|  |
| --- |
| CFPTI |
| TPI |
| Grapheur de fonction |
| **Trifunovic Ivan** |
| **Sous la direction du Professeur Christian Blanvillain** |
| **19/06/2017** |

Table des matières

[Introduction 2](#_Toc485102701)

[Fonction paramétrique 3](#_Toc485102702)

[Cahier des charges 4](#_Toc485102703)

[Etude d’opportunité 5](#_Toc485102704)

[Analyse fonctionnelle 6](#_Toc485102705)

[Fonctionnalités du projet 6](#_Toc485102706)

[Interface 7](#_Toc485102707)

[Analyse organique 8](#_Toc485102708)

[Architecture 8](#_Toc485102709)

[Structogrammes 9](#_Toc485102710)

[Méthodes de réalisation 10](#_Toc485102711)

[Tests 11](#_Toc485102712)

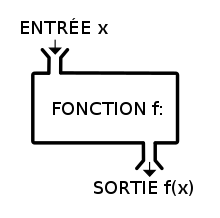
[Pan de tests 11](#_Toc485102713)

[Bibliographie 13](#_Toc485102714)

# Introduction

## Fonction cartésienne affine

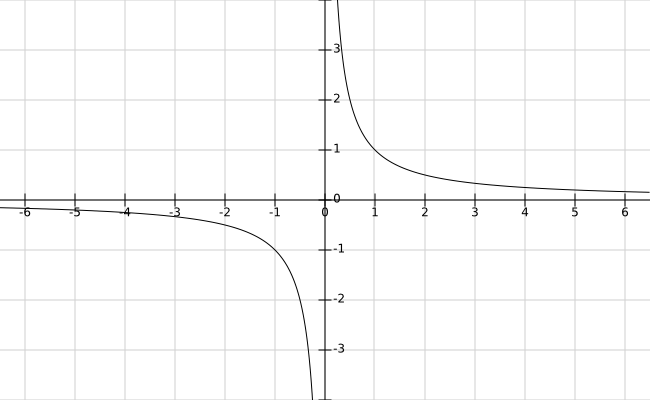
Une fonction est une relation entre deux ensembles un ensemble d’entrée et un autre de sortie.



Un exemple de fonction est la fonction

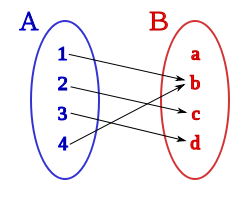
On remplace le paramètre qui par les entrées suivantes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x Entrée | -5 | -2 | -1 | 0 | 1 | 2 | 5 |
| f(x) Sortie | -0.2 | -0.5 | -1 | Erreur | 1 | 0.5 | 0.2 |



L’ensemble de départ A ne peut pas avoir plus d’une sortie alors que l’ensemble de sortie B peut avoir plusieurs entrées A.

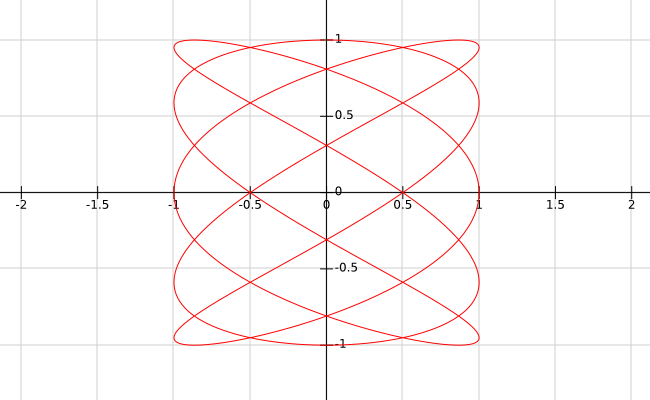
On peut voir dans l’image ci-dessous que A(1) et A(4) ont le deux la même sortie B(b) alors qu’aucune entrée ne possède deux sorties.



## Fonction paramétrique

Une fonction paramétrique est une fonction qui en entrée contient une fonction et en sortie une autre et ces deux fonctions on le même paramètre souvent noté t.

Exemple de courbe paramétrée



# Cahier des charges

L’interface présente : un menu, une zone de définition de l’équation et un graphe en 2D (x,y). Il est prévu deux types de courbes :

1. y(x) = … → Cartésiennes Affines

2. {x(t) = … ; y(t) = ...} → Cartésiennes Paramétriques

La touche entrée, valide l’équation et génère l’affichage de la courbe. Si l’équation n’est pas valide un avertissement est signalé à l’utilisateur et rien ne se passe.

Trois boutons d’action sont présents :



Le bouton Dessiner est activé uniquement lorsque l’utilisateur est en train de saisir une nouvelle équation. Il ouvre une fenêtre modale dans laquelle on peut :

● Définir l’intervalle d’affichage pour les valeurs de x ou de y, pour la zone d’affichage du graphe. L’autre dimension est toujours proportionnelle pour que le repère reste auto-normé en fonction de la taille de l’application. Par défaut centré sur zéro.

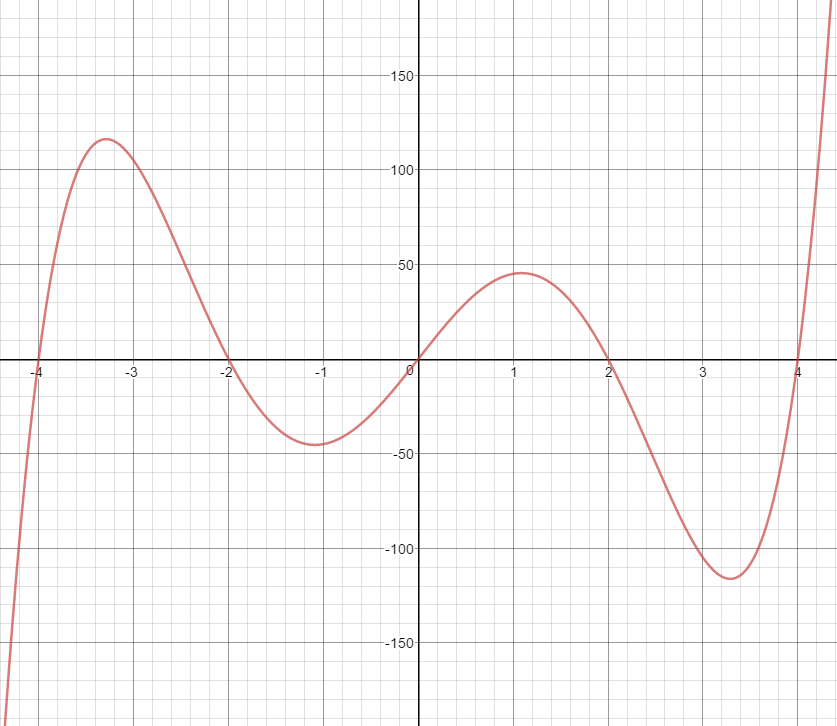
● Affiner le pas de calcul dont la valeur par défaut proposée dans la fenêtre sera calculée automatiquement en fonction du niveau de grossissement et de l’intervalle d’affichage de manière à éviter une trop forte pixellisation. La précision des calculs sera celle proposée de base par la librairie utilisée pour les calculs.

Le bouton Annuler permet de réinitialiser la saisie de l’équation d’une nouvelle courbe. Il ne se comporte donc pas comme un CTRL+Z. Il est activé uniquement lorsque l’utilisateur est en train de saisir une nouvelle équation.

Le bouton Effacer nettoie les courbes de la zone graphique d’affichage et la réinitialise, c’est à dire redéfinit l’intervalle d’affichage par défaut : [-5 ; +5] en x et y ainsi que le pas de calcul : 1 point par pixel affiché. Lorsqu’il est cliqué, un popup d’avertissement demande une confirmation avant d’effacer la zone d’affichage. Une fois la courbe effacée, il est impossible d’annuler l’opération.

Les éléments de l’interface sont en dehors du graphique.

La zone d’affichage des fonctions est similaire à la figure suivante :



L’agencement des éléments d’interface est similaire à cette capture d’écran :

# Etude d’opportunité

Dans le cadre de de mon projet TPI, j’ai voulu développer un grapheur qui consiste à reproduire le graphique d’une équation dans un plan R2 (le graphique d’une fonction). J’ai choisi de faire ce projet, car je trouve que les graphiques d’équations sont beaucoup plus parlants que les chiffres.

Dans mon projet, j’ai choisi de mettre en avant deux types de graphique. Les équations cartésiennes affines et les courbes paramétrées. Mon application possède une interface claire et rapide. Pour obtenir le résultat de l’équation et afficher la courbe il suffit juste de l’écrire de plus une option permet de régler la précision de calcul.

Dans les différents grapheur[[1]](#footnote-1) se trouvant sur le internet, beaucoup sont performants mais la plus part du temps il faut cliquer sur un bouton pour afficher le graphique. Certains ne sont pas précis et restreint à des petits intervalles avec une simple interface sans zoom. Bien sûr, il est impossible de reproduire une fonction parfaitement car on ne peut pas calculer un nombre infini de points sur un ordinateur.

# Analyse fonctionnelle

## Fonctionnalités du projet

Le programme permet de tracer le graphique d’une équation cartésienne affine et paramétrique dans un plan R2.

L’interface présente un graphique avec deux axes x et y. Deux Zone de textes sont présents en bas à gauche. L’utilisateur peut directement insérer du texte dans les champs.

En fonction de ce que l’utilisateur entre l’équation qu’il a entrée sera dessinée sur le graphique. Si l’équation est fausse le graphique de la fonction ne s’affiche pas.

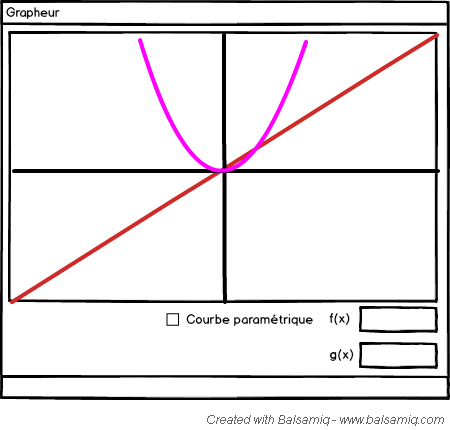
Un menu option est présent en haut à gauche permet d’afficher les différents réglages. L’utilisateur peut mettre l’intervalle du graphique qui lui correspond le plus par défaut il est à **[-5 ; +5]**.Les dimensions du graphique (intervalle) sont toujours proportionnelles entre eux.

Un bouton réinitialiser est présent dans le menu option. Il permet de remettre les dimensions du graphique par défaut **[-5 ; +5]**, ainsi que la précision de calcul et nettoie toutes les courbes présentes sur le graphique.

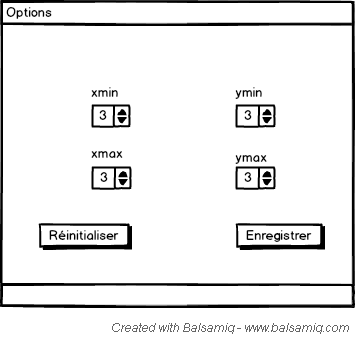
Pour effacer une fonction l’utilisateur peut aussi enlever les équations présentes dans les zones de texte.

Un zoom est présent en scrollant la molette de la souris.

## Interface



Ceci est la fenêtre principale, l’utilisateur peut écrire des équations dans les deux zones de saisies et l’affichage des courbes se fera en temps réel aucun bouton ne devra être activé. L’utilisateur peut aussi cocher la case à cocher qui fera passer l’application en mode paramétrique et les deux zones de saisies ne formeront qu’une équation. L’utilisateur devra rentré deux équation pour avoir un affichage en mode paramétrique.



Cette interface s’ouvre lorsque l’utilisateur appuie sur le bouton option dans la forme principale. Il peut modifier les intervalles du graphique et réinitialiser le graphique en appuyant sur le bouton *Réinitialiser* et une fenêtre d’avertissement apparaitra pour demanderune confirmation de l’action. L’ensemble des courbes sur le graphique seront effacer et les options sur les intervalles seront remises à zéro. Pour enregistrer les valeurs des nouveaux intervalles l’utilisateur doit cliquer sur le bouton enregistrer et le forme se rafraichira avec les nouvelles options.

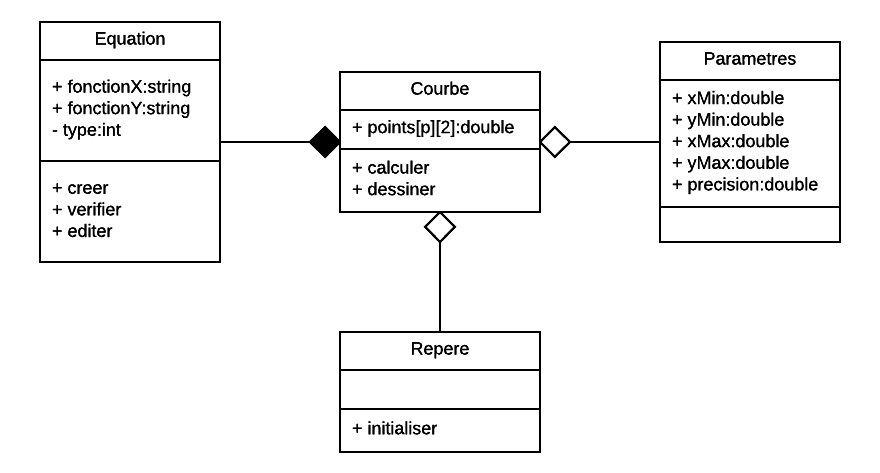
# Analyse organique

Pour élaborer mon projet, j’ai utilisé le logiciel de développement Visual Studio 2015, ayant appris le langage de programmation C# sur ce logiciel et programmé avec celui-ci pendant trois années. Durant lesquelles j’ai acquis beaucoup de connaissances sur ce programme.

Pour les sauvegardes, j’ai utilisé Git Hub et plusieurs stockages externes dont une clé USB et 4 disques durs externes. J’ai stocké mes données à plusieurs endroits pour être certain de ne pas risquer de perdre la totalité des mes données

Au niveau de l’organisation de mon code, j’ai utilisé une interface Modèle-Vue car une interface MVC modèle-vue-contrôleur aurait été obsolète pour ce type de projet.

## Architecture



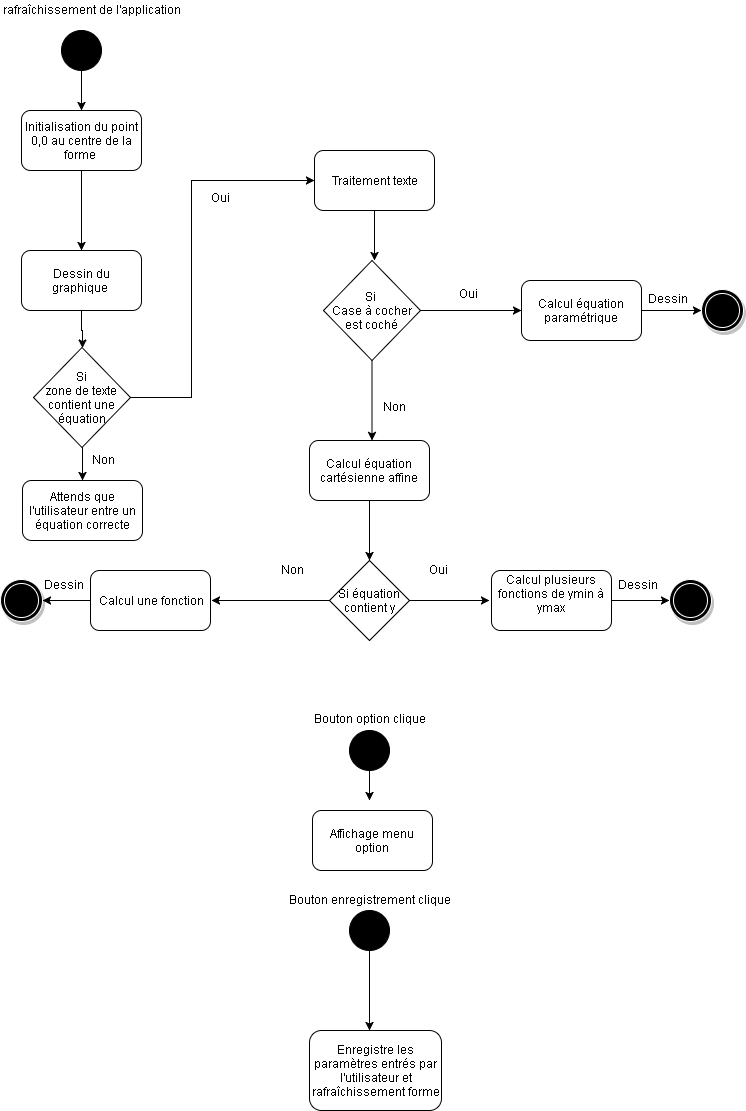
Un objet *Courbe* est créé pour chaque nouvelle courbe dessinée et est capable de se dessiner dans un contexte graphique passé en paramètre. Il contient les informations saisies par l’utilisateur sous la forme de deux objets : *Equation* et *Parametres*.

L’objet *Equation* est créé et modifié par l’objet *Editeur* (non représenté ici). L’objet *Parametres* a tous les attributs nécessaires pour pouvoir calculer les *p* points en *x, y* de la courbe avant de l’afficher. Ce nombre *p* de points dépendra de la taille de la zone affichée, de la précision demandée et du type de courbe (affine ou paramétrique).

L’objet *Repere* ne fait qu’initialiser le contexte graphique dans lequel la *Courbe* se dessine.

La librairie CSharp-Eval (<http://csharp-eval.com/HowTo.php>) est utilisée pour interpréter les équations saisies par l’utilisateur et calculer les points de la courbe dessinée.

## Structogramme



## Méthodes de réalisation

Pour la réalisation de mon code je me suis beaucoup inspiré d’un code sur internet : <http://csharphelper.com/blog/2015/06/graph-the-sine-cosine-and-tangent-functions-in-c/>

Ce code permet d’initialiser le point (0 ; 0) au centre de la forme et calcul la distance entre deux pixels que l’on peut diviser pour augmenter la précision des calculs.

Pour le calcul des points j’ai utilisé la librairie Mathparser que j’ai trouvé sur le site suivant : <http://www.bestcode.com/assets/docs/bcParserNET/>

La librairie Mathparser peut prendre une expression en string sans chiffre exemple : « sin(x) » et elle possède un paramètre qui permet de remplacé x par un chiffre. Elle retourne le résultat de l’expression qu’on lui a envoyé.

Calcul distance entre deux pixels :

//Hauteur et largeur du rectangle

float world\_width = (float)(Math.Abs(xmax - xmin));

float world\_height = (float)(Math.Abs(ymax - ymin));

//Position, Taille du Rectangle

RectangleF world\_coords = new RectangleF((float)xmin, (float)ymin,world\_width, world\_height);

PointF[] device\_coords = {new PointF(0, 0), new PointF(picGraph.ClientSize.Width, 0), new PointF(0, picGraph.ClientSize.Height)};

// Matrice de points proportionnelle au nombre de pixels affichés

gr.Transform = new Matrix (world\_coords, device\_coords);

Matrix inverse = gr.Transform;

inverse.Invert();

PointF [] pixel\_pts = {new PointF (0, 1), new PointF (1, 0)};

// Echantillonner en fonction de la taille de la fenêtre affichée

// Par exemple sur un écran 4K en plein écran, on aura 3800 pixels calculés

inverse.TransformPoints (pixel\_pts);

// Distance qui sépare deux pixels à l'écran en largeur

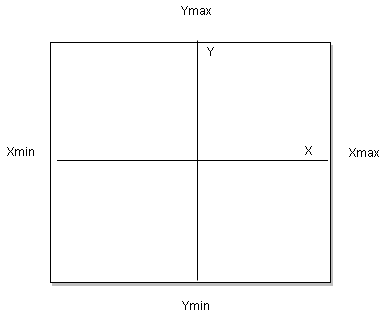
dx = Math.Abs (pixel\_pts [1].X - pixel\_pts [0].X);

La distance entre deux pixels est la variable la plus importante de mon program car elle permet le calcul du graphique et des courbes.

Class static Option :

Elle contient tous les paramètres pour la réalisation de chaque calcul.

Par exemple les dimensions du rectangle du rectangle



Class static Graphique :

Elle dessine l’axe des x et y avec la graduation. Elle reçoit en paramètre le minimum et le maximum du rectangle en x et en y. Avec ces paramètres elle part du minimum au maximum en un nombre qui correspond au nombre de graduation de l’axe

float dy = (float)((ymax - ymin) / 120.0);

double nbGradX = 1;

for (double x = xmin; x < xmax; x += nbGradX)

{

gr.DrawLine(graphPen, (float)x, -2 \* dy, (float)x, 2 \* dy);

}

La classe fait cela pour les deux axes.

Class TraitementTexte :

Elle reçoit l’équation en paramètre et elle permet de rajouter un « \* » dans une équation comme par exemple si elle reçoit 3x en paramètre elle va retourner 3\*x à la class Calcul.

Elle passe tout simplement le texte de l’équation en boucle et elle regarde si l’équation contient un x à côté d’un chiffre et si c’est le cas elle insert un \* entre les deux.

Class Calcul :

Cette class possède un constructeur qui reçoit en paramètre le texte traité par la class précédente (TraitementTexte) et le stocke ensuite dans une variable équation. Elle possède des méthodes qui permettent de faire les différents calculs pour le dessin des courbes. La méthode de calcul incrémente de xmin à xmax en ajoutant dx qu’elle a reçu en paramètre. Elle envoie chaque incrément à la librairie Mathparser qui retourne le résultat de chaque x. Pour finir elle rajoute chaque incrément avec son résultat dans une liste.

/// <summary>

/// Calcul tous les points d'une équation cartésienne affine

/// </summary>

/// <param name="xmin">bord gauche en x du rectangle/carré</param>

/// <param name="xmax">bord droit en x du rectangle/carré</param>

/// <param name="dx">distance entre deux pixels (plus le nombre est petit plus la précision est meilleure)</param>

/// <returns>Retourne une liste avec les points X et Y de la courbe</returns>

public List<PointF> PointXYEquation(double xmin, double xmax, float dx)

{

List<PointF> XY = new List<PointF>();

for (float x = (float)xmin; x <= xmax; x += dx)

{

float y = Convert.ToSingle(Calculate(Equation,x));

// vérifie si le résultat donne un nombre et est différent de zéro

if (!(double.IsNaN(y)) && (y != 0))

{

XY.Add(new PointF(x, -y));

}

}

return XY;

}

/// <summary>

/// CODE • http://www.bestcode.com/assets/docs/bcParserNET/

/// </summary>

/// <param name="formula">équation string qu'on doit calculer</param>

/// <returns>le résultat de l'équation donc le point Y</returns>

double Calculate(string equation,float x)

{

parser.Expression = equation;

parser.X = x;

return parser.ValueAsDouble;

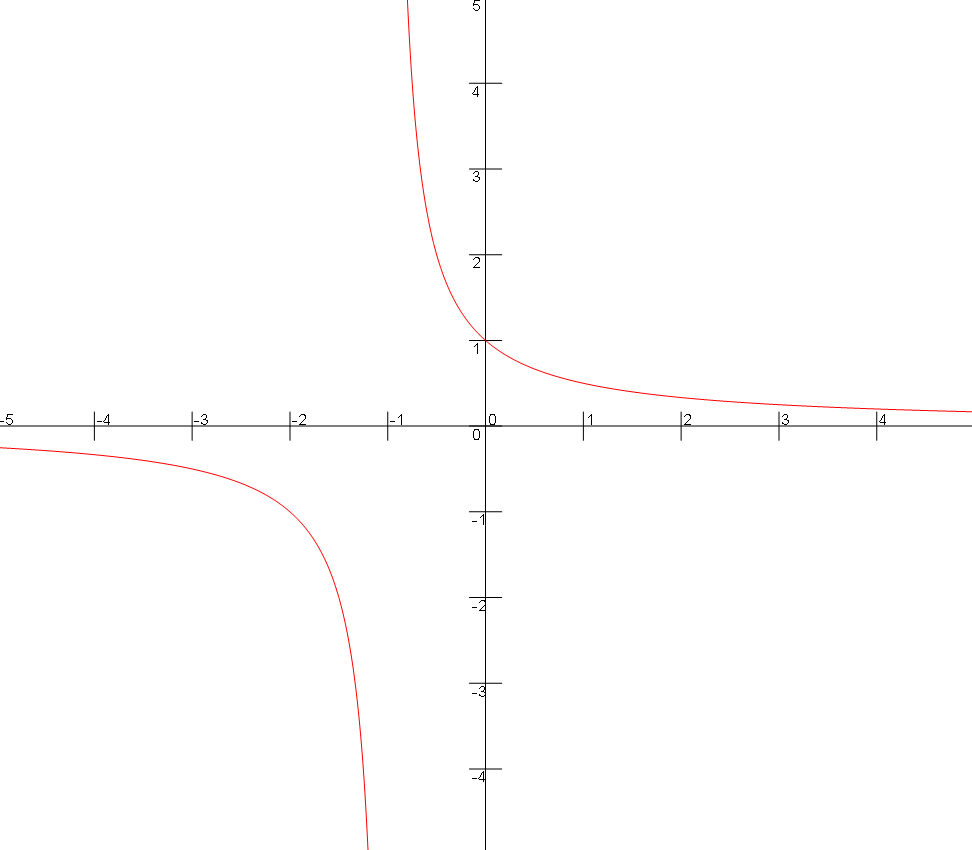
}

Class Dessin :

Cette class permet de dessiner l’ensemble des calculs fait par la class équation. Elle traite aussi ces listes contenant les calculs qui sont issu d’équation contenant des asymptotes. L’affichage d’une fonction qui contient des asymptotes n’est pas le même que l’affichage d’une équation sans asymptote. Une asymptote est une valeur impossible dans une équation. Par exemple dans la fonction suivante on aura une asymptote en car est une valeur impossible.

Voici un exemple de mon programme avec l’algorithme qui permet d’afficher les asymptotes et sans. On prend pour exemple la fonction donc l’asymptote se situe à la position

Résultat avec l’algorithme qui permet d’afficher correctement les asymptotes



Résultat sans l’algorithme en affichant tout simplement les points de la liste

## C:\Users\trifunovii_info\Desktop\Capture2.PNG

La fonction se dessin correctement mais au lieu d’avoir une asymptote à la position, il y a un trait. Ce trait est à la place de l’asymptote car la plus petite valeur y de la fonction se relie avec la plus grande valeur.

Ceci est du car j’utilise la ligne de code suivante qui est la meilleurs méthode pour dessiner une liste de points.

Graphics gr;

gr.DrawLines (pen, list.ToArray());

Cette ligne va relier le tableau de points dans l’ordre et vu que le point le plus négative est directement suivie par le point le plus positive. L’algorithme va les relier ensemble comme le reste des points.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x Entrée | -4 | -3 | -2 | -1 | 0 | 1 | 2 |
| f(x) Sortie | -0. | -0.5 | -1 | Erreur | 1 | 0.5 | 0. |

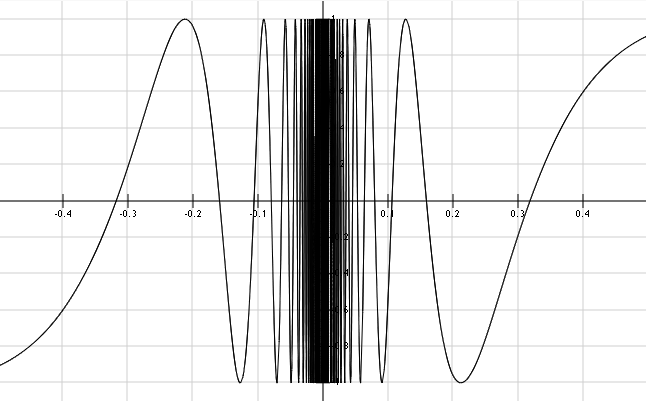
On voit avec le tableau ci-dessus que si on enlève le résultat de le point (-1 ; erreur), ce que l’algorithme fait comme on l’a vu avant. Le point qui suit le point (-2 ; -1) est (0 ; 1) donc l’algorithme va relier ces deux points alors qu’ils sont totalement à l’opposé

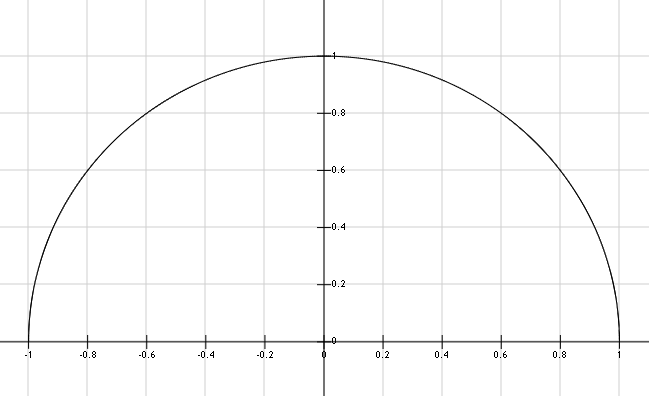
## Différente méthode de résolution :

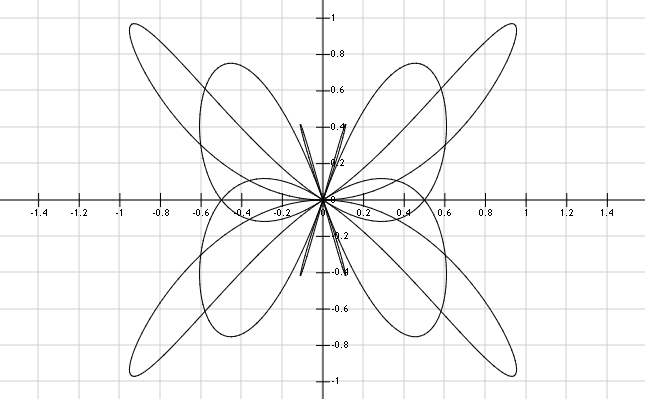
Durant le développement de mon programme j’ai rencontré plusieurs problèmes et pour résoudre ce genre de difficultés j’ai parfois utilisé plusieurs méthodes avant de trouver la plus optimale. Dans ce chapitre

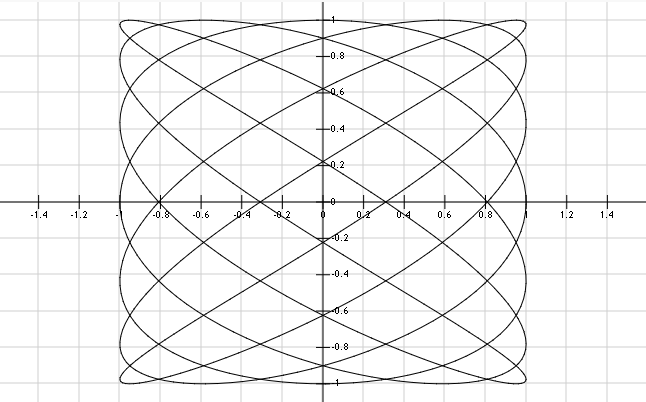
# Tests

## Pan de tests









|  |  |  |
| --- | --- | --- |
| N° | Scénario | Résultat Attendu |
| 1 | Cliquer sur le bouton réinitialiser dans le menu option | Une fenêtre d’alerte doit s’afficher et si l’utilisateur confirme son action le graphique est rems à zéro et l’ensemble des courbes sont effacer |
| 2 | Cliquer sur le bouton enregistrer après la modification de l’ensemble des options | La forme doit se rafraichir et le graphique doit changer en fonction des options initialisé par l’utilisateur |
| 3 | Cliquer sur la case à cocher | Le programme doit passer ne mode paramétrique et l’utilisateur doit écrire deux équation dans les zones de saisies pour avoir un résultat |
| 4 | Écrire une équation puis la supprimer | La courbe devrait s’afficher et ensuite s’effacer |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |

|  |
| --- |
| Planning |
| Voici les principales étapes du projet, pour un total de 80 heures de travails (10 jours x 8h). Les tests et la documentation seront réalisés en continue, en parallèle au développement.  Jour 1  Création des forms et autres éléments d’interface : l’application, le menu, la zone du graphe. Seules les vues sont réalisées : pas les actions.  Jour 2  Création des classes définissant la courbe : *Courbe*, *Equation* et *Parametres*. Implémentation minimaliste de la méthode *Courbe*.dessiner() avec une *Equation* de test simple codée en dur (style y(x)=x).    Jour 3 et 4  Ajout du champ de saisie de l’*Equation*. Implémentation des boutons d’action. Implémentation de la méthode *Courbe*.dessiner() avec l’appel à la librairie CSharp-Eval.    Jour 5 et 6  Implémentation de la méthode *Equation*.verifier(). Ajout dans la classe *Parametre* des calculs automatiques de la précision et des intervalles de valeurs.  Jour 7 et 8  Finalisation du code et tests.  Jour 9 et 10  Finalisation de la documentation. |

# Bibliographie

Wikipédia. Fonction (mathématiques) <https://fr.wikipedia.org/wiki/Fonction_(math%C3%A9matiques)>

Mathematiquesfaciles. Les fonctions. <http://www.mathematiquesfaciles.com/les-fonctions_2_74419.htm>

Apprendre-en-ligne. Etude de courbes paramétrées. https://www.apprendre-en-ligne.net/MADIMU2/ANALY/ANALY6.PDF

1. Exemple de grapheur : <https://www.graphsketch.com/> [↑](#footnote-ref-1)