



Haute école d'ingénierie et d'architecture Fribourg  
Hochschule für Technik und Architektur Freiburg

*Filière Informatique 2020-2021*

Intégration des nombres complexes et des Unum en Java avec  
COJAC

Classe I3

---

Cahier des charges

---

Travail de bachelor

Version : 1.0

Student : Cédric Tâche

Professor : Frédéric Bapst

Expert : Baptiste Wicht

## Table des versions

Version	Date	Auteur	Description
0.1	01.06.2021	Cédric Tâche	Création
1.0	07.06.2021	Cédric Tâche	Description plus précise du projet et des objectifs

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Contexte</b>	<b>1</b>
<b>3</b>	<b>Objectifs</b>	<b>1</b>
3.1	Intégration des nombres complexes . . . . .	2
3.2	Intégration des unums . . . . .	2
3.3	Démonstration des deux fonctionnalités . . . . .	2
<b>4</b>	<b>Objectifs secondaires</b>	<b>2</b>
4.1	Mise à jour des librairies . . . . .	2
4.2	Tests de performance . . . . .	2
<b>5</b>	<b>Tâches</b>	<b>3</b>
5.1	Intégration des nombres complexes . . . . .	3
5.2	Intégration des unum . . . . .	3
5.3	Démonstration . . . . .	3
5.4	Mise à jour des librairies . . . . .	4
5.5	Tests de performance . . . . .	4
<b>6</b>	<b>Planification</b>	<b>5</b>
<b>7</b>	<b>Références</b>	<b>6</b>

# 1 Introduction

La plupart des langages de programmation offrent des capacités similaires pour stocker des nombres et effectuer des calculs. Ces langages permettent d'utiliser des nombres entiers et des nombres réels.

Ces types ont tout de même des limitations. Les nombres entiers sont limités en taille et au domaine du réel. Quant à eux, les nombres à virgule flottante sont inexacts. Par conséquent, lorsque les erreurs s'accumulent, le résultat d'un calcul peut être très éloignée de la réponse exacte.

Pourtant, d'autres alternatives existent. Les nombres complexes sont couramment utilisés en mathématique et en physique. Ils peuvent être utilisés pour simplifier des réponses utilisant des racines de nombres négatifs ou pour combiner deux quantités réelles tels que la tension et l'intensité en électricité. Quant à eux, les nombres réels peuvent aussi être représentés différemment. Les **universal numbers** (**unums**) sont une représentation alternative possible dont la dernière version est expliquée dans l'article *Beating Floating Point at its Own Game : Posit Arithmetic* [2].

## 2 Contexte

En Java, aucun type ni aucune classe dans le JDK permet de gérer des nombres complexes ou des unum, mais il est possible de l'implémenter soi-même.

La première solution pour ajouter ces fonctionnalités et de créer de nouvelles classes : une classe pour les nombres complexes et une classe pour les unums. Ensuite, il faut écrire le code en utilisant directement ces classes. Pour changer le type de calculs effectué (ex : nombre complexe  $\rightarrow$  nombre réel), il faut changer le code source de l'application.

COJAC [1] est une librairie Java permettant de modifier les capacités arithmétiques d'un programme Java sans en modifier le code. Elle utilise l'API d'instrumentation et peut transformer les classes et méthodes au runtime pour changer le type de calcul effectué. Ainsi, pour changer le type de calculs effectué (ex : nombre complexe  $\rightarrow$  nombre réel), il faut seulement changer l'argument donné à COJAC [1] lors du démarrage de l'application. Ceci ne demande aucune modification dans l'application de l'utilisateur.

## 3 Objectifs

Le but de ce projet est d'ajouter deux nouvelles fonctionnalités à COJAC [1]. COJAC devra permettre de remplacer automatiquement les nombres à virgules flottantes par deux nouveaux types numériques.

### 3.1 Intégration des nombres complexes

Une option de COJAC permettra de changer le comportement des calculs dans l'application de l'utilisateur. Les nombres à virgules flottantes (*float* et *double*) seront remplacés, au runtime, par des nombres complexes. Les opérations arithmétiques telles que l'addition, la soustraction, etc. devront être adaptées. De plus, les méthodes souvent utilisées de la librairie standard devront également pouvoir fonctionner. Par exemple, la méthode *Math.sqrt* devra permettre de retourner la racine carrée d'un nombre négatif.

### 3.2 Intégration des unums

Une option de COJAC permettra de changer le format de stockage et de calculs des nombres réels. Les unums seront utilisés à la place de la virgule flottante. Par conséquent, les opérations arithmétiques devront être redéfinies pour fonctionner avec ce nouveau format de stockage. Il faudra probablement utiliser JNI pour accéder à une librairie C/C++ permettant d'utiliser les unums, mais d'autres approches restent possibles.

### 3.3 Démonstration des deux fonctionnalités

Des programmes de démonstrations seront réalisés pour montrer ces des deux fonctionnalités. Ces démonstrations doivent montrer l'utilité et les avantages de cette approche.

## 4 Objectifs secondaires

D'autres ajouts de fonctionnalités ou modifications permettraient d'améliorer ce projet.

### 4.1 Mise à jour des librairies

COJAC [1] utilise plusieurs librairies dont les versions sont désormais obsolètes. Il vaut mieux mettre à jour les versions avant de rencontrer des problèmes à cause de versions trop anciennes. Cependant, COJAC [1] devra garantir une compatibilité pour Java 8+.

### 4.2 Tests de performance

Lorsque les fonctionnalités de remplacement des nombres à virgule flottante par des nombres complexes et des unums, il restera encore un aspect inconnu qui est pourtant important pour décider de l'utilité de cette fonctionnalité : les performances. Pour cette raison, des tests de performance peuvent aussi être ajoutés pour tester l'efficacité de l'implémentation.

## 5 Tâches

Cette section détaille les tâches qui devront être effectuées pour réaliser chacun des objectifs définis précédemment.

### 5.1 Intégration des nombres complexes

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Analyse des implémentations existantes.
- Analyse de COJAC et de comment intégrer un nouveau type.
- Analyse des nombres complexes.
- Concevoir l'algorithme.
- Implémenter l'algorithme.
- Tester l'algorithme.

### 5.2 Intégration des unum

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Analyse des implémentations existantes.
- Analyse de COJAC et de comment intégrer un nouveau type.
- Analyse des unum.
- Concevoir l'algorithme.
- Implémenter l'algorithme.
- Tester l'algorithme.

### 5.3 Démonstration

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Imaginer une situation pour les démonstrations.
- Réaliser les démonstrations.

## **5.4 Mise à jour des librairies**

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Chercher les mises à jour de chaque librairie.
- Regarder les changements avec la nouvelle version.
- Modifier la version et adapter le code.
- Tester les modifications.

## **5.5 Tests de performance**

Les tâches suivantes seront effectuées pour réaliser cet objectif :

- Analyser les tests de performance actuels.
- Réaliser les tests de performance.

## 6 Planification

La figure 1 montre le diagramme de Gantt avec la liste des jalons et des tâches durant le projet. Ce travail dure du 31 mai au 16 juillet 2021.

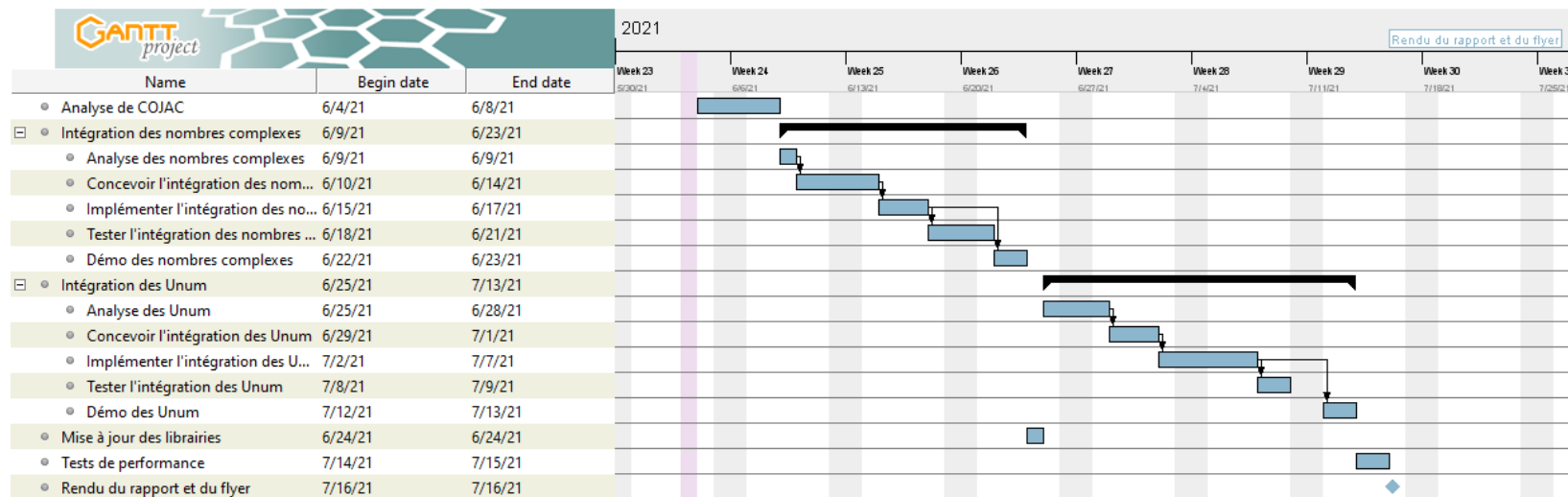


FIGURE 1 – Diagramme de Gantt

D'autres dates importantes sont également prévues plus tard dans l'année pour présenter ce projet.

Activité	Date limite
Rendu du poster	27.08.2021
Exposition du travail de bachelor	03.09.2021
Défense orale	06-08.06.2021



## 7 Références

- [1] Cojac - boosting arithmetic capabilities of java numbers. <https://github.com/Cojac/Cojac>, 2019.
- [2] Gustafson and Yonemoto. Beating floating point at its own game : Posit arithmetic. *Supercomput. Front. Innov. : Int. J.*, 4(2) :71–86, June 2017. ISSN 2409-6008. doi : 10.14529/jsfi170206. URL <https://doi.org/10.14529/jsfi170206>.