom se trouve nécessairement dans `frame` (fonction `mousePressEvent`) car c'est une fonction qui surcharge la même fonction dans la classe mère `QWidget`.

### 6.3 Dezoom

Avec la fonction `zoom` va de paire une fonction `dezoom`. Un bouton a été créé pour redonner aux intervalles abscisses et ordonnées une valeur de  $[-10 ; 10]$  avant de relancer `compute`.

### 6.4 Amélioration du zoom

Il pourrait y avoir un champ pour spécifier le cadre : actuellement, la fonction `dezoom` fixe arbitrairement les valeurs des intervalles à  $[-10, 10]$ . De plus un cadre transparent pourrait s'afficher lors de la sélection.

## 7 Utilisation d'Ibex

9

### 7.1 Rappel de théorie et définitions

Dans cette partie, nous ne ferons que rappeler des notions sur l'analyse par intervalles. Elle est très largement inspirée des cours de méthode ensembliste pour la robotique de M. Jaulin à l'ENSTA Bretagne.

#### 7.1.1 Intervalle

Un intervalle  $[x]$  est un sous ensemble fermé et connexe de  $\mathbb{R}$ . L'ensemble de intervalles de  $\mathbb{R}$  sera noté  $\mathbb{IR}$ . Par exemple  $[1, 3]$ ,  $]1, 3[$ ,  $[3, 2]$  et  $[1, 2] \cup [3, 4]$  n'en sont pas.

Nous définissons la borne inférieure et la borne supérieure d'un intervalle comme suit :

$$x^- = \inf \{x | x \in [x]\} \quad (1)$$

$$x^+ = \sup \{x | x \in [x]\} \quad (2)$$

La longueur (en anglais width) d'un intervalle est définie par :  $w([x]) = x^+ - x^-$

L'intervalle enveloppe  $[X]$  d'un sous ensemble  $X$  de  $\mathbb{R}$  est le plus petit intervalle contenant  $X$ . Par exemple  $[[1, 3] \cup [6, 7]] = [1, 7]$ . Une redéfinition de l'arithmétique usuelle de  $\mathbb{R}$  est nécessaire pour pouvoir être appliqué à  $\mathbb{IR}$ . Notamment, les opérations  $+$ ,  $-$ ,  $*$ ,  $/$  ainsi que la plupart des fonctions usuelles couramment utilisées. Par exemple, les fonctions  $\exp$ ,  $\cos$ ,  $\sin$ ,  $\log$ , puissance et racine sont des fonctions potentiellement applicables de  $\mathbb{IR}$  dans  $\mathbb{IR}$ .

Une boîte  $[x]$  de  $\mathbb{R}^n$  est le produit cartésien de  $n$  intervalles :

$$[x] = [x_1^-, x_1^+] \times \dots \times [x_n^-, x_n^+] = [x_1] \times \dots \times [x_n]$$

L'ensemble des pavés de  $\mathbb{R}^n$  sera noté  $\mathbb{IR}^n$ .

#### 7.1.2 Contracteur

L'opérateur  $C_x : \mathbb{IR}^n \rightarrow \mathbb{IR}^n$  est un contracteur pour  $X \subset \mathbb{R}^n$  s'il satisfait

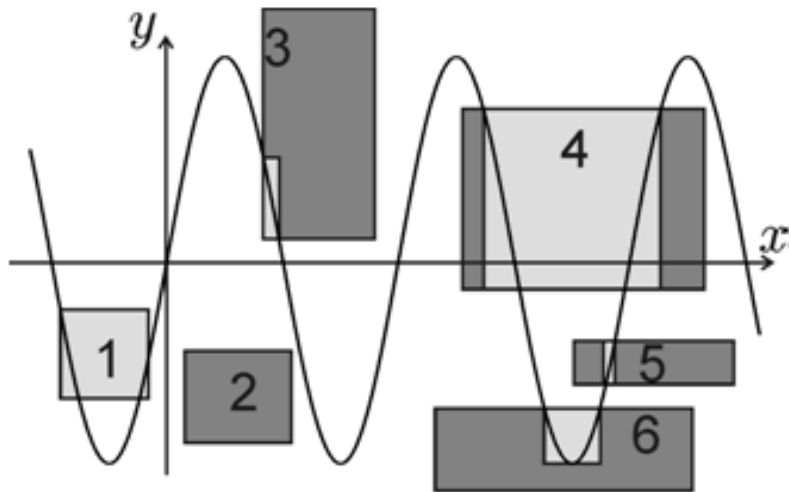
$$\forall [x] \in \mathbb{IR}^n,$$

$$\begin{aligned} C_x([x]) &\subset [x] \text{ (contractance)} \\ C_x([x]) \cap X &= [x] \cap X \text{ (complétude)} \end{aligned}$$

Un contracteur représente un ensemble de points qu'on appelle points fixes. Cet ensemble est généralement défini à partir de contraintes. Sur le schéma suivant, nous pouvons observer la contrainte  $y = \sin(x)$  qui sert à définir un contracteur  $C$ . Ici, les boîtes foncées numérotées de 1 à 6 sont contractées par  $C$  en boîtes claires. Les cas particuliers sont les boîtes 1 et 2. La boîte 1 est contractée en elle-même et la boîte 2 est contractée en l'ensemble vide.

---

<sup>9</sup>Equipe IBEX

Figure 12 – Contrainte  $y = \sin(x)$ 

Considérons un système de trois équations à deux inconnues suivant :

$$(C1): y = x^2$$

$$(C2): xy = 1$$

$$(C3): y = -2x + 1$$

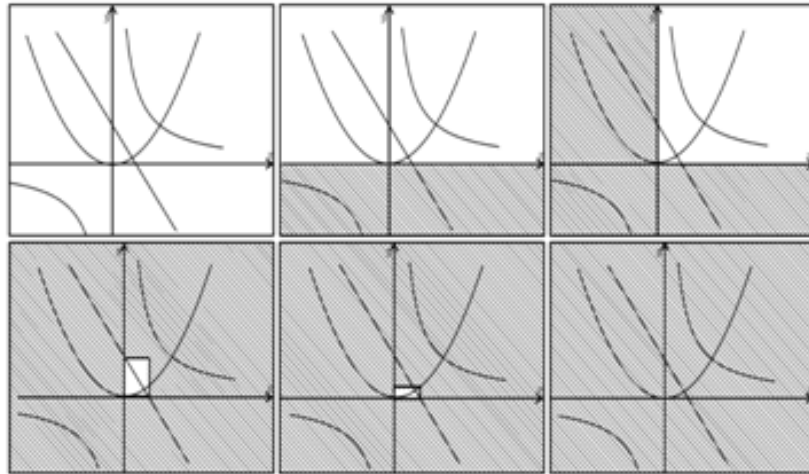


Figure 13 – Système de trois contraintes

Le graphe 13 représente ces 3 contraintes. En contractant avec  $C_1$ , nous obtenons le graphe 2. La partie foncée est la partie rejetée par le contracteur. La contraction avec  $C_2$  donne ensuite le graphe 3. Le graphe 4 est obtenu par contraction avec  $C_3$  puis on réapplique les contracteurs  $C_1$  et  $C_2$  pour obtenir les graphes 5 et 6. L'ensemble de solution est vide.

### 7.1.3 Relaxation

Il est généralement admis que tout moyen de mesure peut générer des valeurs de mesures erronées nuisibles. La robustesse de l'estimation est donc directement liée à sa capacité à s'affranchir de ces erreurs. Le principe d'intersection relâchée  $q$ -intersection est une solution permettant d'augmenter la robustesse. Il repose sur le relâchement de  $q$  contraintes. La figure ci-dessous illustre cette idée. Nous considérons 3 boîtes  $X_1$ ,  $X_2$  et  $X_3$ . Le schéma de gauche représente l'intersection non relâché. Dans le schéma du centre, nous avons relâché une contrainte et deux dans celui de droite. Nous obtenons quelques propriétés immédiates de l'intersection relâchée :

- la  $q$ -intersection est incluse dans la  $(q+1)$ -intersection
- la  $(N-1)$ -intersection de  $N$  boîtes l'union des  $N$  boîtes



Figure 14 – *Relaxation de 0, 1 ou 2 contraintes*

Il faut distinguer la q-intersection d'ensembles et la q-intersection de contracteurs. Dans le cadre du Bureau d'Etude, nous avons rapidement programmé la q-intersection d'ensembles dans un premier temps (voir la fonction " relaxedInside " mise en commentaire dans le programme). Afin d'utiliser les contracteurs d'ibex, nous avons dû implémenter notre propre algorithme de q-intersection, qui génère un contracteur représentant la q-intersection d'autres contracteurs (voir les fonctions " combi " et " siviaFile ").

#### 7.1.4 SIVIA

L'algorithme SIVIA (Set Inverter Via Interval Analysis) qui a été développé, utilise l'analyse par intervalles (outil mathématique permettant de calculer sur les ensembles) pour résoudre de façon globale et garantie les problèmes d'inversion ensembliste (c'est-à-dire ceux où l'ensemble solution peut être défini comme l'inverse d'un ensemble par une fonction). Les propriétés de SIVIA (convergence et complexité) sont établies après une étude approfondie de la structure topologique de l'ensemble des compacts. SIVIA permet de traiter une classe importante des problèmes comme l'estimation non linéaire ou la caractérisation de domaines de stabilité. L'algorithme est décrit ci-dessous. L est une pile dans laquelle les boîtes à traiter sont stockées. Si la boîte testée est dans la boîte cible alors on la colorie en rouge, si elle est en dehors de la boîte cible, elle est coloriée en bleu, si l'intersection est non vide et que la taille de la boîte est inférieure à un  $\epsilon$  qui correspond à la précision, on la colorie en jaune. Sinon, on bissecte la boîte en deux sous-boîtes que l'on stocke dans la pile L des boîtes à traiter. L'algorithme se termine quand toutes les boîtes ont été traitées et donc que la pile L s'est vidée<sup>10</sup>.

**Algorithme SIVIA (in : [x](0), f, Y)**

- 1)  $\mathcal{L} := \{[x](0)\};$
- 2) *pull*([x],  $\mathcal{L}$ );
- 3) *si*  $[f]([x]) \subset Y,$   
 $\text{draw}([x], 'red');$
- 4) *sinon si*  $[f]([x]) \cap Y = \emptyset$   
 $\text{draw}([x], 'blue');$
- 5) *sinon si*  $w([x]) < \epsilon$   
 $\text{draw}([x], 'yellow');$
- 6) *sinon couper* [x] en [x](1) et [x](2)  
 $\text{push}([x](1), [x](2), \mathcal{L})$
- 7) *si*  $\mathcal{L} \neq \emptyset,$  *aller à 2)* *sinon fin*

Figure 15 – *Algorithme sivia*

<sup>10</sup> " Solution globale et garantie de problèmes ensemblistes; applications à l'estimation non linéaire et à la commande robuste ", Jaulin Luc, Université de Paris 11, Orsay, France, 1994



## 7.2 Présentation de l'implémentation de l'analyse par intervalles

### 7.2.1 IBEX et contracteurs

IBEX, Interval-Based Explorer, est une bibliothèque permettant d'implémenter des solveurs et des paveurs basés sur les méthodes ensemblistes avec intervalle. Elle a été développée par M. Gilles CHABERT de l'Ecole des Mines de Nantes. Cette bibliothèque est particulièrement bien adaptée à notre travail vu qu'elle définit tous les objets utiles à l'arithmétique d'intervalle et à la manipulation de contracteurs.

Néanmoins, dans notre cas, ne pouvant utiliser la classe `qInter` permettant de réaliser une  $q$ -intersection (cf. 7.3.2), nous avons du développé un algorithme permettant de réaliser la  $q$ -intersection de liste de listes de boîtes. Pour cela deux solutions s'offraient à nous.

La première consistait à créer l'ensemble des intersections possibles avec les listes de boîtes correspondant à  $q$  instants  $t$ . Ces ensembles solutions étant tous stockés pour ensuite en réaliser l'union (et l'intersection pour les valeurs rejetées). Cependant cet algorithme nous est apparu gourmand en calcul ainsi qu'en mémoire. De plus, le problème du traitement des boîtes contractées par le contracteur de précision dans les ensembles calculés s'est posé et que nous n'y avons pas trouvé de solution simple.

La seconde solution, qui a été choisie, est de créer les contracteurs correspondant à cette  $q$ -intersection. Le principe mathématique est identique, puisqu'il s'agit toujours pour l'ensemble solution de trouver l'union des  $q$ -intersections possibles (et inversement pour l'ensemble non solution), seul l'algorithmie diffère. Cette solution permet a priori de diminuer l'empreinte mémoire, mais surtout elle permet de nous affranchir de trouver une solution au problème des frontières, ces dernières étant normalement traitées par l'algorithmique d'IBEX.

Le fait de choisir de choisir l'union de toutes les  $q$ -intersections possibles se comprend aisément, en ce qui concerne l'ensemble non-solution, cela l'ai déjà beaucoup moins. Pour déterminer ce dernier, nous pouvons utiliser les lois de De Morgan. Celles-ci nous indiquent en effet :

$$\begin{aligned}\overline{A \cup B} &\iff \overline{A} \cap \overline{B} \\ \overline{A \cap B} &\iff \overline{A} \cup \overline{B}\end{aligned}$$

Ainsi, si l'on cherche à obtenir l'ensemble des valeurs non solutions de la 1-intersection de 3 contraintes, qui n'est autre que le complémentaire de l'ensemble des solutions (l'union des 1 parmi 3 1-intersections), on a:

$$\begin{aligned}
\overline{\bigcap_{i=1}^3 \mathbb{X}_i} &= \overline{\bigcup_{i=1}^3 \bigcap_{j=1}^2 \mathbb{X}_i} \\
&= \overline{(\mathbb{X}_1 \cap \mathbb{X}_2) \cup (\mathbb{X}_1 \cap \mathbb{X}_3) \cup (\mathbb{X}_2 \cap \mathbb{X}_3)} \\
&= \overline{(\mathbb{X}_1 \cap \mathbb{X}_2) \cup ((\mathbb{X}_1 \cap \mathbb{X}_3) \cup (\mathbb{X}_2 \cap \mathbb{X}_3))} \\
&= \overline{(\mathbb{X}_1 \cap \mathbb{X}_2)} \cap \overline{((\mathbb{X}_1 \cap \mathbb{X}_3) \cup (\mathbb{X}_2 \cap \mathbb{X}_3))} \\
&= \overline{(\mathbb{X}_1 \cap \mathbb{X}_2)} \cap \overline{(\mathbb{X}_1 \cap \mathbb{X}_3)} \cap \overline{(\mathbb{X}_2 \cap \mathbb{X}_3)} \\
&= (\overline{\mathbb{X}_1} \cup \overline{\mathbb{X}_2}) \cap (\overline{\mathbb{X}_1} \cup \overline{\mathbb{X}_3}) \cap (\overline{\mathbb{X}_2} \cup \overline{\mathbb{X}_3}) \\
&= \bigcap_{i=1}^3 \bigcup_{j=1}^2 \overline{\mathbb{X}_i}
\end{aligned}$$

Donc, en généralisant, l'ensemble des valeurs non solutions est représenté par:

$$\begin{aligned}
\overline{\bigcap_{i=1}^q \mathbb{X}_i} &= \overline{\bigcup_{i=1}^q \bigcap_{j=1}^{p-q} \mathbb{X}_i} \\
&= \bigcap_{i=0}^{p-q} \bigcup_{j=1}^q \overline{\mathbb{X}_i}
\end{aligned}$$

On appliquera donc ces formules aux différents contracteurs dont nous disposons (chaque contrainte étant généralement représentée par 4 contracteurs, 2 pour définir l'ensemble des solutions et 2 pour l'ensemble des non-solutions). L'implémentation de cet algorithme est expliquée dans le paragraphe suivant.

### 7.2.2 Sivia.cpp

Le fichier `sivia.cpp` contient 12 fonctions nécessaires au déroulement de l'algorithme de résolution de système par contracteurs :

- `ibex::Env initEnv()`
- `ibex::Space initSpace(ibex::Env& env)`
- `ibex::Sequence getHC4Revise_in(ibex::Env& env, ibex::Space& space, int dataIndex)`
- `ibex::Sequence getHC4Revise_out(ibex::Env& env, ibex::Space& space, int dataIndex)`

- `int getAllHC4Revise_in(ibex::Env& env, ibex::Space& space, vector<const ibex::Sequence*>`
- `int getAllHC4Revise_out(ibex::Env& env, ibex::Space& space, vector<const ibex::Sequence*`
- `void delSequencesVector(vector<const ibex::Sequence*>& simpleCtc)`
- `void delContractorsVector(vector<const ibex::Contractor*>& combinedCtc)`
- `const ibex::Expr& EvalExpr(ibex::Env& env, int i)`
- `const ibex::Expr& EvalExpr(ibex::Env& env)`
- `int combi(ibex::Space& space, int q, const vector<const ibex::Sequence*>& simpleCtc_in`  
`const vector<const ibex::Sequence*>& simpleCtc_out, vector<const ibex::Contractor*`  
`vector<const ibex::Contractor*>& combinedCtc_out)`
- `void siviaFile(INTERVAL_VECTOR& X)`

Les six premières fonctions servent à l'initialisation de l'ensemble des objets ibex nécessaires : l'environnement, le domaine de valeurs et les contracteurs générés à partir de contraintes.

```

/#!/ @fn      ibex::Env initEnv()
* @brief      Initializes an ibex Environment with the symbols 'p1' and 'p2'.
* @author     Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version    1.0
* @date       2011-03
* @return     an ibex Environment.
*/
/#!/ @fn      ibex::Space initSpace(ibex::Env& env)
* @brief      Defines the domains of symbols 'p1' and 'p2' to [-10,10] in an Environment.
* @author     Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version    1.1
* @date       2011-03
* @param      env an ibex Environment by reference.
* @param      X an interval vector by reference, representing the domain.
* @return     an ibex Space.
*/
/#!/ @fn      ibex::Sequence getHC4Revise_in(ibex::Env& env, ibex::Space& space, int dataIndex
* @brief      Builds a HC4 contractor that colors the inside of a domain defined by a const
* @author     Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version    1.2
* @date       2011-03
* @param      env an ibex Environment by reference.
* @param      space an ibex Space by reference.
* @param      dataIndex an integer. It specifies the index of the value of
't' to take into account in the constraint.
* @return     an ibex Sequence that gives a Contractor.
* @var        Global variables used in this function are: 'data'
*/
/#!/ @fn      ibex::Sequence getHC4Revise_out(ibex::Env& env, ibex::Space& space, int dataIndex

```

```

* @brief      Builds a HC4 contractor that colors the outside of a domain defined by a cons
* @author     Luc Banchieri, Maxime Desgroussilliers & Olivier Voisin
* @version    1.2
* @date       2011-03
* @param      env an ibex Environment by reference.
* @param      space an ibex Space by reference.
* @param      dataIndex an integer. It specifies the index of the value of 't' to take into
account in the constraint.
* @return     an ibex Sequence that gives a Contractor.
* @var        Global variables used in this function are: 'data'
*/

/#!/ @fn      int getAllHC4Revise_in(ibex::Env& env, ibex::Space& space,
vector<const ibex::Sequence*>& ctcs_in)
* @brief      Adds to a vector all the HC4 contractors that color the inside of a domain,
for all the values of 't'.
* @author     Luc Banchieri, Maxime Desgroussilliers & Olivier Voisin
* @version    1.1
* @date       2011-03
* @param      end an ibex Environment by reference.
* @param      space an ibex Space by reference.
* @param      ctcs_in a vector of Sequences by reference. It is the vector to fill with the
* @return     an integer that is the number of Contractors added to the vector.
* @warning    For each Contractor, some space is allocated in the memory with 'new'.
* @warning    To avoid memory leaks, they will have to be deleted.
* @var        Global variables used in this function are: 'data'
*/

/#!/ @fn      int getAllHC4Revise_out(ibex::Env& env, ibex::Space& space,
vector<const ibex::Sequence*>& ctcs_out)
* @brief      Adds to a vector all the HC4 contractors that color the
outside of a domain, for all the values of 't'.
* @author     Luc Banchieri, Maxime Desgroussilliers & Olivier Voisin
* @version    1.1
* @date       2011-03
* @param      end an ibex Environment by reference.
* @param      space an ibex Space by reference.
* @param      ctcs_in a vector of Sequences by reference. It is the vector to fill with the
* @return     an integer that is the number of Contractors added to the vector.
* @warning    For each Contractor, some space is allocated in the memory with 'new'.
* @warning    To avoid memory leaks, they will have to be deleted.
* @var        Global variables used in this function are: 'data'
*/

```

Les deux fonctions suivantes (" delSequencesVector " et " delContractorsVector ") libèrent la mémoire réservée pour les contracteurs et les séquences de contracteurs, évitant ainsi les fuites de mémoire.

```

/#!/ @fn      void delSequencesVector(vector<const ibex::Sequence*>& simpleCtc)

```

```

* @brief      Desallocates the memory reserved for all the Sequences stored in a vector.
* @author     Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version    1.0
* @date      2011-03
* @param     simpleCtc a vector of Sequences.
*/

/! @fn      void delContractorsVector(vector<const ibex::Contractor*>& combinedCtc)
* @brief     Desallocates the memory reserved for all the Contractors stored in a vector.
* @author    Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version   1.0
* @date     2011-03
* @param    combinedCtc a vector of Contractors.
*/

```

Les deux fonctions " EvalExpr " contruisent les contraintes ibex, utiles à la génération des contracteurs, à partir des contraintes enregistrées dans l'arbre syntaxique.

```

/! @fn      const ibex::Expr& EvalExpr(ibex::Env& env, int i)
* @brief     Evals the expression of the constraint stored in the global variable 'syntax_
* @author    Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version   1.3
* @date     2011-03
* @param     env an Environment.
* @param     i an interger, that is the index of the current node.
* @return    an ibex Expression.
* @warning   This function is recursive and should not be called directly.
             Call 'EvalExpr(ibex::Env& env)' instead
* @var       This function uses the specific global variables 'evalExpr_dataString'
             and 'evalExpr_dataIndex' to recognize 't' and replace it by one of its values.
* @var       Others global variables used in this function are: 'syntax_tree', 'data'
*/

/! @fn      const ibex::Expr& EvalExpr(ibex::Env& env)
* @brief     Evals the expression of the constraint stored in the global variable 'syntax_
* @author    Luc Banchieri, Maxime Desgrousilliers & Olivier Voisin
* @version   1.0
* @date     2011-03
* @param     env an Environment.
* @return    an ibex Expression.
* @var       Global variables used in this function are: 'syntax_tree'
*/

```

Les deux dernières fonctions sont les fonctions clés de l'algorithme : " combi " et " siviaFile ". La fonction " combi " permet de générer un vecteur contenant l'ensemble des combinaisons possible de N-q éléments parmi N. Ces vecteurs permettent ensuite de créer les contracteurs représentant la q-intercétions de N contracteurs.

```

/! @fn      int combi(ibex::Space& space, int q, const vector<const ibex::Sequence*>& sim

```

```

* @brief      Prepares the q-intersection of contractors in input by adding the k-combinati
* @author     Luc Banchieri, Maxime Desgroussilliers & Olivier Voisin
* @version    1.3
* @date       2011-03
* @param      space an ibex Space.
* @param      q an integer that is the number of contractors to relaxe.
* @param      simpleCtc_in a vector of Sequences by reference. It represents the set of ins
* @param      simpleCtc_out a vector of Sequences by reference. It represents the set of ou
* @param      combinedCtc_in a vector of Contractors by reference. It represents the set of
* @param      combinedCtc_out a vector of Contractors by reference. It represents the set o
* @return     an integer that is the number of Contractors added to the output vectors.
* @var        Global variables used in this function are: 'cout_sivia'
*/

```

La fonction " siviaFile " déroule l'algorithme de résolution du système de contraintes en utilisant la q-intersection de contracteurs. Cette fonction n'a plus rien à voir avec l'algorithme Sivia et ne conserve son nom que pour une question de compatibilité avec le reste du logiciel. Il serait plus judicieux de la renommer qIntersectSolver ".

```

/#!/ @fn      void siviaFile(INTERVAL_VECTOR& X)
* @brief      Solves the constraint stored the global variable 'syntax_tree' using the q-in
* @author     Luc Banchieri, Maxime Desgroussilliers & Olivier Voisin
* @version    1.7
* @date       2011-03
* @param      X an interval vector by reference, representing the domain.
* @var        Global variables used in this function are: 'precision', 'relaxedData', 'pava
*/

```

L'algorithme se déroule selon les 4 phases suivantes :

1. Initialisation : Les objets d'ibex sont créés et initialisés. Les six premières fonctions sont appelées ici pour initialiser l'environnement, le domaine de définition, les contracteurs représentant l'intérieur du domaine de solution des contraintes, ainsi que les contracteurs représentant l'extérieur du domaine de solution des contraintes.
2. Q-Intersection : l'intersection relâchée de contracteurs se fait par la création d'un nouveau contracteur. Pour l'intérieur du domaine de solution des contraintes par exemple, on commence par calculer toutes les combinaisons d'intersection de N-q contracteurs parmi N. Ensuite, on définit un nouveau contracteur comme l'union de toutes ces combinaisons. Pour l'extérieur du domaine de solution, on fait l'intersection des combinaisons d'unions de N-q contracteurs parmi N.
3. Résolution : cette étape s'effectue par au travers de l'objet Paver d'ibex. On lui renseigne les deux contracteurs créés précédemment ainsi qu'un contracteur de précision pour assurer la terminaison de la résolution. L'appel à la méthode " explore " du Paver résout ensuite le système.
4. Affichage : la dernière étape consiste à transformer les boites résultats afin de pouvoir afficher à l'écran le résultat des contractions.

## 7.3 Difficultés rencontrés et solutions apportées

### 7.3.1 Affichage des pavés

L'affichage des pavés sur le frame nous a aussi posé un certain nombre de problème que nous avons fini par corriger. Notre problème surtout de la compréhension des objets d'IBEX et notamment de l'objet Paver. En effet, c'est cet objet qui va parcourir l'espace et le contracter. Nous avons passé beaucoup de temps à essayer de comprendre les fonctions box et rej de l'objet Paver. C'est en fait la cohérence ces noms qui nous a bloqué. Ainsi, nous pensions que rej renvoyait les boîtes éliminées (" rejetées ") par le contracteur alors qu'il donne accès à la boîte résultant de la contraction. La fonction box, elle, permet d'obtenir la boîte avant la contraction. A partir de ce constat, nous avons dû trouver une solution pour afficher les pavés contractés comme illustrés sur la figure ci-dessous.

1	2	3
4	5	6
7	8	9

Ainsi, la boîte bleu  $X_5$  correspond à paver.rej et la boîte  $\bigcup_{k=1}^9 X_k$  correspond à paver.box. Dans la mesure où nous voulons afficher les boîtes rouges, nous avons dû reconstruire chaque boîte rouge séparément à partir des valeurs des extrema de paver.box et paver.rej. Dans la plupart des cas, la boîte bleu est au contact d'un des bords de l'union de toutes les boîtes. Cette situation a dû être prise en compte ainsi que le cas où paver.rej renvoie l'ensemble vide. Pour simplifier le code, nous avons regroupé les boîtes rouge ainsi : 1,4,7, 1, 2,3, 3,6,9 et 7,8,9. Dans ce cas, nous n'avions que 4 boîtes à reconstruire au lieu de 8. Le problème était alors que comme ces grandes boîtes se superposent, la couleur des boîtes 1, 3, 7 et 9 changeait de teinte. Par soucis d'affichage, nous avons donc changé le regroupement des boîtes pour adopter celui de la figure ci-dessous.

1	2
4	3
5	

Ainsi les boîtes reconstruites ne se chevauchent plus et le nombre de boîtes reconstruites reste limité à 4.

### 7.3.2 Utilisation de IBEX/QInter

Dans IBEX, la classe Qinter a été créée pour permettre l'intersection relâchée d'intervalles. Ce contracteur réduit le domaine d'un vecteur  $X$  à l'enveloppe de toutes les intersections de  $q$  boîtes parmi  $p$ . Son implémentation est basée sur l'algorithme de Jaulin- Goldsztejn. Ce contracteur est typiquement utilisé pour intersecter  $p$  boîtes correspondant à des mesures d'un même objet avec des données aberrantes. Il est donc tout indiqué pour réaliser un  $q$ -intersection. Néanmoins, dans notre cas, Qinter n'est pas utilisable. En effet, à chaque instant  $t$ , nous allons générer une liste de boîtes solutions issues de chaque contraintes. Ce que nous voulons relâcher, c'est la liste de boîtes solutions de  $q$  instants. Or Qinter ne s'applique pas sur une liste de liste de boîtes mais seulement sur une liste boîte. En conséquence, nous n'avons pas pu utiliser Qinter, nous avons dû implémenter une fonction réalisant la  $q$ -intersection de liste de liste de boîtes. La principale difficulté de cette nouvelle fonction a été de pouvoir réaliser toutes les  $\binom{n}{n-q}$  combinaisons possibles d'intersections.

### 7.3.3 Utilisation des objets IBEX et mémoire

Une autre difficulté rencontrée au cours de ce projet a été l'utilisation des objets IBEX, en effet il a fallu trouver les différents types de données pouvant nous permettre de réaliser l'algorithme détaillé précédemment, ainsi que la manière qu'il faut utiliser pour déclarer l'environnement, l'espace et les contraintes initiales ainsi que leur contracteur associé.

Finalement, on initialise l'environnement et l'espace, ensuite on génère l'ensemble des contracteurs pour chacune des contraintes. Pour ce qui est des objets utilisés, les contracteurs sont générés à l'aide de la classe *HC4Revise*. Les différentes combinaisons sont obtenues en insérant ceux-ci dans un vecteur de contracteur, ce dernier est ensuite transformé en *ContractorList*. À partir de cette liste, on crée une *Sequence* réalisant l'union ou l'intersection des différents contracteurs. Cette opération est réalisée  $\binom{p}{p-q}$  fois. Les séquences, qui sont également des contracteurs, ainsi générées, sont insérées à leur tour dans 2 vecteurs. Ces vecteurs sont à leur tour transformés en *ContractorList*, puis en *Sequence*. Ces 2 dernières séquences sont utilisées sur l'espace afin de résoudre le problème.

De plus les mots-clés *const*, ainsi que les passages des objets directement ou de leur adresse nous ont également posé un certain nombre de problèmes et ont ralenti relativement fortement l'avancement de la partie.

Il est également apparu un problème de mémoire, en effet, lors de leur initialisation, les vecteurs n'allouent pas l'espace nécessaire pour les contracteurs qu'ils vont devoir contenir. Il faut donc allouer cet espace à chaque insertion de contracteur, et ne pas oublier de le libérer en fin d'exécution. La résolution de ce problème n'a pas été d'une grande difficulté, sa détection a, par contre, été plus difficile.



## 8 Méthode de Monte Carlo

11

### 8.1 Intérêt et Principe

Implémenter dans ce logiciel une méthode de Monte Carlo permet de fournir une alternative à l'algorithme Sivia, un moyen de vérification ou simplement une autre méthode d'évaluation. La méthode de Monte Carlo n'est pas ensembliste mais probabiliste. Le principe est d'effectuer un test sur un grand nombre de points tirés aléatoirement. On évalue l'expression pour un point donné. Ceci donne un réel qui est testé: s'il est inférieur à 0, il appartient à la solution, sinon, il n'est pas dans l'ensemble des solutions. Contrairement à l'algorithme de Sivia, il n'y a pas de zones d'incertitudes: le réel est dans l'ensemble de définition ou pas mais il n'existe pas de réels sur lesquels il pourrait y avoir une incertitude (en effet, il ne s'agit pas d'intervalles).

### 8.2 Implémentation

#### 8.2.1 classe montecarlo

Premièrement, afin d'implémenter la méthode de Monte-Carlo, j'ai ajouté une classe `monte_carlo.cpp` contenant la fonction `monte_carloFile()` dont les entrées et sorties sont strictement équivalentes à celles de la méthode `siviaFile()`. Ceci afin de permettre une utilisation indifférente des deux fonctionnalités et ainsi faciliter l'intégration au sein de l'interface graphique.

La fonction `monte_carloFile()` prend en entrée un `INTERVAL` contenant les valeurs de `px` et `py`. Ces valeurs sont évaluées à l'aide de la classe `syntaxtree` modifiée pour l'occasion et affichées dans le pavage principal: une box rouge autour du point `(px, py)` signifie que le point appartient à l'ensemble des solution, une box bleu, qu'il est hors de l'ensemble des solutions. Le test qui détermine cette appartenance est réalisé par la fonction `isIn()`. Un affichage de box (petite boîte autour du point évalué) permet de réutiliser le même type d'affichage que pour l'algorithme `sivia`.

La génération de points aléatoire se fait grâce à la fonction suivante:

```
double nbAlea (double borneInf, double borneSup)
{
    int taille = (int) (borneSup-borneInf);
    double r=(double)rand()/(double)RAND_MAX;
    double rez = r*taille + borneInf;
    return rez;
}
```

Cette fonction est appelée deux fois pour déterminer abscisse et ordonnée d'un point tiré au hasard. Notons que dans cette version du logiciel, les points sont choisis entre -10 et 10 et ce paramètre n'est pas paramétrable par l'utilisateur.

---

<sup>11</sup>Equipe Montecarlo

### 8.2.2 window

Pour la phase de test, j'avais originellement prévu un bouton "Monte Carlo" qui permettait de faire appel à la fonction `monte_carloFile()` à la place de `siviaFile()`. Dans la version définitive du programme, il s'agit de cocher une case monte carlo et le lancement du programme appellera alors la fonction de traitement monte carlo:

```
if(MC)
    monte_carloFile(X, color);
else
    siviaFile(X, color);
```

### 8.2.3 syntaxtree

Comme indiqué précédemment, la méthode de monte carlo n'utilise pas d'intervalles pour évaluer l'expression. La fonction `Eval(X)` dans la classe `syntaxtree` est donc inefficace. J'ai, par conséquent, ajouté la fonction suivante:

```
double EvalR(INTERVAL_VECTOR X , double px, double py)
```

Il s'agit d'une fonction récursive qui évalue l'arbre syntaxique (X) et retourne une valeur réelle en fonction des coordonnées (px, py) du point choisis par la classe `monte_carlo.cpp`.

## 8.3 Exemple de résultat

Ci-dessous, nous avons évalué la proposition suivante:  $\cos(x_1) * \sin(x_2) + \cos(x_2) * \sin(x_1) < 0$

L'image de gauche montre le résultat fournis par sivia tandis celle de droite montre le résultat fournis par la méthode de monte carlo.

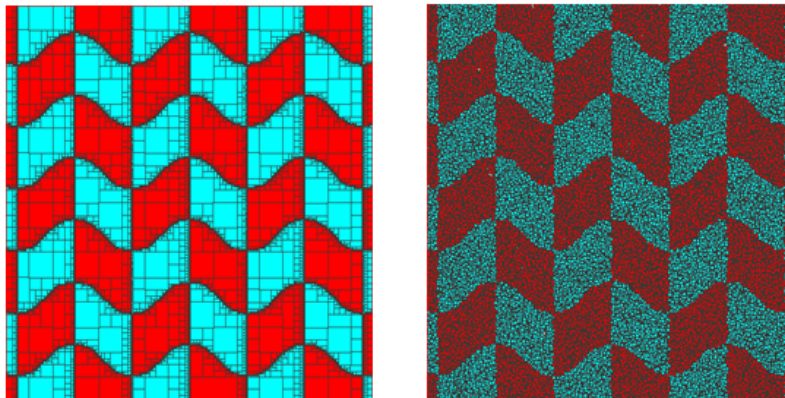


Figure 16 – *comparaison Sivia– Montecarlo*

On peut également régler la précision pour une méthode de monte carlo: plus la précision est fine (petite), plus le nombre de points évalué sera grand.

## 9 Modélisation

12

Le travail de ce groupe était de modéliser en UML l'intégralité du projet, et pouvoir générer du code directement à partir du modèle. Néanmoins, pour arriver à ce résultat, il a fallu passer par différentes étapes.

### 9.1 Importation de Qt à Rhapsody

Notre première idée fut d'importer directement le projet créé sous Qt dans Rational Rhapsody, un environnement de développement dédié à la modélisation UML, édité par IBM. A la suite de cet import, nous aurions pu améliorer les liens entre les différentes classes et générer un code sans trop de variables globales, et plus modulaire. Cette opération (code vers diagramme de classes) s'appelle le *roundtrip*, ou *reverse engineering*.

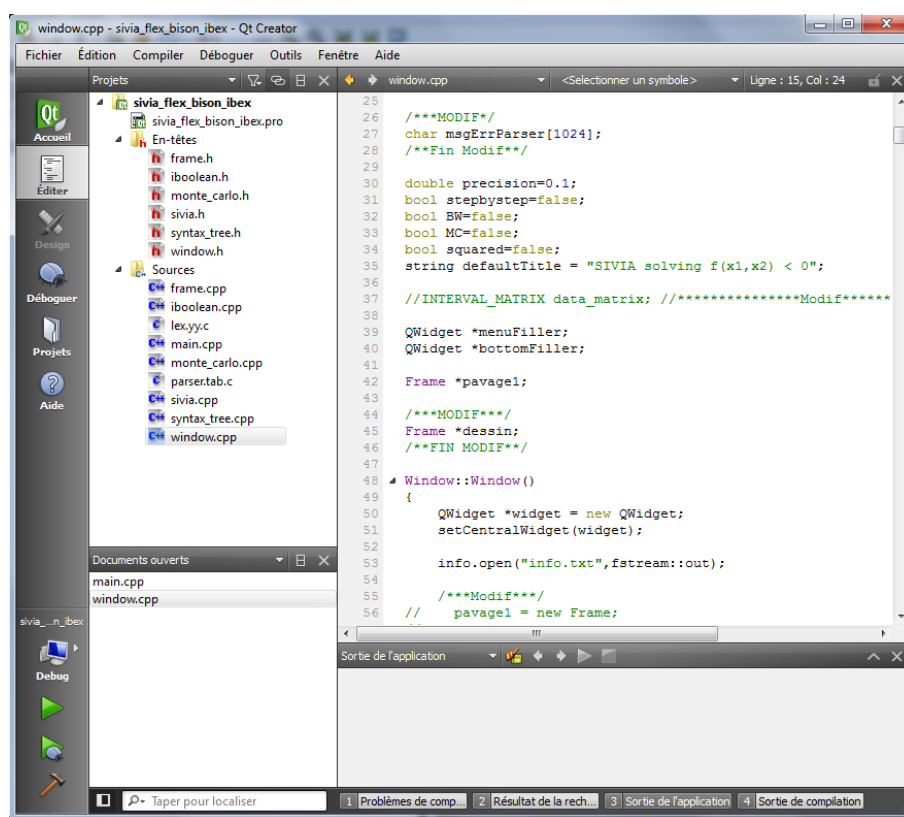


Figure 17 – *interface de Qt*

#### 9.1.1 Problèmes rencontrés

Le principal inconvénient du *roundtrip* est que celui-ci ne marche parfaitement qu'avec des langages purement orientés objets et ne nécessitant pas beaucoup de bibliothèques externes.

<sup>12</sup>Equipe UML

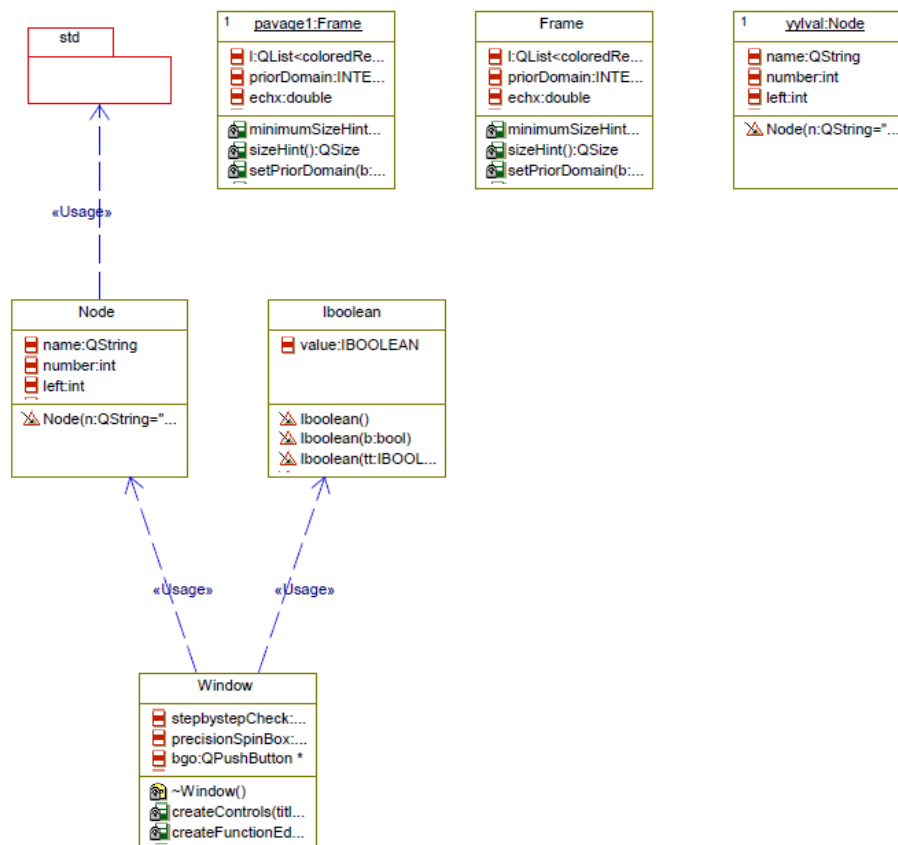


Figure 18 – Diagramme de classe généré par rountrip

De plus, les projets en C++ nécessitent une compilation globale, et la création d'un makefile est souvent nécessaire, et il faut donc spécifier à Rhapsody quel doit être la compilation. Nous reviendrons sur ce problème lorsque nous aborderons l'intégration de MinGW et MSYS.

## 9.2 Utilisation du plugin Qt sous Eclipse

Les diagrammes de classes générés par Rhapsody étaient incomplets. Or, nous avons constaté qu'il était possible de faire 'communiquer' ce dernier avec Eclipse, environnement de développement très utilisé. La gestion de projet sous Qt étant spécifique (création d'un fichier de référence .pro), nous avons trouvé un plug-in à Eclipse permettant l'importation du fichier .pro et donc du projet.

Malheureusement, ce transfert ne nous a pas réellement simplifié la tâche, la connexion entre Eclipse et Rhapsody n'étant pas aussi effective que nous l'avions pensé. Nous avons donc décidé finalement que la modélisation UML se ferait 'à la main'. Cette solution n'est pas très pratique, car le code initial et le code généré seront relativement différents, mais cela peut apporter une meilleure vision des liaisons entre les éléments dusdit code.

Néanmoins, nous avons procédé à la génération d'un diagramme de classe initial.

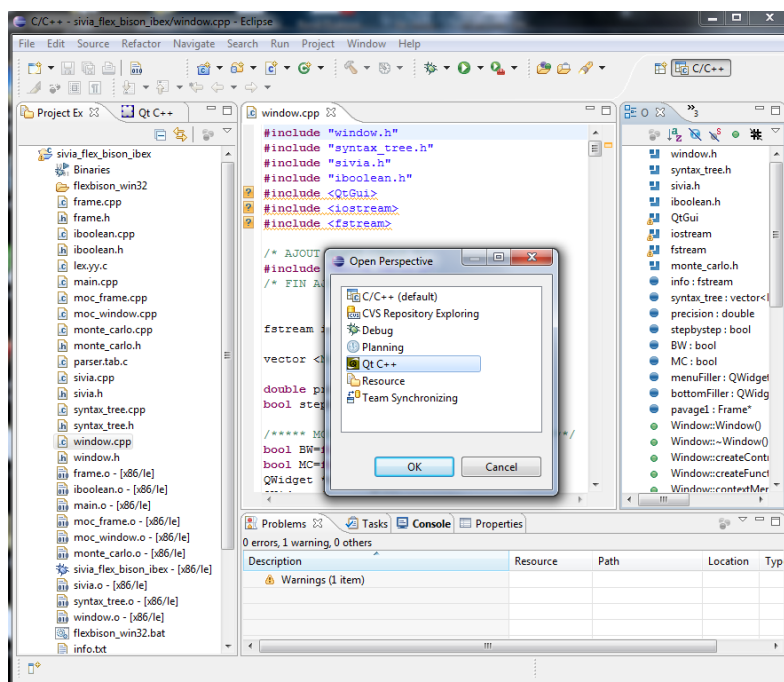


Figure 19 – vue du projet sous Eclipse

### 9.3 Génération de code C++ sous Rhapsody

Même si nous pouvions créer le modèle du projet en UML, le principal intérêt de cette méthode est la génération de code 'utile', à savoir exécutable et fonctionnel. Or, il existe de nombreuses différences entre compiler un code C++ sous Windows et sous Linux. Rhapsody fonctionnant sous Windows, l'import de certaines bibliothèques fut un souci.

#### 9.3.1 MinGW et MSYS

MinGW et MSYS permettent d'émuler un environnement linux comportant un compilateur c++ sous windows. Leur utilisation est indispensable à la compilation d'un projet C++. MinGW est nativement installé dans la plupart des Environnement de Développement, tout comme Cygwin, qui est un autre logiciel de compilation. Toutefois, pour que Rhapsody nous exécute et nous génère un code fonctionnel et pour qu'il reconnaisse la plupart des bibliothèques C++ et Qt, il a fallu les installer. Les erreurs obtenues par la suite lors de l'exécution du modèle furent des erreurs relatives au modèle et non plus à la compilation. Pour pallier ces erreurs, de nombreuses modifications ont dues être apportés au projet (suppression de bibliothèques et de fichiers sources...) et les résultats obtenus furent encourageants, bien que plus très pertinents vis-à-vis du projet.

## 9.4 Perspectives

Nous nous sommes rendus compte assez vite que la modélisation et l'exécution allaient être plus problématiques que prévu. C'est pourquoi nous nous sommes attelés à d'autres tâches. L'intégration du code final nous fut confiée, et bien que celle-ci s'est effectuée régulièrement au cours du projet par différents groupes, nous avons tout regroupé et assemblé. Nous détaillerons plus tard et plus en détail les problèmes relatifs à cette dernière. Plusieurs options furent envisagées, comme l'utilisation d'un éditeur UML crée sous Qt, BOUML. Bien que Rhapsody fut notre outil de travail privilégié, l'utilisation d'un éditeur plus simple a permis la génération de diagrammes de classes paraissant plus pertinents. Toutefois, ces diagrammes de classe ne furent pas exécutables.

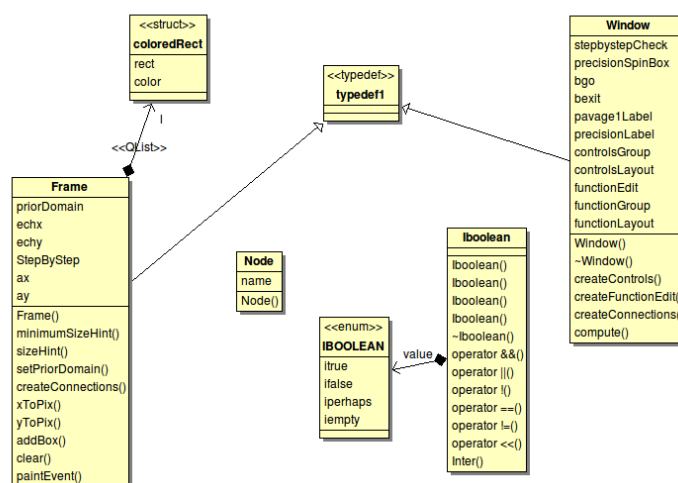
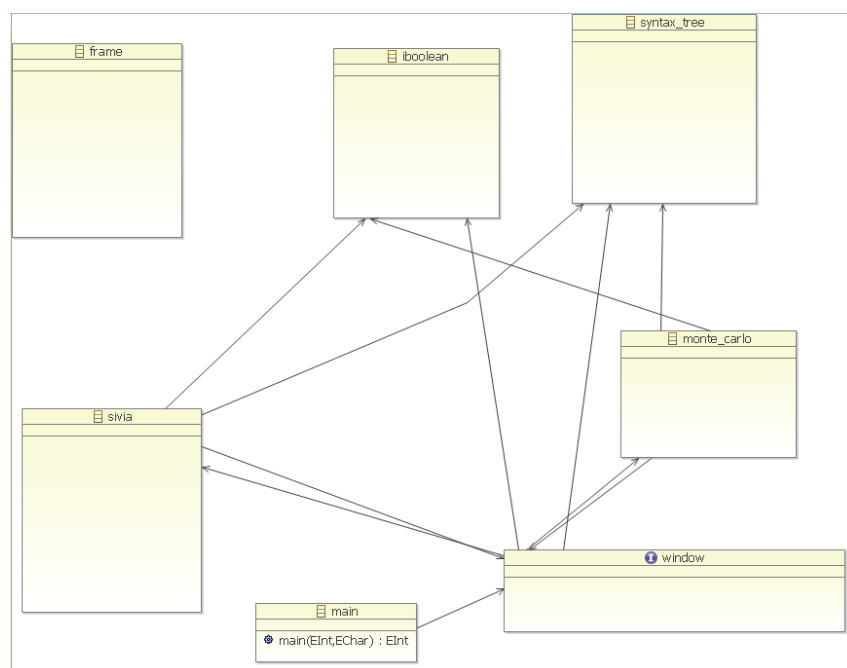


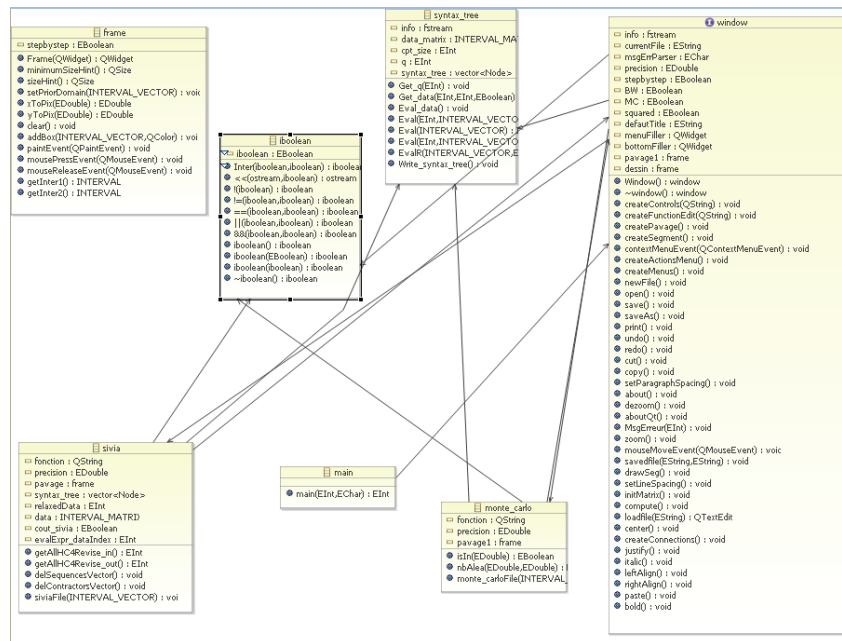
Figure 20 – Diagramme de classes généré avec Bouml

### 9.4.1 Modèle global

Une fois l'intégration terminée, nous avons pu créer le diagramme de classe du projet. Celui-ci est à but didactique, car il permet d'un coup d'oeil de repérer toutes les fonctions composant le projet, ainsi que les différentes liaisons. Les figures ci-dessous montrent la complexité du modèle, et de fait la difficulté inhérente à son exécutabilité.

Figure 21 – *modèle 'vide'*

La modélisation ne fut donc finalement pas une tâche critique, du fait des problèmes rencontrés et de la génération de code non conforme, et c'est pour cela que l'équipe chargée de cette tâche opéra différemment. La prise en charge du projet, à savoir la gestion des versions, du dépôt des fichiers, de l'intégration du rapport et du code lui fut attribuée en plus de la création du diagramme de classe du projet.

Figure 22 – *modèle complet*



## 10 Intégration

Dans cette partie, qui constituera en quelque sorte la conclusion, nous expliquerons comment s'est passée concrètement l'intégration du code et du rapport.

### 10.1 Intégration du code

<sup>13</sup> Du fait du grand nombre de fichiers modifiés simultanément lors du projet, l'établissement du google Document a permis une certaine hiérarchisation. Cette dernière n'a toutefois pas empêché bon nombres de problèmes récurrents.

#### 10.1.1 Qt

Même si Qt est un excellent environnement de développement, son architecture interne (création d'un fichier .pro, gestion automatique des sources) rend les projets peu transportables. Même si un code fonctionnait sur une machine, il pouvait très bien ne pas compiler sur une autre. Certaines erreurs ont pu être identifiées (suppression du profil utilisateur pour forcer Qt à tout recompiler avec les nouvelles sources), certains problèmes ont bloqué plusieurs équipes des heures durant. On peut par exemple citer le cas d'une intégration d'un fichier dans un projet qui ne marche pas alors que si l'on exporte le projet et qu'on le met dans le dossier source dudit fichier la compilation s'opère...

#### 10.1.2 Travail d'équipe

La plupart des fichiers du projet étant dépendants, un énorme travail de communication a dû être effectué, afin d'une part pouvoir se mettre d'accord sur certaines variables, et d'autre part avancer lors de l'intégration. Nous avons donc procédé à des intégrations partielles avant d'en choisir une et de la compléter. Cela a eu plusieurs avantages:

- avec plusieurs codes ayant été importés de manière différente, nous pouvions mieux nous préparer à corriger les erreurs, voire à choisir la marche à suivre lors de l'intégration pour avoir le moins de remaniement possible.
- Plusieurs groupes pouvaient avoir une version avancée du projet et donc pouvaient constater de l'efficacité de leur travail.
- plusieurs intégrations ratées en cours de projet valent mieux qu'une intégration finale impossible...

Tandis que le groupe UML s'attelait à trouver un moyen de créer un diagramme de classe exécutable, le groupe interface aidé du groupe bison procédait à des tests et des intégrations successives. Néanmoins, les intégrations et 'debuggage' finaux furent effectués par le groupe UML. Notons toutefois que le programme final ayant les mêmes problèmes que les versions bêta, l'une des versions les plus stable fut implémentée sur l'intégration du groupe interface.

---

<sup>13</sup>Equipes UML, Interface

Nous avons toutefois constaté qu'en fin de projet, le mieux est l'ennemi du bien; à vouloir améliorer sans cesse le code, on repousse l'échéance de l'intégration, voire on contrarie les personnes ayant travaillé sur cette dernière (l'ajout de code peut parfois rendre le travail de plusieurs heures obsolète). On se rend compte alors de l'importance de dates butoirs, mettant tout le monde d'accord quant à la version finale du projet.

## 10.2 Intégration du rapport

<sup>14</sup> Bien que cette partie n'entre pas en compte directement dans le projet, elle fut néanmoins un élément essentiel à la cohérence du rendu. Un seul rapport pour une quinzaine de personnes signifie soit une organisation sans faille dès le début, à savoir chaque groupe travaillant sur UNE partie spécifique, soit une harmonisation des productions complexe. Dans le cadre de ce projet, et dans l'optique de rendre un produit pouvant être vendu, l'organisation a beaucoup changé au cours du temps. Si un groupe avait fini sa tâche initiale, il s'attelait à une autre tâche, créant ainsi une sorte de "flou artistique", où chacun travaille pour le projet et non plus pour son groupe. Les principaux problèmes ont eu lieu lors de l'intégration, phase pendant laquelle tout le monde a mis la main à la pâte, que ce soit pour corriger les fonctions de l'un, ou pour enlever les variables globales de l'autre. Dès lors, la distinction de la production personnelle au sein du rapport est biaisée. Par exemple, lors de la rédaction finale effectuée par le groupe UML, nous nous sommes rendus comptes que plusieurs groupes avaient rédigé leurs rapports sans se concerter et il en a résulté des redites. Il a donc fallu faire des choix, voire demander aux personnes concernées de rédiger en intelligence leurs productions respectives.

## 10.3 Gestion du projet

<sup>15</sup> Le groupe UML s'est donné pour tâche d'être le groupe en charge de la gestion 'macro' du projet. Ainsi, l'administration du Google Document, La mise en commun des rapports (par nécessité d'une certaine homogénéité entre les productions des différents groupes) en LATEX, la communication générale (envoi de mails hebdomadaires voire quotidiens en fin de projet) furent accomplis par ce dernier. Il faut néanmoins souligner que l'ensemble des personnels impliqués dans le projet ont travaillé POUR le projet, et non pas seulement pour la tâche assignée. Sans cette émulation, la finalisation aurait été bien plus difficile, et bien qu'elle fut plus longue que prévue, l'intégration finale s'est passée sans trop de heurts, et s'il y a eu des problèmes, ces derniers concernaient les zones où plusieurs groupes travaillaient en parallèle.

En conclusion, l'accomplissement de ce projet fut une expérience intéressante. Nous avons pu constater que le travail assigné en début de projet peut radicalement changer, que le cahier des charges même peut être sujet à caution sur certains points, et que la moitié du temps en travail de groupe consiste à s'entendre et à faire fusionner les différents travaux. Finalement, les projets se concrétisent lorsque d'une part le travail est fait, mais surtout d'autre part il est fait en intelligence.

---

<sup>14</sup>Equipe UML

<sup>15</sup>Equipe UML

## List of Figures

1	vue du projet sous Google documents . . . . .	6
2	Structure du parseur ESTIM . . . . .	9
3	Mise à jour de scanner.l . . . . .	12
4	Message Box de retour d'erreur . . . . .	12
5	gestion de la casse . . . . .	12
6	Tests : arbre syntaxique généré (info.txt) . . . . .	13
7	arbre syntaxique . . . . .	14
8	vue de l'interface graphique . . . . .	18
9	schéma de l'héritage . . . . .	20
10	Repère utilisé pour les interfaces graphiques . . . . .	24
11	schéma zoom classique-orthogonal . . . . .	27
12	Contrainte $y = \sin(x)$ . . . . .	30
13	Système de trois contraintes . . . . .	30
14	Relaxation de 0, 1 ou 2 contraintes . . . . .	31
15	Algorithme sivia . . . . .	32
16	comparaison Sivia– Montecarlo . . . . .	42
17	interface de Qt . . . . .	43
18	Diagramme de classe généré par rountrip . . . . .	44
19	vue du projet sous Eclipse . . . . .	45
20	Diagramme de classes généré avec Bouml . . . . .	46
21	modèle 'vide' . . . . .	47
22	modèle complet . . . . .	48

## A Autres travaux

### A.1 Implémentation de la fonction d'ouverture de fichiers

```
QTextEdit* Window:: loadfile (QString aPathfile)
```

Cette fonction permet l'ouverture d'un fichier et le parcours des lignes de ce fichier. Elle sert à récupérer les données qui seront ensuite parsées.

### A.2 Plage d'affichage des segments

En plus d'afficher le résultat des calculs, le logiciel peut également afficher les ensembles récupérés via la donnée data (`data_matrix`). Ces données sont représentée par des boxes.

Cet affichage se fait dans un seconde pavage et est implémenté par la fonction suivante:

```
void Window::drawSeg()
```

Dans cette version du logiciel, les segments affichés ne sont pas dynamiques (c'est à dire qu'il ne peuvent être modifiés graphiquement par l'utilisateur). Néanmoins, les données peuvent être modifiées dans la zone de texte prévue à cet effet.