APIthagore
Rapport de soutenance 2

Epiquation

Fernando Borja
Lucas Rangeard

Thomas Boucard

26 Avril 2017

Table des matières

1 Introduction

Le projet Epiquation est un solveur d'équation, qui a pour but de faciliter le travail et les calculs avancés et simples. Le groupe APIthagire formé par trois étudiants d'Api (année préparatoire pour le cycle ingénieur) s'est assez facilement formé du fait d'avoir déjà travaillé plus d'un semestre ensemble. Notre projet étant assez ambitieux pour des étudiants de SPE, la phase de recherche et création d'algorithmes a été une étape importante et nécessaire avant toute implémentation. On a donc pensé à plusieurs techniques et après quelques recherches nous avons opté à utiliser les arbres binaires comme basse de notre solveur puis d'autres algorithmes tels que Crout. Ce rapport détaille le travail effectué depuis la naissance du projet. Nous verrons que ce dernier a bien avancé et respecte le rendu de la première soutenance et deuxième. Dans un premier temps nous présentons le projet de façon générale puis une présentation plus détailler partie par partie.

2 Présentation du projet

Alors que nous progressons dans le programme en mathématiques et que notre niveau évolue, la nécessité de pouvoir rapidement vérifier la justesse de nos calculs s'est faite sentir. Il nous est alors venu à l'esprit l'idée de développer un solveur mathématique qui puisse nous aider lors de notre formation, et nous donner plus d'aisance avec la matière.

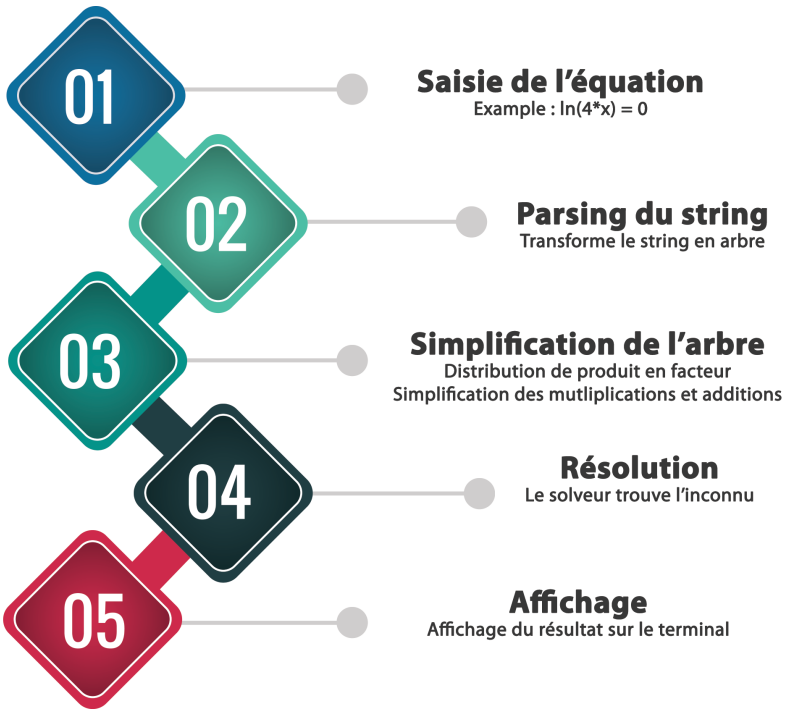

FIGURE 1 – Étapes de traitement soutenance 1

Sur la figure 1 les étapes de traitement du logiciel sont explicitement expliquées. Tout d'abord la saisie de l'équation sur le terminal suivi par le parsing qui s'occupera de créer des arbres avec les priorités de calculs, puis la simplification de l'arbre qui va simplifier le plus possible l'équation pour passer à l'étape quatre qu'est la résolution de l'équation, puis en dernière étape on gère l'affichage on montre explicitement le résultat de la variable.

Pour notre deuxième soutenance nous avons comme objectif de développer un algorithme de résolution d'équation du deuxième degrés et les systèmes. Pour cela nous avons dû se baser sur algorithme de Crout expliqué plus tard pour les systèmes et le discriminent pour les polynômes.

En ce qui concerne les étapes de traitement, elles sont exactement les mêmes

que sur la Figure 1, sauf qu'au lieu de faire une simplification de l'arbre nous ferons appel à l'algorithme de Crout ou bien nos algorithme de résolution de polynômes, tout cela est gérer par notre structure MVC [1] expliqué plus tard.

2.1 Objectifs soutenance 1

Pour la soutenance du 14/03/2017 nous avons planifié plusieurs tâches et objectifs, voici la liste de nos objectifs étudiés et développés pour cette soutenance :

1. Parsing
2. Solveur
3. Fonctions affines
4. Puissance et racine
5. Trigonométrie
6. Site Web

Aujourd'hui tous les points cités plus haut sont fonctionnels et affichent des bons résultats, les algorithmes implémentés ne sont pas encore optimisés.

2.2 Objectifs soutenance 2

Pour la soutenance du 25/04/2017 nous avons planifié plusieurs tâches et objectifs, voici la liste de nos objectifs étudiés et développés pour cette soutenance :

1. Parsing (Système et Polynômes)
2. Système
3. Polynômes
4. Interface (IHM)
5. Site Web

Nous avons donc réussi à attendre nos objectifs pour cette soutenance, nous sommes rester au second degré pour les polynômes pour des question de complexité que nous n'arrivons pas à gérer actuellement.

Ensuite nous avons notre résolution de système qu'est fonctionnel actuellement, pour cela nous utilisons la LU Décomposition expliquée plus bas.

1. Modèle-Vue-Contrôleur

3 Répartition des tâches

Soutenance (24/04/2017)			Fernando	Lucas	Thomas
Analyse					
	Solveur				
	Système		x		x [1]
	Derivée		x	x [1]	
	Primitive			x [1]	
Développement					
	Solveur				
	Système			x	x [1]
	Polynome			x [1]	x
	IHM		x [1]		x
	Site Web		x [1]		

4 Structure de données

4.1 Les unions

4.1.1 Le type

L'utilisation d'une variable de type (void*) pour stocker les données de l'arbre (sa structure sera expliquée plus tard) nous a poussé à utiliser une énumération pour définir les différentes données que celui-ci va être amené à stocker. Cet ensemble contient les valeurs suivantes :

— OPERATOR
— FUNCTION
— VARIABLE
— VALUE

4.1.2 Les opérateurs

Les opérateurs ayant un nombre de valeurs défini, nous avons aussi choisi de les représenter sous la forme d'une énumération contenant les valeurs ci-après :

— PLUS
— MINUS
— TIME
— UNKNOW

On peut remarquer l'absence de l'opérateur de division qui bénéficie d'un traitement spécifique expliqué plus tard dans ce rapport. Pour les spécificités de certains de nos algorithmes nous avons aussi rajouté la valeur "UNKNOW".

1. chef de la tâche

4.1.3 Les fonctions

Les fonctions sont gérées de la même manière que les opérateurs. Ainsi une liste de fonctions à été défini de même qu'une inconnu appelé "UNKNOW_F". Celle-ci contient :

- LN
- EXP
- POW
- SQRT
- COS
- ACOS
- SIN
- ASIN
- TAN
- ATAN
- UNKNOW_F

4.2 Les structures

4.2.1 L'arbre

L'arbre est la parti fondamentale de notre projet. C'est grâce à lui que nous pouvons prioriser les opérations. Sa structure est la suivante :

```
1  enum e_type
2  {
3    OPERATOR,
4    FUNCTION,
5    VARIABLE,
6    VALUE
7  };
8
9  struct          s_tree
10 {
11     struct s_tree *left;
12     struct s_tree *right;
13     enum e_type   type;
14     void          *data;
15 };
```

FIGURE 2 – Struct Tree

Elle contient un champ type qui permet de connaître le type du champ data. Ce dernier peut contenir des informations à propos d'un opérateur, d'une fonction, d'une variable ou un nombre.

4.2.2 Les fonctions

Pour pouvoir gérer les fonctions nous avons aussi utilisé une structure. Celle-ci va nous permettre de stocker diverses informations telle que le paramètre utilisé pour les fonctions puissance et racine. Ainsi que les champs pow et mult qui contiennent respectivement la puissance et le coefficient multiplicateur de la fonction. Sa structure est définie ci-après.

```
1  struct s_function
2  {
3    enum e_function *function;
4    float           param;
5    int             power;
6    float           mult;
7  };
```

FIGURE 3 – Struct fonction

4.2.3 Les variables

Les variables respectent le même principe que les fonctions à l'exception du paramètre qui n'est pas présent.

```
1  struct          s_variable
2  {
3    char          name;
4    int           power;
5    float         mult;
6  };
```

FIGURE 4 – Struct variables

4.2.4 Les listes

Les listes utilisées sont de simple listes chaînées utilisant une sentinelle.

```
1  struct s_list {
2    struct s_list *next;
3    struct s_tree *tree;
4  };
```

FIGURE 5 – Struct list

5 Les algorithmes

5.1 Le parsing

Le parsing consiste à rendre utilisable une chaîne de caractères pour notre programme. Ce parsing transforme donc une chaîne de caractères saisie par l'utilisateur en un arbre ayant pour racine le signe égal. Le fils droit(resp. gauche) sera la partie droite(resp. gauche) de l'équation. Ces arbres seront créés de manière à respecter les priorités d'opérations. L'ordre de priorité de traitement étant :

1. La recherche d'opérateurs prioritaire(si absence de parenthèses)
2. Recherche de fonctions
3. Gestion des parenthèses
4. Nombres ou variables

Un traitement est effectué pour les divisions afin de les transformer en multiplications. La fonction clean_string sera nécessaire afin de rendre l'équation saisie compréhensible pour le parseur. Pour l'instant, il est nécessaire d'inclure une variable et un signe égale dans la chaîne saisie.

5.2 La simplification des additions et des multiplications

La simplification des multiplications (resp. additions) passe tout d'abord par un parcours d'arbre qui lors d'une rencontre avec un signe multiplié (resp. additionné), commence à construire une liste avec n'importe quel type sauf une valeur. Si une valeur est rencontrée, elle est multipliée (resp. additionnée) au coefficient afin de le garder lors de la remontée récursive. l'arbre est ensuite reconstruit.

5.3 La reconstruction de l'arbre

La reconstruction de l'arbre récupère la liste résultant de la simplification. Pour reconstruire une suite de multiplications, il crée des nœuds contenant l'opérateur multiplier possèdent en feuilles les valeurs de la liste. La première feuille était multipliée par le coefficient. La reconstruction de l'addition fonctionne de manière similaire à la multiplication à ceci près que le coefficient est ajouté en feuilles de l'arbre reconstruit.

5.4 Multiplication d'un arbre

Cette étape correspond entre autre a un développement. Il y a 5 cas possibles :

- La rencontre avec un signe "+", l'appel recusif se fait sur les deux fils.
- Lors de la rencontre d'un signe "*", le rappel recursif ne se fait que sur le fils gauche.
- La rencontre avec une valeur multiplie celle-ci avec le coefficent retenu lors du parcours recursif.
- La rencontre avec une variable ajoute le coefficient à la structure associée à la variable.
- De même lors de la rencontre avec une fonction, le coefficient dans la structure est modifé.

5.5 Le solveur

Construites deux listes contenant les différents sous arbres séparés par une addition ou une multiplication, elles doivent toutes être séparées par le même symbole. Ces listes sont ensuite parcourues, les nœuds contenant des variables sont réunis dans une liste tandis que la valeur des autres nœuds est calculée. Le résultat est ensuite calculé et renvoyé à l'utilisateur.

5.6 le gestionnaire d'erreurs

Le gestionnaire d'erreurs affiche simplement une chaîne de caractères suivant un nombre (une variable statique) donné.

5.7 Méthode de Crout

La méthode de Crout nous a permis de calculer les systèmes. Cette méthode utilise la décomposition LU. Cette méthode consiste à décomposer une matrice en deux.

La première, la matrice L est une matrice triangulaire inférieur composé de 1 sur sa diagonal :

$$\begin{pmatrix} 1 & 0 & 0 \\ \alpha_{1,0} & 1 & 0 \\ \alpha_{2,0} & \alpha_{2,1} & 1 \end{pmatrix}$$

Ses autres valeurs sont calculées grâce à l'équation suivante :

$$\alpha_{i,j} = \frac{1}{\beta_{j,j}} \left(a_{i,j} - \sum_{k=0}^{j-1} \alpha_{i,k} \beta_{k,j} \right)$$

La deuxième, est une matrice triangulaire supérieur :

$$\begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} \\ 0 & \beta_{1,1} & \beta_{1,2} \\ 0 & 0 & \beta_{2,2} \end{pmatrix}$$

Et ses valeurs sont calculé grâce à cette expression :

$$\beta_{i,j} = a_{i,j} - \sum_{k=0}^{i-1} \alpha_{i,k} \beta_{k,j}$$

On peut constater deux choses, les éléments des deux matrices ne se superpose pas et un élément de la matrice d'origine est utilisé une seul fois lors de la décomposition. C'est pourquoi cette décomposition peut se faire en place. Néanmoins nous pouvons rencontré un problème lorsque la valeur calculée en $\beta_{j,j}$ dans l'équation précédente est égal à 0. Pour palier à ce problème, nous utilisons un pivot. Il faut entendre par là l'inversion de deux lignes. Pour choisir le pivot, nous avons choisi de sélectionner le plus grand coefficient, de plus nous avons créée un vecteur de pivot pour sauvegarder ces changements.

Une fois la décomposition effectuée, il ne reste plus qu'à résoudre le système. On pose l'équation suivante :

$$A \cdot x = (L \cdot U) \cdot x = L \cdot (U \cdot x) = b$$

Et on la résout en deux étapes. La première consiste à poser :

$$y = U \cdot x$$

Et donc à résoudre :

$$L \cdot y = b$$

Grâce aux équation suivante :

$$y_0 = \frac{b_0}{\alpha_{0,0}}$$

$$y_i = \frac{1}{\alpha_{i,i}} \left(b_i - \sum_{j=0}^{i-1} \alpha_{i,j} y_j \right) \quad i = 1, 2, ..., (n-1)$$

Puis à résoudre :

$$U \cdot x = y$$

A l'aide des résultats suivant :

$$x_{n-1} = \frac{y_{n-1}}{\beta_{(n-1)(n-1)}}$$

$$x_i = \frac{1}{\beta i, i}\left(y_i - \sum_{j=i+1}^{n-1} \beta_{i,j} x_j\right) \; i = (n-2), (n-3), ..., 0$$

Il ne reste plus qu'à lire le résultat.

5.8 Les polynômes

Concernant les polynômes, nous avons choisis de ne résoudre que ceux du second degré. Comme l'implémentation a été plutôt facile, nous avons choisis de renvoyer le delta et la/les solution/s de x. Afin de calculer de le delta nous avons utilisé la formule suivante :

$$\Delta = b^2 - 4ac$$

Une trouvé, il ne reste plus qu'a effectuer le calcule adequat suivant le delta.

$$Si\ \Delta > 0 \implies x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$Si\ \Delta = 0 \implies x = \frac{-b}{2a}$$

$$Si\ \Delta < 0 \implies x_{1,2} = \frac{-b \pm i\sqrt{\Delta}}{2a}$$

6 Problèmes rencontrés

Lors de la phase de développement, plusieurs problèmes ont été rencontrés. La recherche d'algorithmes efficace nous a demandé énormément de temps. La résolution de bugs a elle aussi été chronophage.

Voici la liste des problèmes majeurs rencontrés.

6.1 Création de l'arbre

La création de l'arbre a été possible grâce à un champ type et un pointeur sur data ajouté à la structure de l'arbre.

6.2 Gestion des soustractions

Il nous a été nécessaire de simplifier l'arbre le plus possible ainsi nous avons eu l'idée d'appliquer la soustraction sur le fils droit de l'arbre afin de remplacer les nœuds de celui-ci par des additions.

6.3 Portée des soustractions

La soustraction nous a posé un second problème lors de la phase de testés. En effet, lorsque l'équation saisie commençait par un signe moins, le résultat affiché n'était pas celui escompté. Il a fallu effectuer des vérifications sur les algorithmes afin que l'opération s'effectue de la manière dont nous le souhaitions.

6.4 Gestion des divisions

Par un procédé similaire à celui de la soustraction, la division dans l'arbre s'opère en remplaçant cette opération par une multiplication a la puissance -1. Il a bien sûr fallu s'assurer que le nombre n'était pas nul.

6.5 Opérations sur l'arbre

Enfin, l'un des plus gros problèmes fut la gestion des opérations sur l'arbre. Nous avons finalement opté pour une cohabitation entre arbres et listes. Si nous avions seulement utilisé la manipulation des arbres, le nombre de cas à prévoir aurait été bien plus nombreux.

7 Technologies utilisées

Certaines contraintes et besoins quant aux technologies utilisées. On du Les contraintes sont les suivantes :

- Le projet est fait sous **Linux**.
- Il a été codé en **C**.
- Le projet fonctionne sur les **machines de l'école**.
- Un **site** est présent afin de montrer l'état d'avancement du projet.

De même nous avons utilisé les librairies standard suivantes :

- err.h.
- math.h.
- regex.h
- stringh.h.
- stdlib.h.
- stdio.h.
- sys/types.h

7.1 Le Regex

Nous avons utilisé pour ce projet la technologie regex qui permet de détecter des motifs dans des chaînes de caractères.

Disponible sur les ordinateurs de l'école par le biais de la bibliothèque Posix « regex.h »cette technologie nous a permis d'analyser les chaines de caractère.

7.2 GTK+ 3 et Glade

Pour ce projet nous avons fait le choix d'utiliser la bibliothèque GTK+ pour tout ce qui concerne la partie graphique et interface.

Cette bibliothèque a été développée originellement pour les besoins du logiciel de traitement d'images GIMP, puis GTK+ est maintenant utilisé dans de nombreux projets, dont les environnements de bureau GNOME, Xfce, Lxde et ROX.

Mais pour notre projet nous avons utiliser de même le logiciel Glade qu'est un outil interactif de conception d'interface graphique GTK+. Il prend en charge toute la partie de gestion/génération de l'interface pour permettre au développeur de se concentrer sur le code "utile".

Glade enregistre les interfaces graphiques en générant des fichiers XML et nous n'avons donc qu'à développer la partie "utile" cela veut dire le code en C avec la bibliothèque GKT+ 3.

Nous avons de même opté pour une structuration de type MVC (Modèle-Vue-Contrôleur), le MVC est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978 et très populaire pour les applications web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

1. Un modèle contient les données à afficher.
2. Une vue contient la présentation de l'interface graphique.
3. Un contrôleur contient la logique concernant les actions effectuées par l'utilisateur.

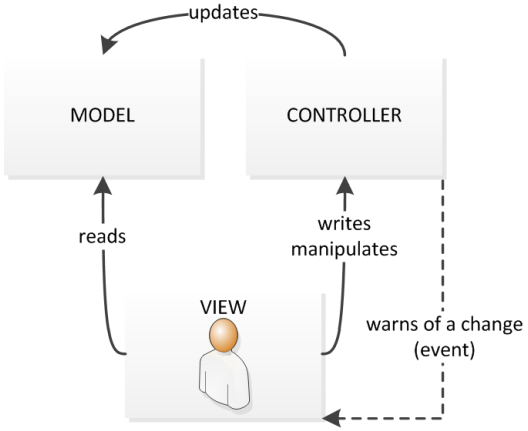

FIGURE 6 – Modèle MVC

Sur la figure 6 nous avons une idée de la structure MVC, précisément chaque point correspond à une partie importante du logiciel :

Le modèle :
C'est l'élément qui contient les données ainsi que de la logique en rapport avec les données : validation, lecture et enregistrement. Il peut, dans sa forme la plus simple, contenir uniquement un simple texte, voire des données beaucoup plus compliquées, dans notre cas il s'agit de tous les sous-programmes qui effectuant les calculs. Le modèle représente l'univers dans lequel s'inscrit l'application. Par exemple pour notre application de banque, le modèle représente les algorithmes de calcul, de pivotement, de constructions des arbres, listes. Le modèle est indépendant de la vue et du contrôleur et peut être utilisé sans les deux autres.

La vue :
C'est la partie visible d'une interface graphique. La vue se sert du modèle, et peut être un diagramme, un formulaire, des boutons. Une vue contient des éléments visuels ainsi que la logique nécessaire pour afficher les données provenant du modèle. Dans notre cas, il s'agit de notre interface graphique crée grâce à GTK+ 3 et Glade.

Le contrôleur :
C'est le module qui traite les actions de l'utilisateur, modifie les données du modèle et de la vue. Dans notre projet il s'agir de tous les algorithmes chargé de traiter les données reçues par l'interface graphique et les transmettre sur le bon format au modèle.

L'ensemble :
Les trois éléments sont indépendants les uns des autres, le modèle ne se sert ni de la vue ni du contrôleur, il peut cependant leur envoyer des données. Il y a deux liens entre la vue et le modèle : premièrement la vue lit les données du modèle et deuxièmement reçoit des données provenant du modèle. Dans la mesure où une vue est associée à un modèle et un modèle est indépendant, un même modèle peut être utilisé par plusieurs vues.

Le contrôleur dépend de la vue et du modèle : la vue comporte des éléments visuels que l'utilisateur peut actionner. Le contrôleur répond aux actions effectuées sur la vue et modifie les données du modèle.

8 Site Web

Le site Web est une partie importante et essentielle du projet, il sert de vecteur majeur de communication avec la communauté des développeurs mais aussi au public. Celui-ci sert à présenter le projet de façon global mais aussi de façon précise si le visiteur veut en savoir plus. Nous avons opté pour un site "One-page" qui permet d'avoir tout le contenu que nous avons besoin sur une unique page, nous avons de même opté pour un site assez épuré mais aussi attirant et facile d'utilisation.

Sur celui-ci nous présentons le projet, le groupe, notre répertoire "Github" de

la version actuel du projet, le cahier des charges et tous les rapports de soutenances sont disponibles. Puis nous essayons de mettre régulièrement à jour le site avec des news et aussi en mettant à jour les animations rendant celui-ci assez interactif.

Nous avons opté à l'héberger sur "Github" car grâce à nos adresses mail de l'EPITA nous avons des comptes professionnels gratuits nous donnant assez à la partie "Pages" de Github, ainsi l'URL du site est :

https ://fadao23.github.io/webepiquation/

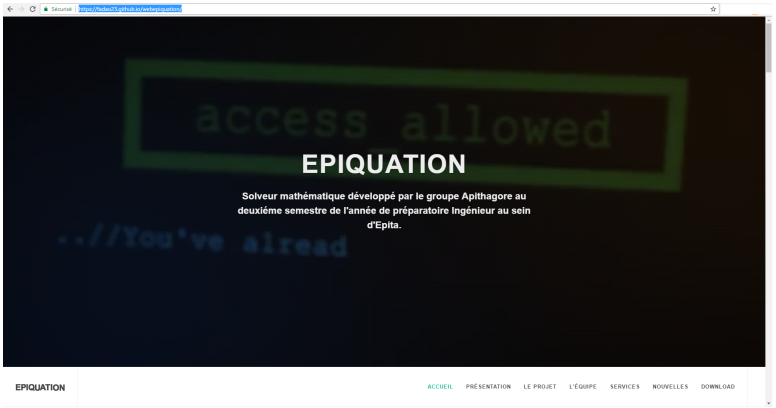

FIGURE 7 – Index site Web

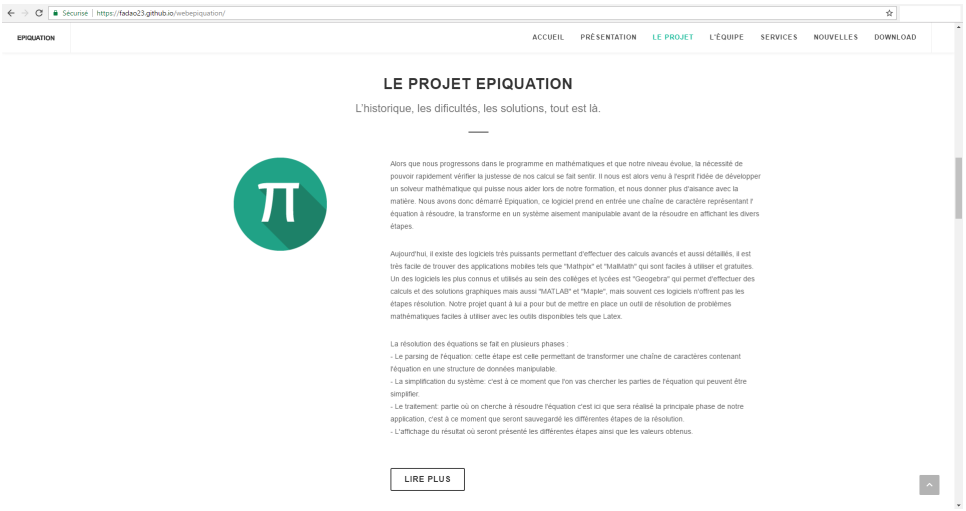

FIGURE 8 – Présentation projet

Sur la partie présentation du projet nous avons fait une simple présentation avec l'état de l'art sur les logiciels et application mobiles disponibles sur le marché actuellement. Si l'utilisateur souhaite avoir plus d'informations, nous avons inclus un bouton "Lire plus" qui redirige vers le cahier des charges du projet géré avec "PDF js" de chez Mozilla.

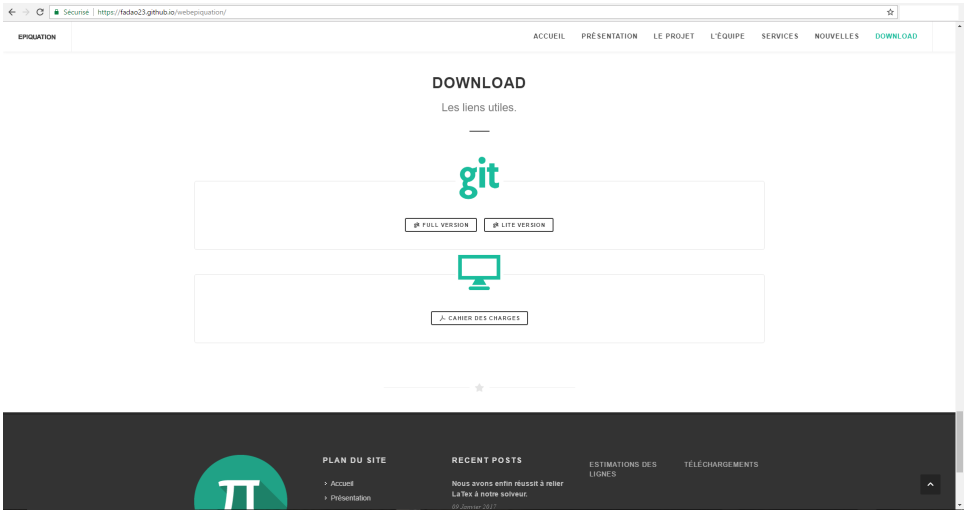

FIGURE 9 – Téléchargements

9 Objectifs soutenance N°3

Soutenance (29/05/2017)		Fernando	Lucas	Thomas
Analyse				
	Primitive		x	x [1]
	Derivée		x [1]	x
Développement				
	Derivée	x	x [1]	
	Primitive	x	x [1]	
	IHM	x [1]		x
	Site Web	x [1]		

Pour la prochaine soutenance qui se déroulera dans la semaine du 29/05/2017 nous avons plusieurs objectifs, tout d'abord nous allons continuer à travailler sur un aspect visuel de notre logiciel avec le développement de l'IHM, ensuite nous souhaitons avoir des résolutions des primitives et dérivées, il faut savoir que les fonctions affines et polynômes de degré 2 sont actuellement gérer par notre logiciel grâce à l'algorithme de crout pour le deuxième degrés par exemple. Nous avons de même pour objectif d'optimiser nos algorithmes mais aussi d'avoir une plus grande gestion des erreurs avec un parsing améliorer.

1. chef de la tâche

Table des figures