Perception, manipulation et Protection d’image

Marie Guénon / Jean-Dominique Favreau / Arnaud Tanguy

Compte rendu de TP3

Table des matières

[Description du sujet 2](#_Toc377484487)

[1. Affichage des chiffres 2](#_Toc377484488)

[2. Travail demandé 3](#_Toc377484489)

[Travail réalisé 5](#_Toc377484490)

[1. Initialisation des xi 5](#_Toc377484491)

[2. Constantes 6](#_Toc377484492)

[3. Initialisation des wi 6](#_Toc377484493)

[4. Descente de gradient 6](#_Toc377484494)

[5. Résultats 7](#_Toc377484495)

# Description du sujet

Le but de ce projet est de créer un algorithme automatique basé sur une descente de gradient et qui saurait dire si un chiffre affiché d’une manière particulière est pair ou impaire.

## Affichage des chiffres

Les chiffes sur lesquels nous allons travailler suivent un affichage et une mise en forme bien particulière. Globalement, on peut dire que l’affichage des chiffres que nous allons utiliser suit l’affichage classique des horloges numérique. Ce qui nous donne :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 |  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  | 6 |  | 7 |  | 8 |  | 9 |

Nous pouvons donc dire que chaque chiffre est constitué de 7 segments qui sont "allumé" (ici en gras) ou non et qui définit le chiffre affiché. Chaque segment a été numéroté comme suit :

|  |  |  |
| --- | --- | --- |
|  | 1 ↓ |  |
| 6 → | 7 ↓ | ← 2 |
| 5 → |  | ← 3 |
|  | ↑ 4 |  |

A partir de cette numérotation, on définit l’état de chaque segment de manière binaire : le segment est à 1 si il est "allumé", à 0 sinon. Puis concatène dans l’ordre l’état binaire de tous les segments. Ce qui nous donne :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1111110 | 0110000 | 1101101 | 1111001 | 0110011 | 1011011 | 1011111 | 1110000 | 1111111 | 1111011 |

## Travail demandé

Le but est de détecter de manière automatique si les chiffres formatés comme décrit ci-dessus sont pairs ou impairs grâce à un algorithme de perceptron. Celui-ci sera de la forme suivante :

o=1 si l’entrée est paire, 0 sinon

W1

W2

W3

W4

W5

W6

W7

W0

Où les xi sont les entrées correspondant à l’état des segments et x0 est fixé à 1.  
Les wi sont les poids attribués à chaque entrée et w0 est le coefficient synaptique.

La sortie o est calculée selon la formule suivante :

Soit c la sortie qui était attendue. A chaque itération, les wi sont recalculés suivant la formule :

Où ε est une constante influant la vitesse de convergence de notre algorithme.

# Travail réalisé

Le travail que nous avons réalisé a été codé parallèlement en Matlab et en C++11. Cependant, les méthodes implémentées étant les même et suivant une mise en forme très similaire, elles seront expliquées simultanément.

## Initialisation des xi

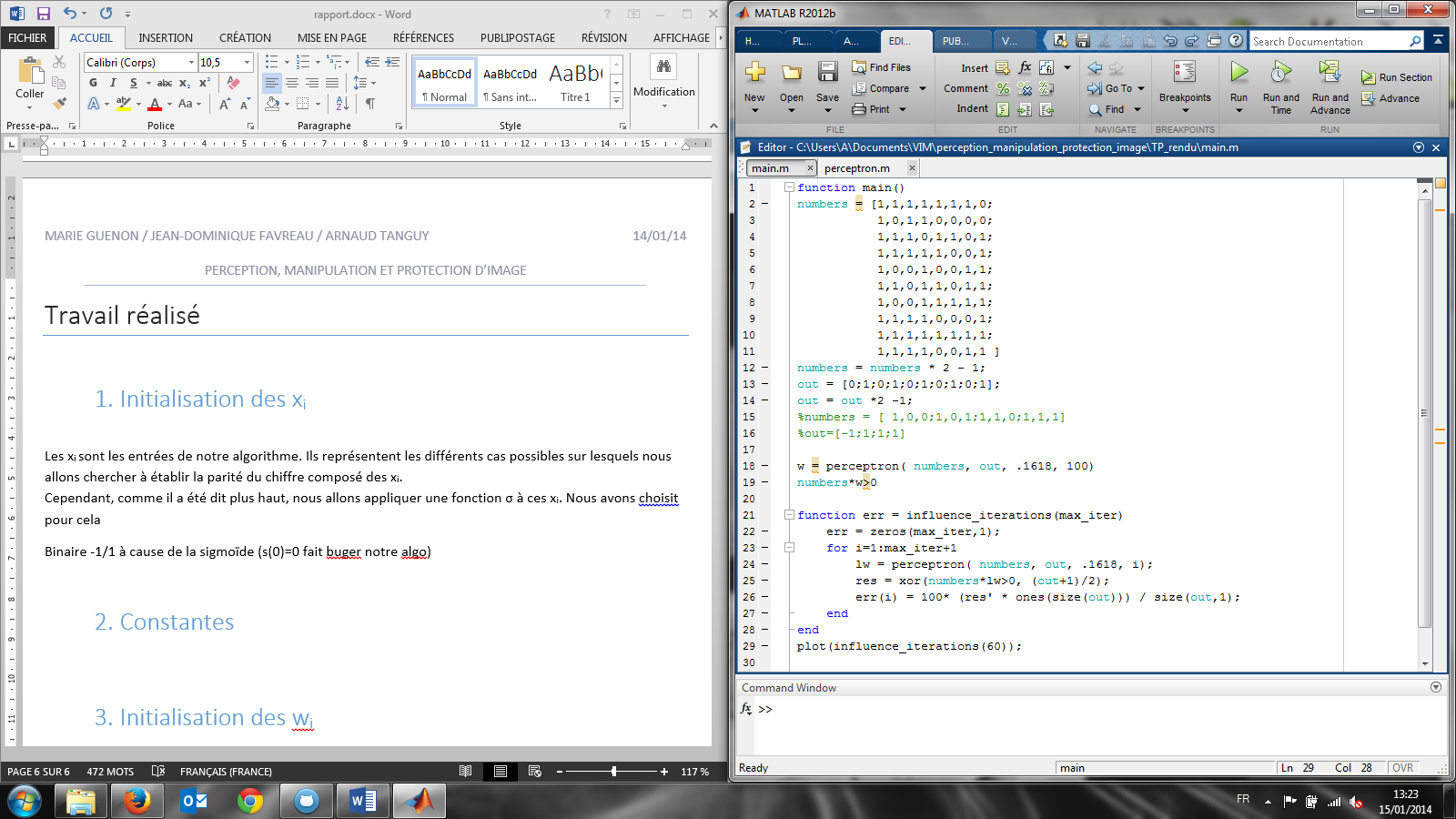
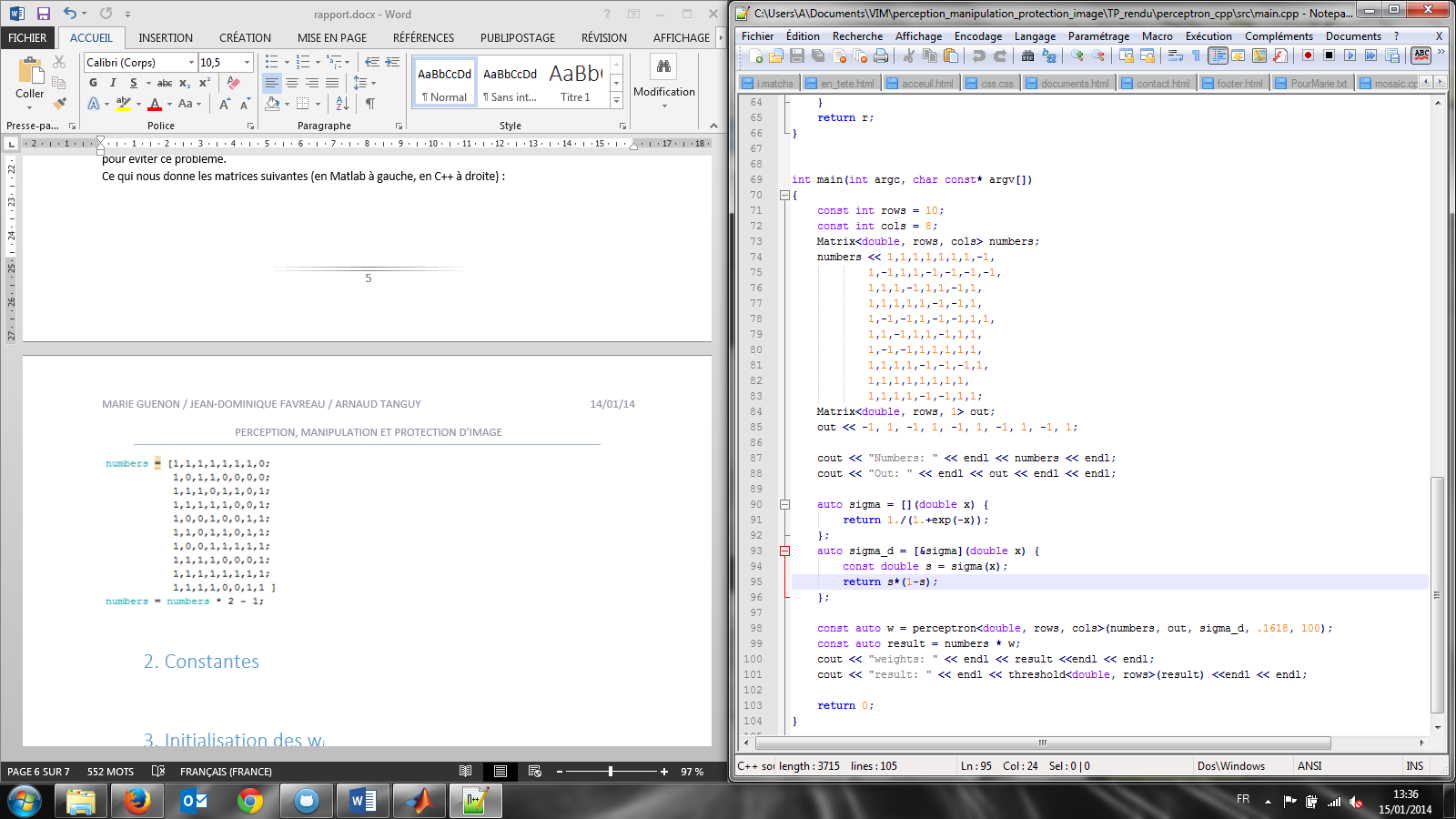
Les xi sont les entrées de notre algorithme. Ils représentent les différents cas possibles sur lesquels nous allons chercher à établir la parité du chiffre composé des xi. Nous avons choisi de réunir tous les xi dans une seule matrice

Cependant, comme il a été dit plus haut, nous allons appliquer une fonction σ à ces xi. Nous avons choisi pour cela d’utiliser une sigmoïde :

Et donc de dérivée :

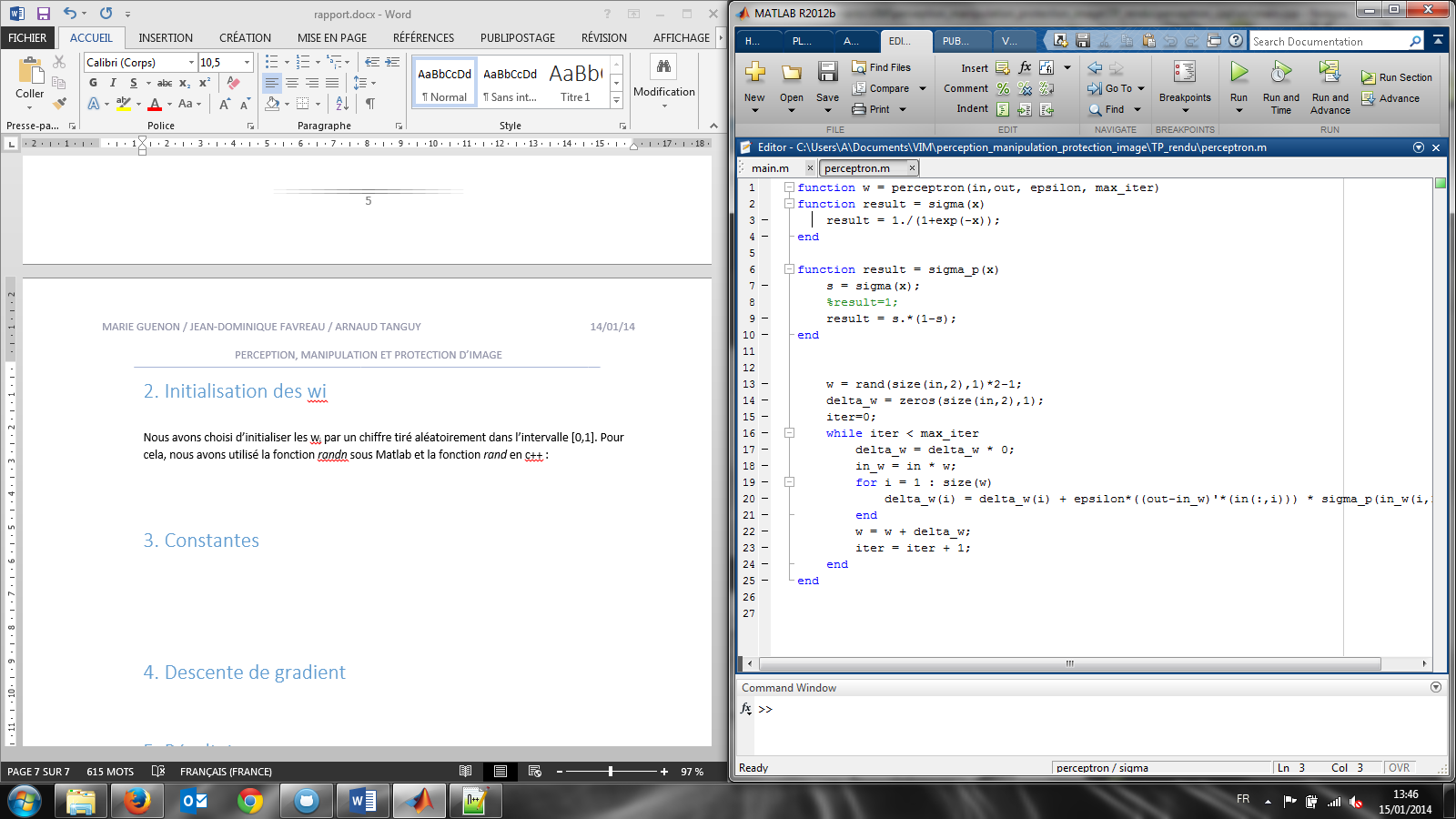
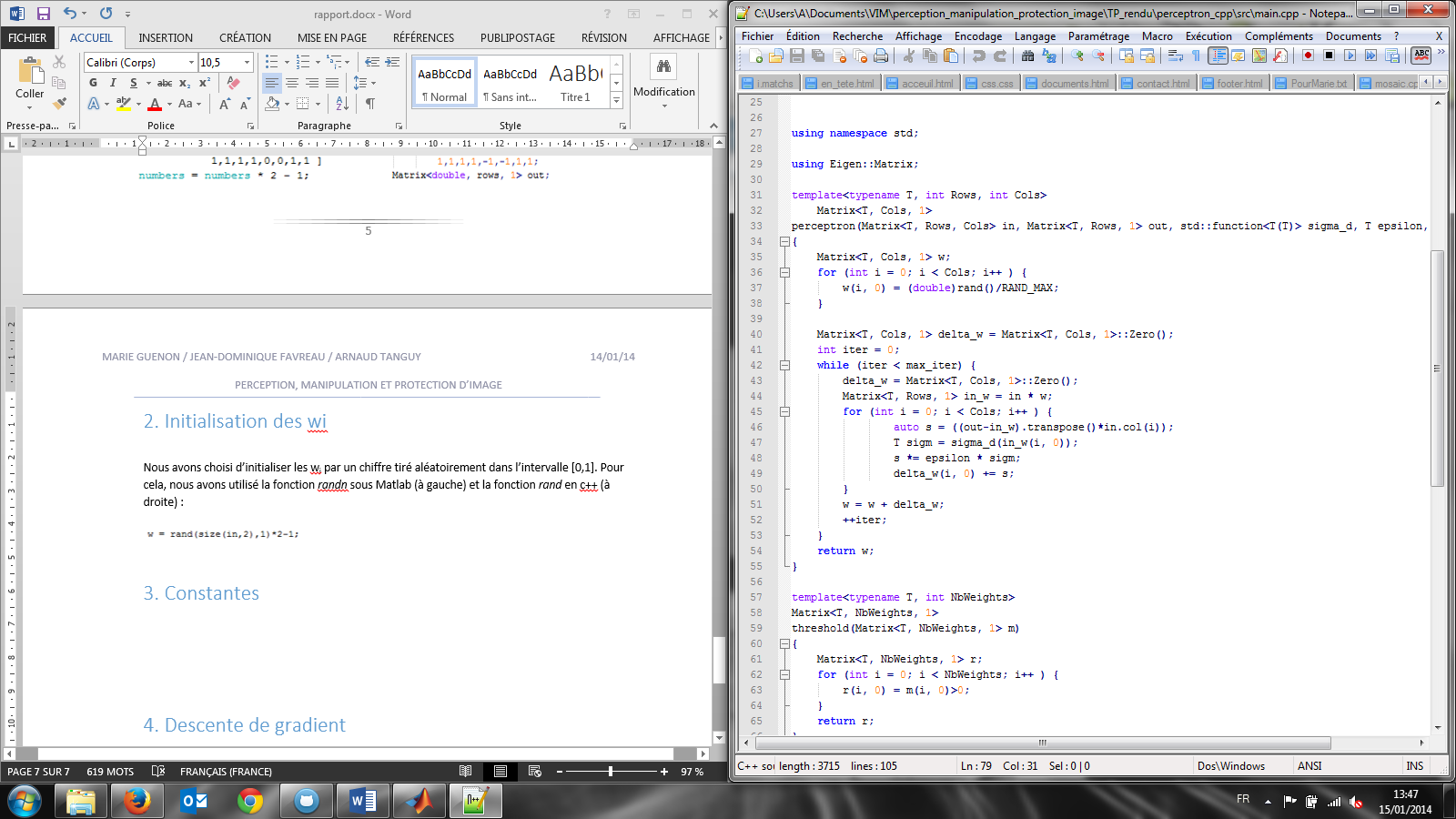
Or, cette dérivée s’annule en 0, empêchant la variation des poids si l’entrée est nulle, ce pose un problème évidant et non-négligeable. C’est pourquoi, nous avons choisi d’utiliser un code binaire -1/1 pour éviter ce problème.

De plus, nous avons choisi de mettre les chiffres à apprendre dans l’ordre croissant. Ce qui nous donne les matrices suivantes (en Matlab à gauche, en C++ à droite) :

## Initialisation des wi

Nous avons choisi d’initialiser les wi par un chiffre tiré aléatoirement dans l’intervalle [0,1]. Pour cela, nous avons utilisé la fonction *randn* sous Matlab (à gauche) et la fonction *rand* en c++ (à droite) :

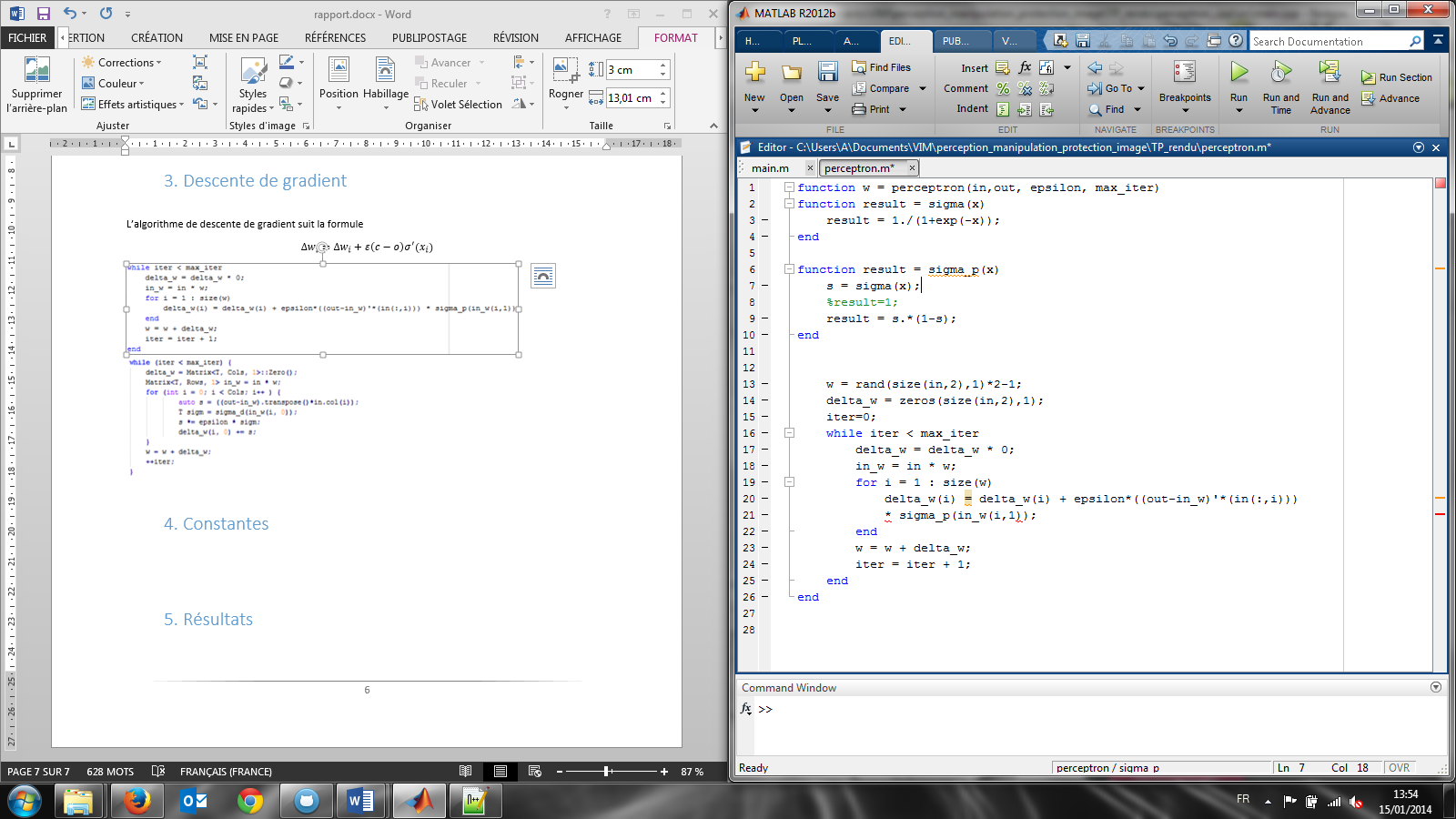
 

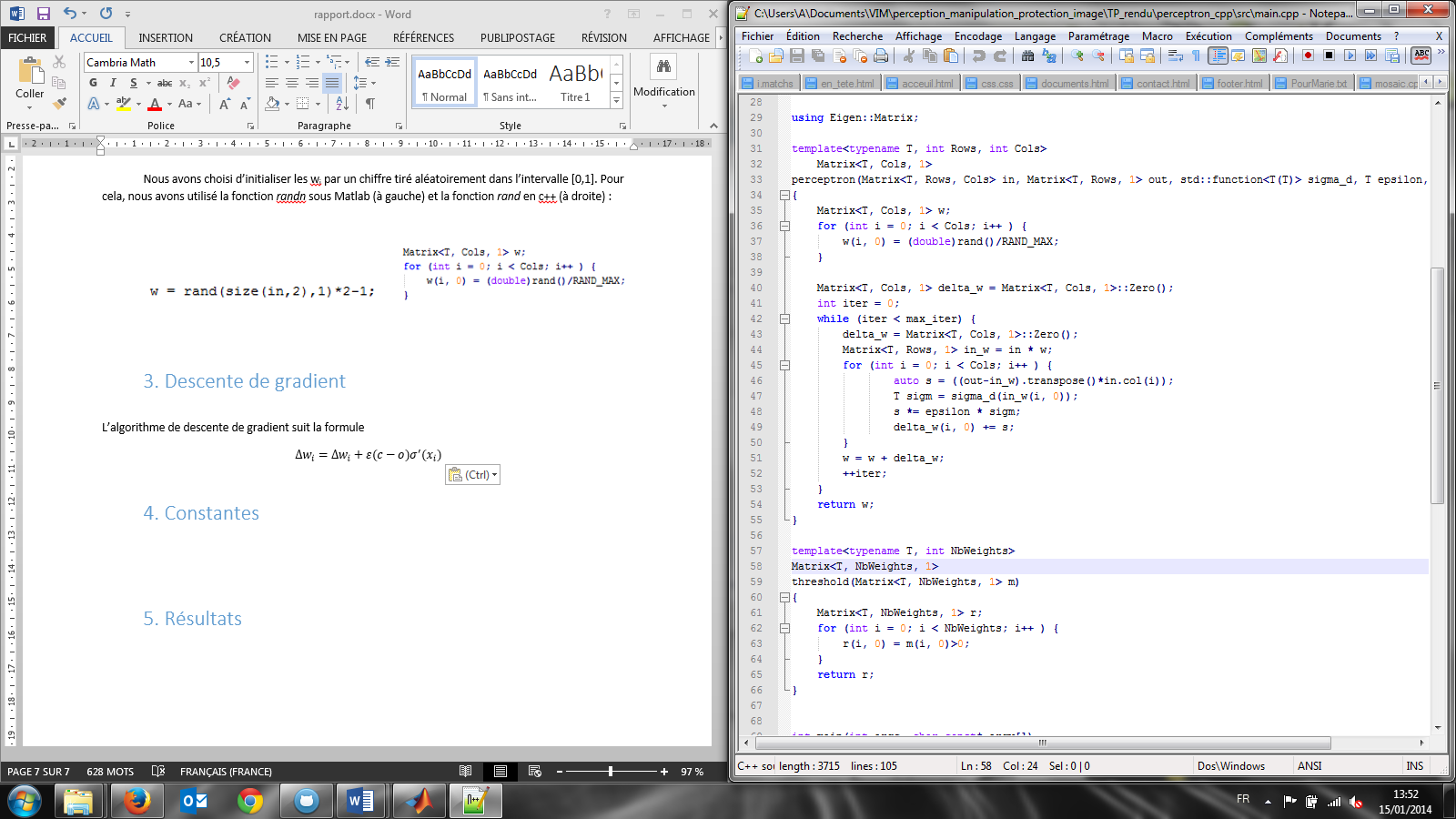
## Descente de gradient

L’algorithme de descente de gradient suit la formule :

Il s’agit ensuite d’itérer un certain nombre de fois (max\_iter) sur tous les poids wi pour les mettre à jour, sachant qu’au bout d’un nombre assez grand d’itérations, les wi devraient converger et se stabiliser.

Nous obtenons alors les algorithmes suivants, à gauche en Matlab et à droite en c++ :





## Constantes

Comme nous avons pu le voir plus haut, nous utilisons deux constantes : le nombre d’itérations et ε qui influence la vitesse de convergence de l’algorithme. Après quelques tests, nous avons choisi de prendre

et

En effet, cet ε permet une convergence assez rapide mais en évitant les oscillations du résultat. De même le *max\_iter* que nous avons choisi suffit à la convergence de l’algorithme. Cependant, pour plus de précision quant à ces choix, vous pouvez vous référer à notre analyse numérique des constantes ci-après.

## Résultats

### Etude de l’influence du nombre d’itérations

Pour faire cette étude, nous avons décidé de lancer plusieurs fois de suite notre algorithme suivant un nombre croissant d’itérations et de calculer de manière systématique la "distance" entre le résultat obtenu et le résultat attendue. La distance a été calculée suivant la formule :

Ce qui nous donne le résultat suivant :

Nous pouvons constater (surtout sur la courbe de tendance, en rouge ici) que l’augmentation du nombre d’itération fait clairement baisser le pourcentage d’erreur dans le résultat. De plus, nous pouvons dire ici qu’à partir d’une quarantaine d’itérations, nous obtenons une solution stable qui n’est plus modifiée par la suite. Il est donc suffisant de prendre un maximum d’itération à 50, pour garder une marge d’erreur dans les cas les plus défavorables.

### Etude de l’influence de ε