CentraleSupélec

Projet ST7 : Protection des réseaux

Ines Ben Cherifa
Alexian Helaine
Erwan Tissot
Terrence Grandchamps
Théophile Blanpain
Maxime Worms

2A
Cursus Ingénieur
2024-2025

Table des matières

Table des figures

1 Motivation du projet

1.1 Contexte

La protection des réseaux est un enjeu crucial de nos jours. Prenons l'exemple des lignes à haute tension en France. Celles-ci peuvent présenter, ponctuellement, ce que nous appellerons par la suite un défaut, par exemple, un arbre qui tombe sur une ligne. Lorsqu'il y a présence d'un défaut sur une ligne, il est nécessaire de couper celle-ci dans les plus brefs délais afin de ne pas générer de complications. L'arrêt d'une ligne n'est pas gênant dans la mesure où le réseau assure une sécurité N-1 : le réseau est toujours opérationnel lorsque l'on enlève une de ses lignes. Au-delà d'une ligne coupée, l'intégrité du réseau n'est plus assurée. Il est nécessaire de garantir cette sécurité pour ne pas entraîner de surtensions qui pourraient, à terme, causer un *blackout*.

Il est donc primordial de savoir détecter les défauts rapidement et précisément. Pour cela, des transformateurs sont placés à chaque relais pour mesurer tension et courant. L'impédance au relais en est déduite et si elle diffère de l'impédance attendue à cause d'un défaut sur la ligne, un disjoncteur la coupe.

Cependant, deux problèmes principaux nous empêchent de couper une ligne à la moindre variation d'impédance :

— le cas où la variation d'impédance est due au fonctionnement d'un appareil branché sur le réseau (variations minimes en général mais non négligeables parfois (par exemple avec le démarrage d'un train)) ;

— le cas où l'impédance mesurée appartient à une zone d'incertitude où l'on n'est pas bien certain que le défaut soit effectivement sur la première ligne ou sur la suivante : dans ce cas, il est nécessaire de laisser passer un certain délai pour obtenir la confirmation.

Il est donc absolument nécessaire de considérer ces cas-là pour ne pas couper des lignes intempestivement et mettre en péril l'intégrité du réseau.

Ce projet aura donc pour but de définir des zones qui permettent

de réagir judicieusement face à une impédance mesurée à un relais. Les trois zones à établir seront appelées par la suite :

- **Zone 1** : on coupe immédiatement la ligne (car le défaut est dessus) ;
- **Zone 2** : on coupe après quelques millisecondes d'attente (si le défaut est sur la ligne 2, le relais de la ligne deux l'aura coupé pendant la période d'attente, sinon on le coupera) ;
- **Zone 3** : on ignore (la zone correspond au reste du plan (R, X)).

Cette méthode a pour objectifs de couper rapidement une ligne qui présente un défaut sans couper celles qui n'en présentent pas.

1.2 Solution actuelle

On pourrait donc penser qu'il suffirait de simuler un grand nombre de défauts sur la première et la deuxième ligne à l'aide d'une méthode de Monte Carlo par exemple et définir ces zones comme l'enveloppe des points générés. Malheureusement, si ces zones sont les meilleures d'un point de vue théorique, elles ne peuvent être appliquées concrètement car elles présentent des pointes indésirables.

Actuellement donc, on sait travailler à partir de polygones que l'on cherche à optimiser de manière à obtenir une matrice de confusion proche de l'identité tout en cherchant à obtenir un polygone final convexe (en minimisant son volume notamment).

Voici des exemples de zones obtenues par cette méthode (où ω représente le poids que l'on accorde à la convexité) :

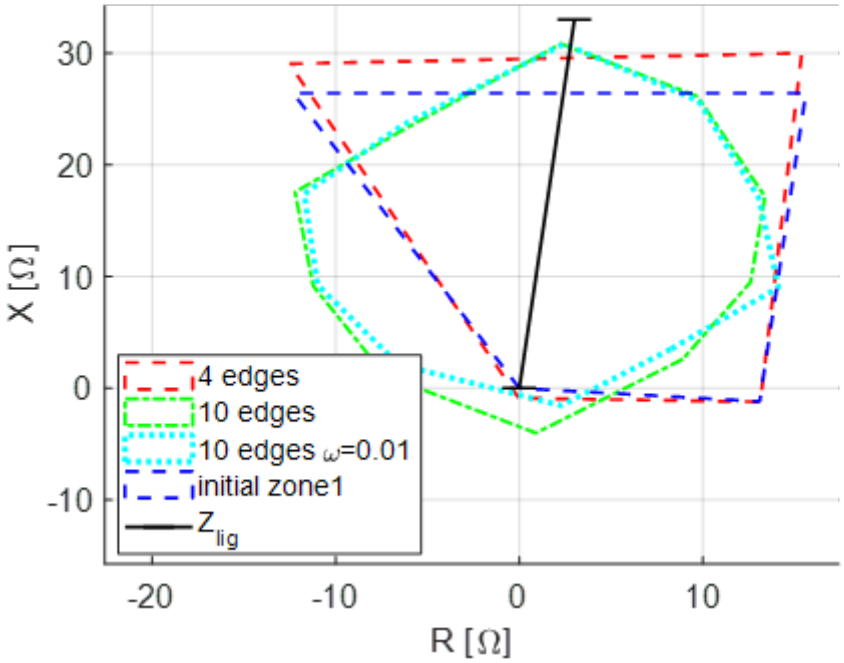


FIGURE 1 – Modélisation actuelle de la zone 1

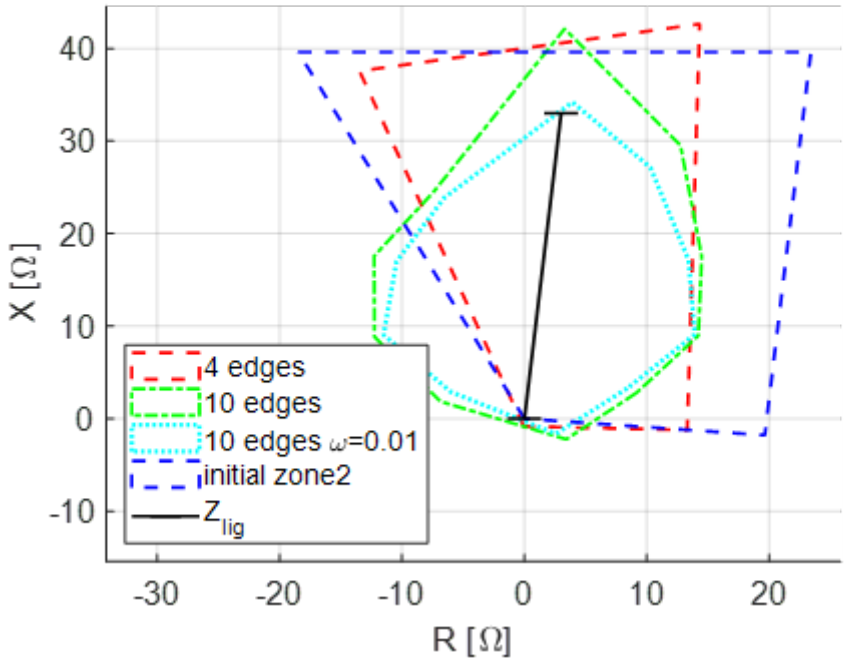


FIGURE 2 – Modélisation actuelle de la zone 2

2 Modélisation mathématique

2.1 Définition du problème d'optimisation

On se place dans le plan Réactance-Résistance. Dans la première partie de ce rapport, nous allons voir comment obtenir les densités de probabilités caractérisant les défauts pouvant se produire sur les lignes une et deux.
Dans la partie optimisation, nous chercherons les formes optimales des zones de coupe instantanée (zone 1) et de coupe retardée (zone 2).
Pour ce faire, nous définissons les fonctions qui vont guider notre optimisation.

2.2 Mise en équation du problème

Notons f_1 la densité de probabilité des défauts sur la ligne 1 et f_2 celle sur la ligne 2.

Il s'agit d'un problème de détection, nous définissons donc les quatre coefficients de la matrice de confusion.

Vrai Positif :

$$VP = \int_{Z_1} f_1 \, dA$$

Que nous cherchons à maximiser.

Faux Négatif :

$$FN = \int_{Z_2} f_1 \, dA$$

Que nous cherchons à minimiser.

Faux Positif :

$$FP = \int_{Z_1} f_2 \, dA$$

Que nous cherchons à minimiser.

Vrai Négatif :

$$VN = \int_{Z_2} f_2 \, dA$$

Que nous cherchons à maximiser.

Contraintes :

De plus, nous aimerions imposer certaines contraintes à notre problème. En effet, il faudrait que les zones obtenues se rapprochent de formes convexes. Ainsi, une idée pourrait être de vouloir minimiser le périmètre ou encore de minimiser le rapport périmètre sur le volume.

Forme du problème

Dans toute la partie optimisation, nous nous concentrerons sur l'optimisation de la frontière entre les deux zones. Nous fixerons donc les deux points aux extrémités de chaque zone.
La directive principale pour l'optimisation est que le taux de faux positif soit très proche de zéro. En effet, on veut absolument ne pas couper la ligne deux avant que le relais de ladite ligne ait eu le temps d'intervenir, on perdrait la garantie sûreté N-1.

3 Étude des densités de probabilité

3.1 Modèle électrique

Commençons par modéliser l'apparition d'un défaut sur les lignes du réseau de manière probabiliste.

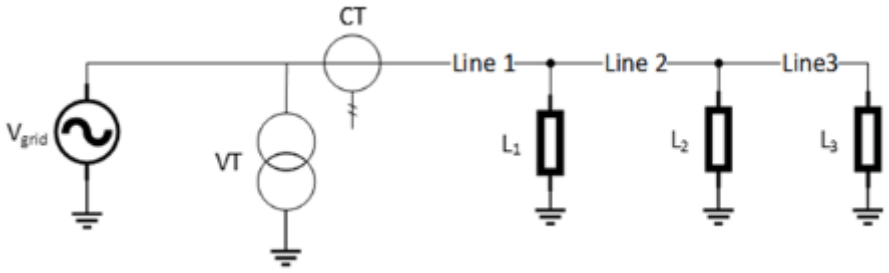


FIGURE 3 – Modélisation des lignes réseaux

Les lignes du réseau électriques sont modélisées par le circuit ci-dessus auquel on va ajouter une impédance de défaut placée aléatoirement. Pour chaque ligne, on place le défaut selon une distribution uniforme et on définit également une incertitude sur les impédances distribuées selon une loi normale. Ensuite, grâce à une mesure de la tension et du courant qui traverse le circuit on peut remonter à l'impédance totale de ce dernier. Enfin, on effectue cette expérience un grand nombre de fois pour chaque ligne afin d'avoir une base de données de résultats pour ensuite travailler sur l'optimisation car il est difficile d'avoir un nombre représentatif de données réelles.

3.2 Étude théorique du circuit

Le schéma électrique représentant le réseau est composé à l'entrée d'un modèle de Thévenin avec un générateur de tension E, une résistance R_{cc} et une réactance X_{cc}, de trois lignes de tension en parallèle d'impédance Z_{ligne} et dont les notations $R_{1,2,3}$, $X_{1,2,3}$ et $C_{1,2,3}$ sont associées à la ligne correspondante, ainsi que d'une charge en sortie du circuit de résistance R_L et de réactance X_L. Le défaut quant à lui est défini par la ligne sur laquelle il se trouve (choisie à l'avance), par sa

résistance R_{def} et sa position sur la ligne notée $posdef \in [0;1]$ selon la proportion de ligne avant défaut ($posdef$ %) ou après défaut ($1-posdef$ %).

Maintenant, on calcule l'impédance équivalente du circuit en partant de la charge et en remontant chaque ligne jusqu'à la source, sachant que les impédances des lignes vont être affectées par des incertitudes distribuées selon une loi normale définie au préalable et le défaut va fractionner la ligne en 2 parties selon sa position ce qui fera varier l'impédance équivalente.

Le programme Matlab disponible en annexe (7.1.1) procède ainsi au calcul de Z_{tot}, l'impédance équivalente, en suivant la structure récursive expliquée précédemment en simplifiant les mises en parallèles et en série successives et en remontant les lignes une par une. On s'épargnera l'expression analytique complète de la solution, trop complexe pour être pertinente.

3.3 Construction des densités de probabilité

L'objectif est donc de construire des densités de probabilité 3D. Elles correspondent à la probabilité d'avoir une valeur R de résistance et X de réactance lorsqu'un défaut survient sur une des 2 lignes électriques étudiées. Elles serviront à faire fonctionner les algorithmes d'optimisation (stochastiques, déterministes).

3.3.1 Simulation de Monte Carlo

Pour construire les densités de probabilité nous avons réalisé une simulation de Monte Carlo qui calcule l'impédance apparente lorsqu'un défaut survient. Plus précisément, pour une ligne et un défaut choisi, le programme simule N_{MC} fois Z_{app}. Pour cela nous utilisons la fonction "calcul tension courant relai ddp" qui permet de calculer la tension et l'intensité aux bornes du relai et donc d'obtenir Z_{app}.

Voici l'algorithme utilisé en pseudo-code :

Algorithm 1 Algorithme de Monte Carlo avec incertitudes

```
for i = 1 to N_MC do
    (Vmes, Imes) ← calcul_tension_courant_relai_ddp(E, Xcc, Rcc, Zlig, Clig, RL, XL, pdf_R, pdf_X, pdf_C, lignedef, posdef, omega, Rdef)
    Vmes ← |Vmes|× random(pdf_V) × exp(j × (angle(Vmes) + random(pdf_phiV)))
    Imes ← |Imes|× random(pdf_I) × exp(j × (angle(Imes) + random(pdf_phiI)))
    Zapp(i) ← Vmes / Imes
end for
```

Cet algorithme fonctionne pour une ligne et une position de défaut fixée. Nous devons le répéter pour plusieurs lignes et plusieurs positions de défaut. Nous avons pour cela construit une variable $plan\_exp$ qui définit l'ordre dans lequel l'algorithme est appliqué à chaque ligne et chaque position de défaut. Nous ne considérons ici que les deux premières lignes et nous choisissons aléatoirement N_{pos} positions de défaut sur chaque ligne. Nous remarquons par ailleurs que l'utilisation des boucles "for" va de paire avec une complexité spatiale très importante rendant très longue l'exécution du programme. Nous avons donc choisi de vectoriser ce programme afin de gagner en vitesse d'exécution et de pouvoir simuler plus de valeurs bien que nous soyons conscient que cela augmente la complexité spatiale.
Le code Matlab est disponible en annexe.

3.3.2 Interpolation permettant de passer d'un histogramme à une densité de probabilité

Nous avons donc obtenu des histogrammes 3D à partir de Z1 et Z2 dont l'abscisse et l'ordonnée sont la résistance R et la réactance X obtenues à partir de Z_{app}. Nous devons maintenant convertir ces histogrammes en densités de probabilité exploitable pour les algorithmes d'optimisation. Pour cela nous avons utilisé une méthode d'interpolation de l'histogramme sur un quadrillage plus fin afin d'avoir des densités "continues" que nous avons finalement normalisées en divisant la

hauteur de chaque pic par le volume total de l'enveloppe de la densité de sorte que celui-ci soit égal à 1.

Malheureusement, cette méthode n'est pas satisfaisante du fait de la présence de quelques artefacts dans les histogrammes qui ne suivent pas la tendance général de celui-ci. Ces artefacts proviennent du nombre insuffisant d'expérience réalisé. Nous pouvons difficilement corriger ce problème en augmentant ce nombre sans avoir plus de puissance de calcul. Pour palier à ce problème, nous avons plutôt choisit d'utiliser une méthode de lissage qui fonctionne sur le principe des masques de convolution en traitement d'image. Cela permet de supprimer ces artefacts et de fournir des densités plus exploitables pour l'optimisation. Nous sommes néanmoins conscient que la convolution entraîne une légère modification des lignes de niveau des densités de probabilité.

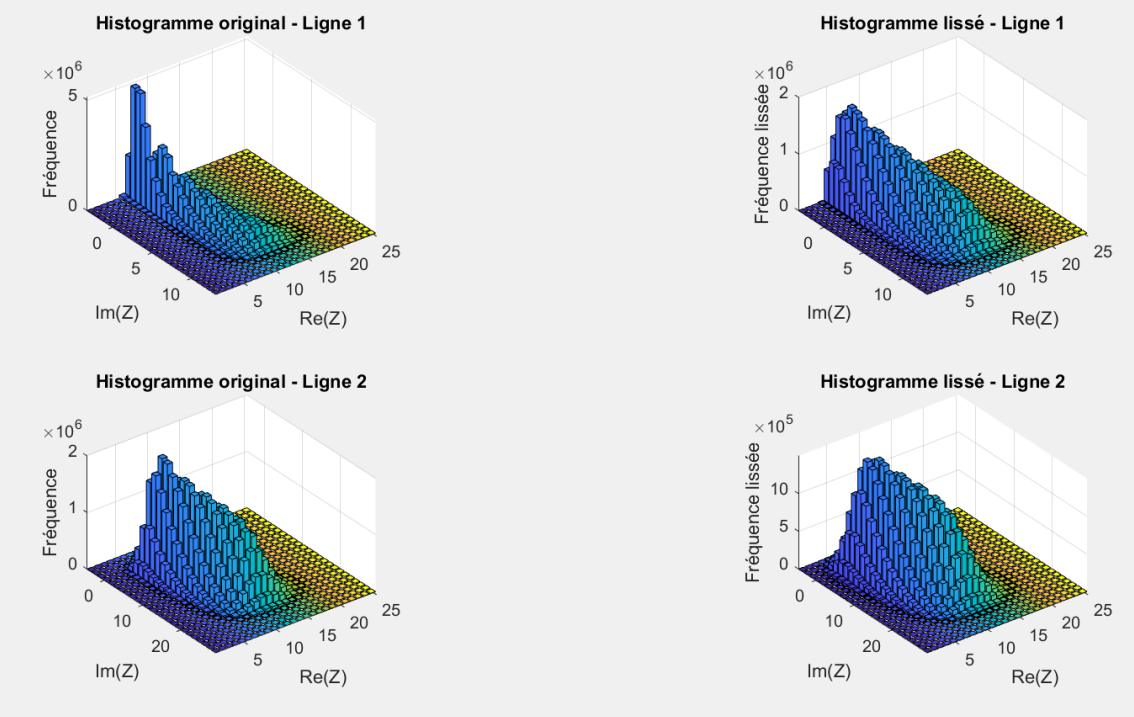
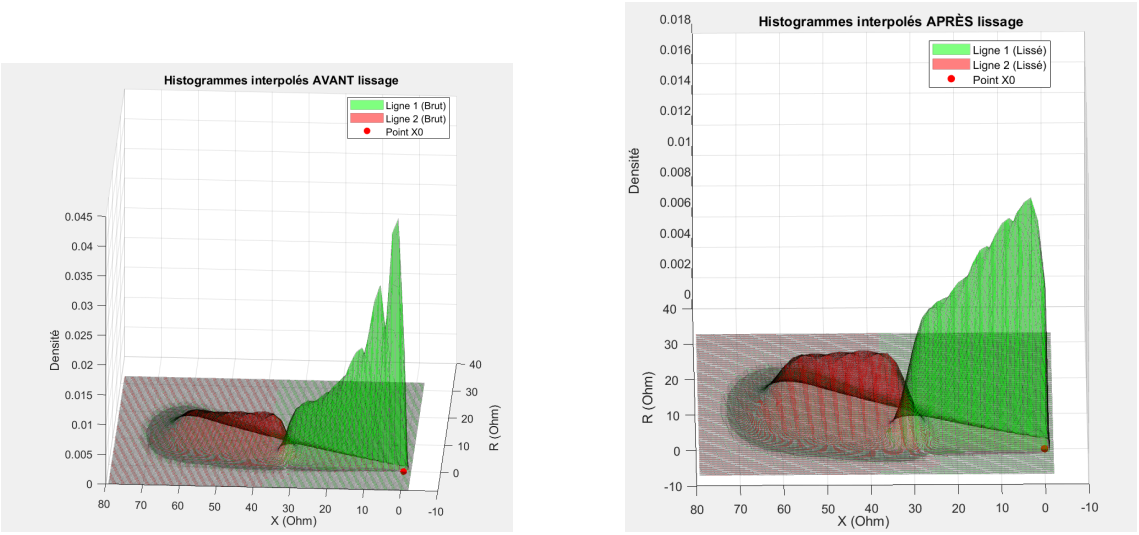


FIGURE 4 – Masque de convolution appliqué aux histogrammes



(a) Densités brutes

(b) Densités après lissage

FIGURE 5 – Comparaison "avant/après" la méthode de lissage

Ces densités ont été obtenues en simulant 10000 défauts sur chacune des 2 lignes soit 20000 positions de défauts différentes. Pour chaque défaut 10000 valeurs de Z_{app} sont simulées.

4 Optimisation du code et temps de calcul

Pour la suite nous noterons (N_Mc, N_pos) respectivement le nombre de simulation de Monte Carlo et le nombre de points de défauts sur les lignes. Par exemple une simulation (10,30) signifie que le nombre de simulation de Monte Carlo est de 10 et le nombre de points de défauts 30. On s'intéresseras aussi au nombre de points total : (N_Mc, N_pos) -> N_mc * N_pos. Enfin k,M,G représentent les unités mathématiques usuelles

Avec une implémentation naive (boucle for) notre algorithme avait, pour une simulation (100,200) un temps de calcul de 170 secondes. Pour référence l'ordinateur utilisé avait un Dual-core.

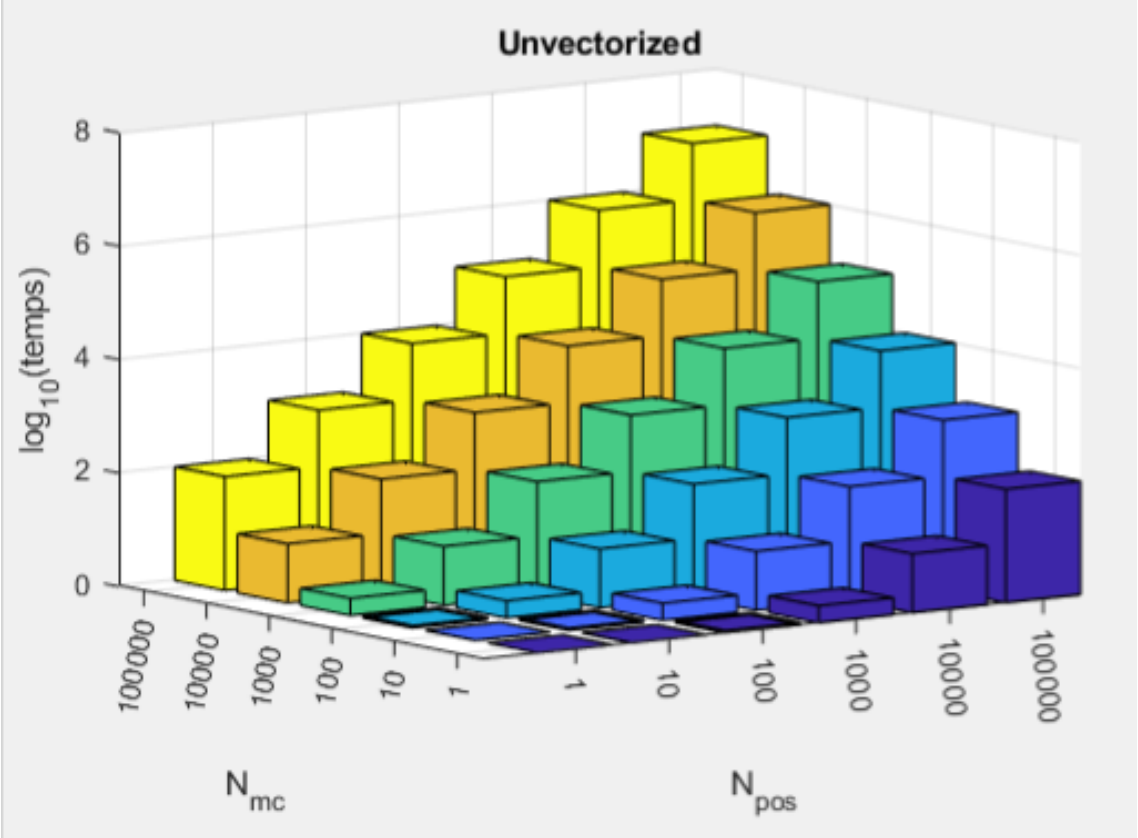


FIGURE 6 – Temps de calcul d'une simulation (x,y)

Sur le même ordinateur, la vectorisation du code a permis de passer d'un temps de calcul de l'ordre de 170 secondes à 0,3 secondes. Nous permettant notamment de passer à des simulations (10k, 20k) pour un temps de 40 secondes seulement.

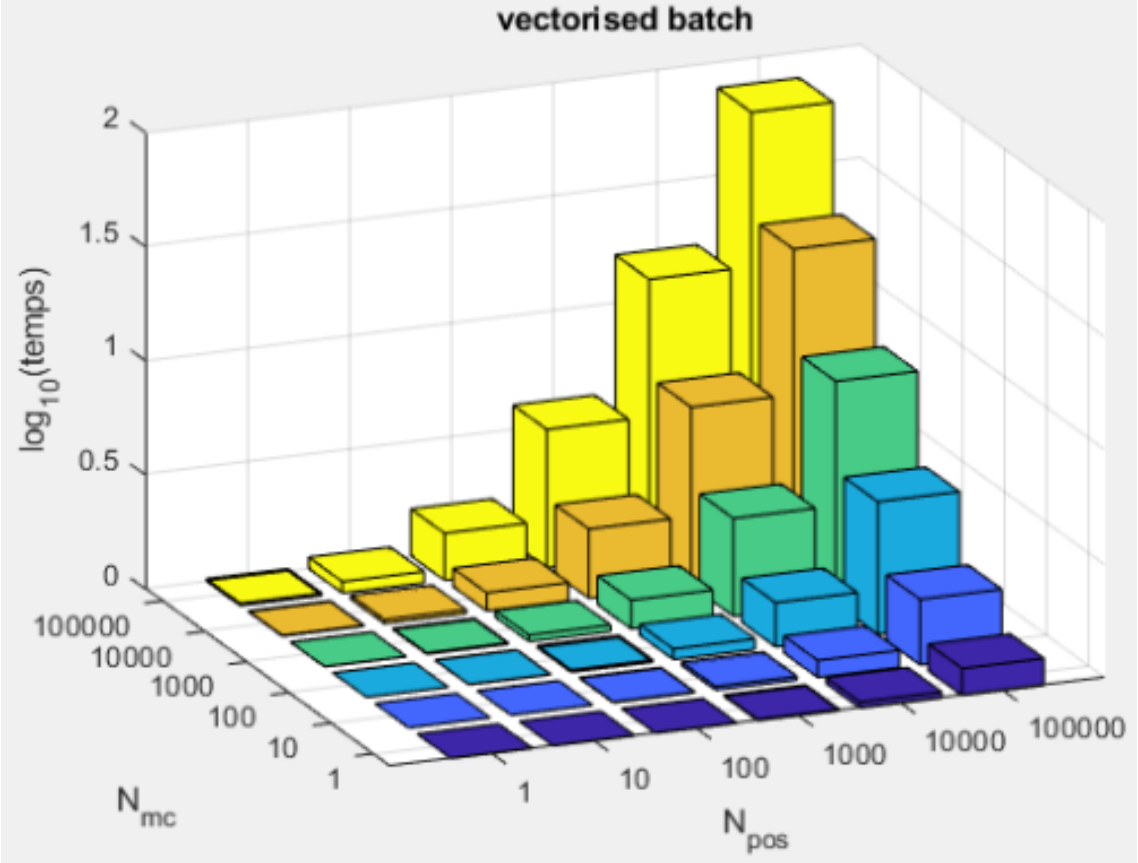


FIGURE 7 – Temps de calcul d'une simulation (x,y) vectorisé

Note : Les figures précédentes ne relèves pas de simulations réelles mais d'extrapolation du temps de calcul nécessaire pour des valeurs élevés de $N_pos et N_mcen supposant quel'volution pour de grande valeur soit sensiblement la$

4.1 le coût de cette vectorisation

Cette vectorisation ne vient pas sans un prix, celui de l'espace de calcul. De fait, un ordinateur possède une mémoire vive attribué aux tâches telle le calcul vectorisé de nos simulations. On a observé empiriquement une surchagre lorsque le produit N_mc * N_pos dépassait les 200 millions de points. Ainsi est née l'idée d'une optimisation personnalisée du code.
Le principe est simple, on créer des vecteurs de tailles quantifiée $N_vect quel'on appellera u$ $N\_mc ne surcharge pas la RAM del'ordinateur (valeur trouver empiriquement). L'alg$

Il est intressant de noter qu'un tel algorithme rduit plus efficacement le temps de calcul o
N_pos dpasse la capacit de l'ordinateur, et que les valeurs aiune relle diffrence d'ordre d
Cela conclut notre travail sur la production des densits de probabilit ncessaires l'optimis

5 Implémentation d'un algorithme déterministe

Dans cette partie, nous allons implémenter une méthode de résolution déterministe par descente de gradient pour résoudre le problème d'optimisation.

5.1 Définition du problème d'optimisation

Nous allons optimiser la position des points de la frontières des deux polygones pour minimiser la fonction objectif suivante.
Nous décidons de fixer les abscisses des points de la frontière.

Fonction Objectif

On va utiliser plusieurs paramètres de pondération pour modérer l'optimisation par rapport à certaines fonction.
Même si nos composantes principales de la fonction objectif sont Vrais Positifs et Faux Positifs, nous trouvons qu'il est nécessaire de prendre en compte tous les coefficients de détections.

$$\omega = 0.95$$

Permet de prendre en compte la maximisation des Vrais Négatifs.

$$\omega_2 = 0.08$$

Permet de prendre en compte la minimisation du périmètre de la zone 1 : permet d'assurer que la forme de la solution est acceptable par le client.

$$\omega_3 = 0.85$$

Permet de prendre en compte la minimisation des Faux Négatifs.

$$\beta = 140$$

Permet d'imposer plus ou moins de contrainte sur les Faux (positifs et négatifs).

$$\epsilon = 0.01$$

Tolérance sur les fractions de Faux (positifs et négatifs).

$$f = \omega_2 P + (1-\omega_2)[-\omega VP - (1-\omega)VN + \beta/2 * (\omega_3(FP-\epsilon)^2 + (1-\omega_3)(FN-\epsilon)^2)]$$

Contraintes

Un des objectifs de notre projet, était notamment de transformer certaines composantes de la fonction objectif en des contraintes : par exemple $FP < \epsilon$ pour assurer une certaine fraction de Faux Positifs ou encore $P < P_{convexe}$ pour obliger la zone à être convexe et donc forcer l'optimisation à nous donner une zone de forme acceptable pour le client sans minimiser directement le périmètre dans la fonction objectif.
Or, l'implémentation de contraintes a toujours résulté en des solutions non-conformes à nos attentes : que ce soit en terme d'exigence de détection du défaut ou de forme de la solution. Nous n'utilisons donc pas de contraintes d'égalité ni d'inégalité dans l'algorithme.

5.2 Résultats

L'algorithme tourne en 16 minutes grâce à la parallélisation implémentée.

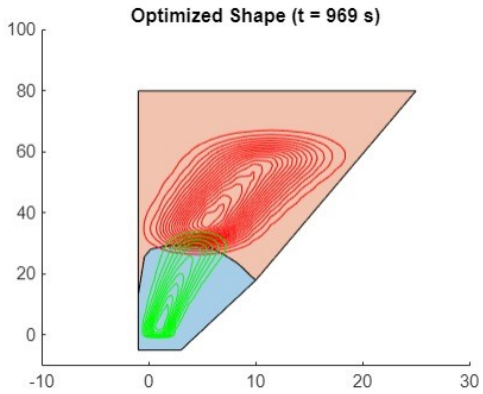


FIGURE 8 – Solution minimale obtenue avec $\beta = 140$ avec 150 départs.
VP = 0.91469 FP = 0.0102

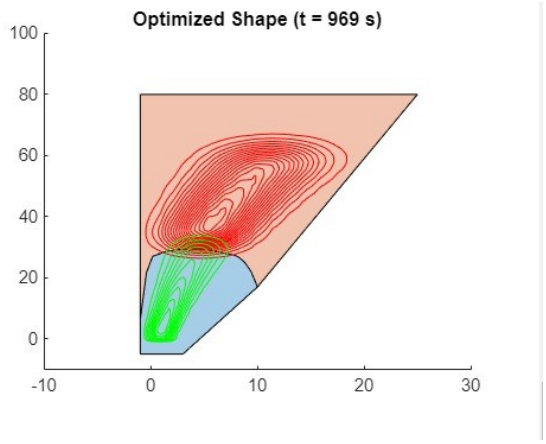


FIGURE 9 – Autre solution obtenue avec $\beta = 140$ avec 150 départs.
VP = 0.9097 FP = 0.0092

Ces solutions proviennent de deux optimisations avec des points initiaux différents. On peut voir que la première est meilleure par rapport à VP et que la seconde est meilleure par rapport à FP.

5.3 Dépendance aux paramètres d'optimisation

Dépendance à l'état initial

Dans l'algorithme, nous initialisons la forme en créant un bruit sur les ordonnées autour de $Y0 = 30$. Ce bruit est aléatoire et permet de pouvoir commencer l'optimisation en partant de points initiaux différents pour ne pas tomber dans le même minimum local.
Nous avons remarquer que optimiser sur 150 points initiaux était suffisant pour obtenir une solution satisfaisante.

Voici la répartition globale des solutions obtenues triée par ordre croissant de fonction objectif.

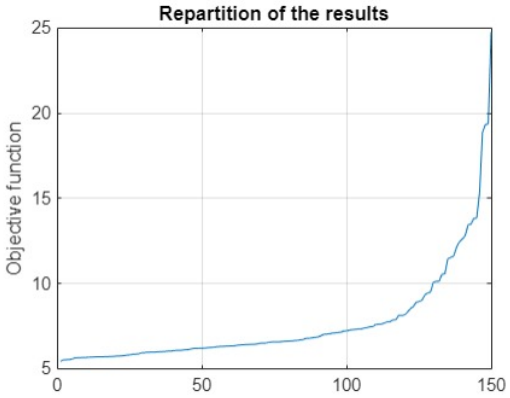


FIGURE 10 – Valeurs des objectifs pour chaque solutions issues d'une optimisation à partir d'un point initial différent.

Pour vérifier la pertinence des paramètres choisis, nous avons effectuer plusieurs expériences.

Intérêt de prendre en compte VN et FN

Dans ce test, on fixe $\omega = 1$ et $\omega_3 = 1$ pour ne prendre en compte que les Vrais Positifs et Faux Positifs.

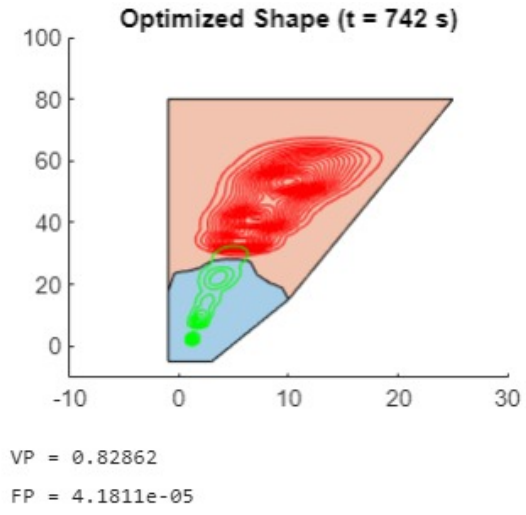


FIGURE 11 – Solution trouvée sans prendre en compte VN ni FN.

On voit que la fraction VP est plus faible. FP aussi est plus faible mais nous n'avons pas besoin d'une fraction de Faux Positifs aussi faible.

Cela nous montre que la prise en compte de FN et VN est nécessaire malgré que nos objectifs principaux soient autour de FP et VP.

Impact de la variation de la contrainte β

β est le paramètre le plus intéressant à modifier. Il est permet de choisir la priorité entre la optimisation par rapport à VP ou FP.
Diminuer la contrainte β a pour effet de prioriser l'augmentation de VP à la minimisation de FP. Au contraire, augmenter β permet d'être plus sélectif et d'exclure plus de points de défauts de la ligne 2 de la zone de coupure instantanée.

Impact de l'augmentation de la contrainte sur le périmètre

Renforcer la contrainte sur le périmètre va avoir pour effet de réduire la taille du polygone final obtenu. Or, cette contrainte est là initialement pour garantir la bonne forme de la solution obtenue.

Conclusion

Cette méthode d'optimisation offre une grande liberté au client. On peut choisir aisément les paramètres en fonction des attentes de sélectivité. L'optimisation est fiable.

5.4 Perspectives d'améliorations

Recherche de paramètres adaptés

Avec plus de temps, nous pourrions approfondir la recherche de jeu de paramètres pertinents pour le client.

Résolution par dérivée de forme

Notre algorithme utilise la résolution par différences finies. Nous avons essayé d'implémenter la dérivée de forme en utilisant les gradients des densités de probabilités. Cependant, l'optimisation se termine trop tôt et le résultat est insatisfaisant.

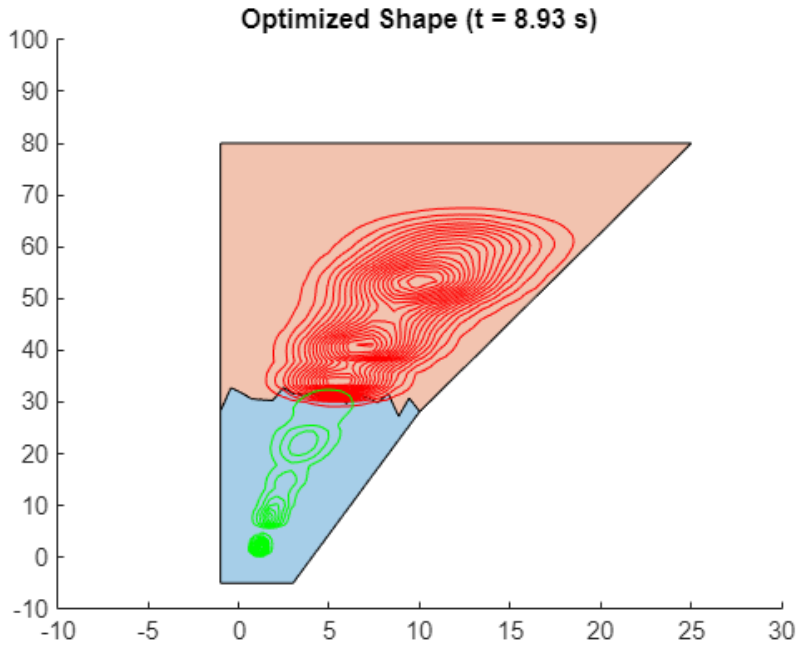


FIGURE 12 – Tentative d'implémentation de l'optimisation par dérivée de forme

L'optimisation par dérivée de forme permettrait de réduire le temps d'optimisation par un facteur 10 !

6 Implémentation d'un algorithme probabiliste

6.1 Définition du problème d'optimisation

L'objectif de cette partie est d'optimiser la configuration des polygones afin de maximiser le taux de vrais positifs tout en minimisant le taux de faux positifs. Pour ce faire, nous avons comme objectif de tracer un front de Pareto interactif qui permet de visualiser les polygones optimisés.

Les définitions des critères d'optimisation sont les suivantes :

$$\text{Taux de vrais positifs (TVP)} = \frac{VP}{VP + FN} \tag{1}$$

$$\text{Taux de faux positifs (TFP)} = \frac{FP}{FP + VN} \tag{2}$$

Pour atteindre cet objectif, nous utilisons un algorithme génétique de type NSGA-II, implémenté à l'aide de la fonction `gamultiobj` de Matlab. Inspirée de la sélection naturelle, cette méthode repose sur un processus itératif comprenant les étapes suivantes :
- **Initialisation** : une population aléatoire de solutions est générée avec des bornes inférieures et supérieures que nous pouvons choisir.
- **Sélection** : les meilleures solutions sont conservées.
- **Croisement** : de nouvelles solutions sont formées par recombinaison des meilleures solutions.
- **Mutation** : des modifications aléatoires peuvent avoir lieu

Un front de Pareto représente l'ensemble des solutions optimales lorsqu'on cherche à optimiser plusieurs critères en même temps. Un point appartient au front de Pareto s'il n'existe pas un autre point qui soit meilleur sur un critère sans être moins bon sur un autre critère.

6.2 Réalisation de premiers algorithmes sur des fonctions de densité test

6.2.1 Un seul polygone

Dans un premier temps, nous optimisons l'emplacement d'un unique polygone dans la zone 1. Les objectifs considérés sont :

— Maximiser le nombre de vrais positifs (VP),
— Minimiser le nombre de faux positifs (FP).

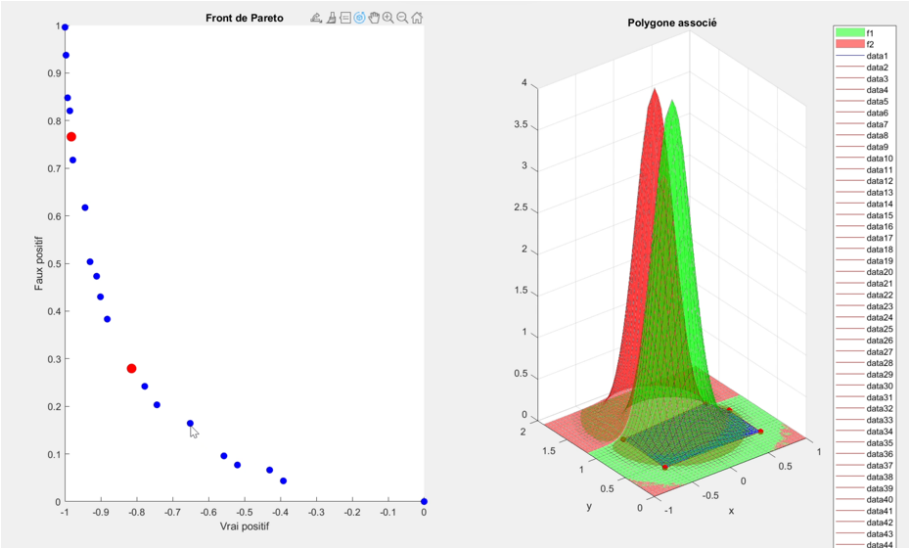


FIGURE 13 – Exemple d'optimisation d'un polygone avec son front de Pareto.

Le front de Pareto obtenu (figure de gauche) permet de sélectionner un polygone optimal dont la forme évolue pour réduire les faux positifs.

6.2.2 Mise en place d'un deuxième polygone

Afin d'atteindre les objectifs du projet, nous introduisons un second polygone pour optimiser la densité de la zone 2 tout en minimisant celle de la zone 1. L'optimisation repose sur le vecteur :

$$X = (x_{11}, y_{11}, x_{12}, y_{12}, \ldots, x_{1nb}, y_{1nb}, x_{21}, y_{21}, x_{22}, y_{22}, \ldots, x_{2nb}, y_{2nb})$$

Ce vecteur regroupe les nb coordonnées des sommets du premier polygone, suivies des nb coordonnées des sommets du second polygone. Notre fonction objectif prend donc en compte trois objectifs : maximiser VP,

minimiser FP et maximiser VN. L'utilisation de trois objectifs conduit à l'apparition d'un front de Pareto en 3 dimensions. Cependant, les résultats obtenus ne sont pas satisfaisants, les polygones se superposent et n'ont pas la forme désirée.

6.3 Reformulation du problème et passage aux vraies densités

Pour éviter les superpositions, nous avons repensé l'approche :

— Seuls les sommets situés sur la frontière entre les deux polygones sont libres : en bleu sur la figure

— Certains sommets du polygone 1 et du polygone 2 sont fixés comme les points extrêmes de la frontière en noir.

— Le vecteur d'optimisation devient :

$$X = (y_1, y_2, \ldots, y_{nb}) \tag{3}$$

Ici, les y représentent les coordonnées verticales des points de la frontière, espacés régulièrement selon leur abscisse.

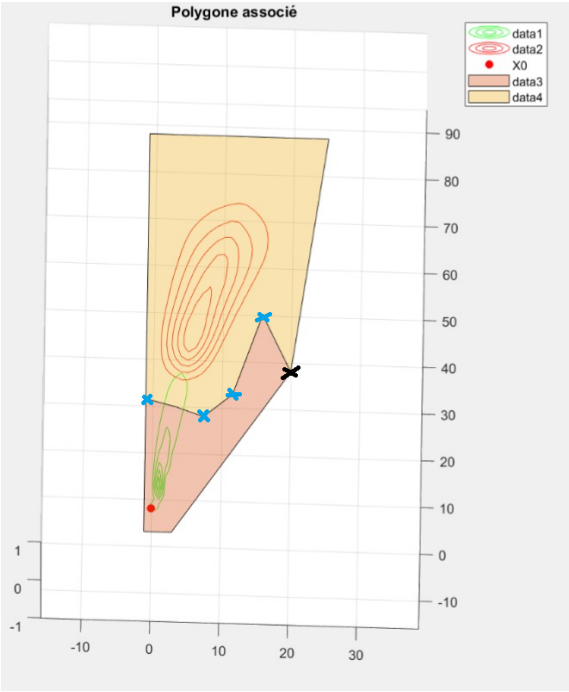


FIGURE 14 – Optimisation de la frontière

6.4 Optimisation avec l'Algorithme NSGA-II

L'algorithme NSGA-II est appliqué à l'optimisation de la frontière avec pour objectifs :

— Maximiser le taux de vrais positifs (TVP),

— Minimiser le taux de faux positifs (TFP).

Un nombre trop important de points sur la frontière rend les résultats bruités contrairement aux résultats avec l'algorithme déterministe, il est donc préférable de réduire ce nombre. En effet, nous voudrions que le zone soit convexe donc minimiser les sauts et bruits inutiles.

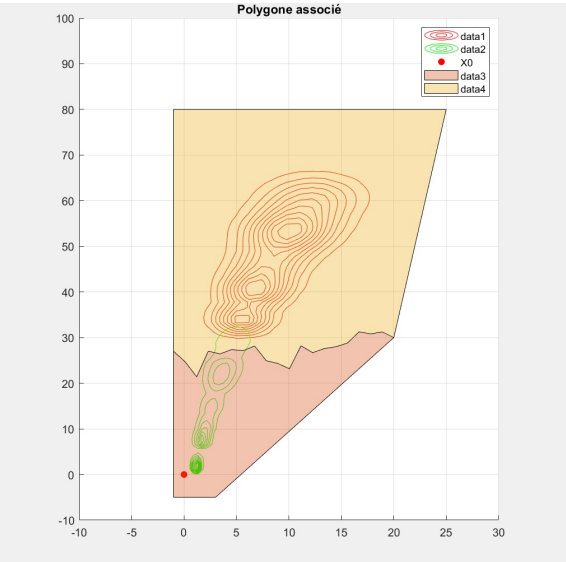


FIGURE 15 – Exemple de frontière bruitée avec 20 points d'optimisation.

Dans le cas de l'algorithme génétique il faut ainsi privilégier moins de points sur la frontière. Ici nous choisissons 6 points, une population de 300 avec 100 générations. Nous fixons les deux côtés de la frontière et la fonction d'optimisation est avec $J = [-TVP; TFP]$ pour minimiser les deux objectifs. En se déplaçant sur le Pareto nous pouvons choisir les polygones que nous préférons. Nous retenons ceux de la figure ci dessous, car ils permettent d'avoir des vrais positifs de 0.87 et 0,0038 de faux positifs. Ainsi cela permet d'éviter de couper inutilement la ligne 1, en ne touchant presque pas la courbe rouge.

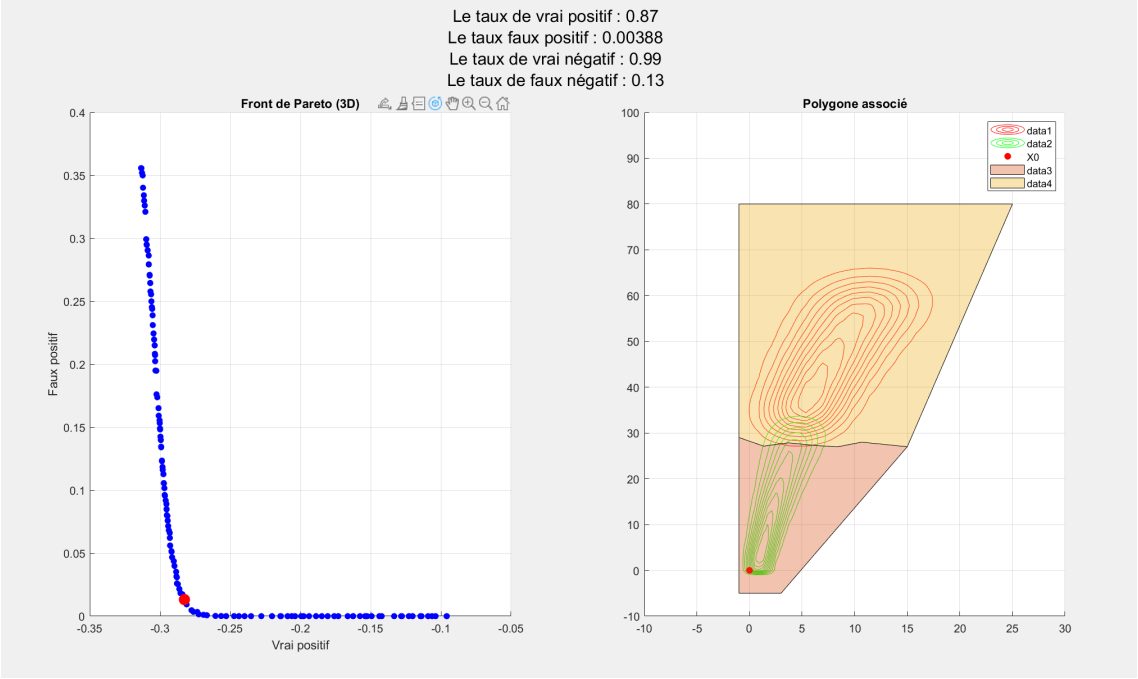


FIGURE 16 – Exemple de frontière pour minimiser les FP

Afin de maximiser le nombre de vrais positifs, nous pourrions envisager d'élargir la zone du polygone 1 afin d'englober l'ensemble de la courbe verte. Cette approche se traduirait néanmoins par une augmentation du taux de faux positifs. Ce résultat a été obtenu en considérant une population de 400 points sur 70 générations.

Dans ce cadre, nous avons fixé les deux côtés de la frontière et utilisé une fonction d'optimisation définie par $J = [-TVP; TFP]$, permettant de minimiser simultanément les deux objectifs. En explorant la frontière de Pareto, nous avons retenu les polygones présentés dans la figure ci-dessous, car ils permettent d'atteindre un nombre de vrais positifs de 0.98 et un nombre de faux positifs de 0.053 . Cette configuration permet ainsi de couvrir presque entièrement la courbe verte.
Les coordonnées du polygone 1 sont en annexe.

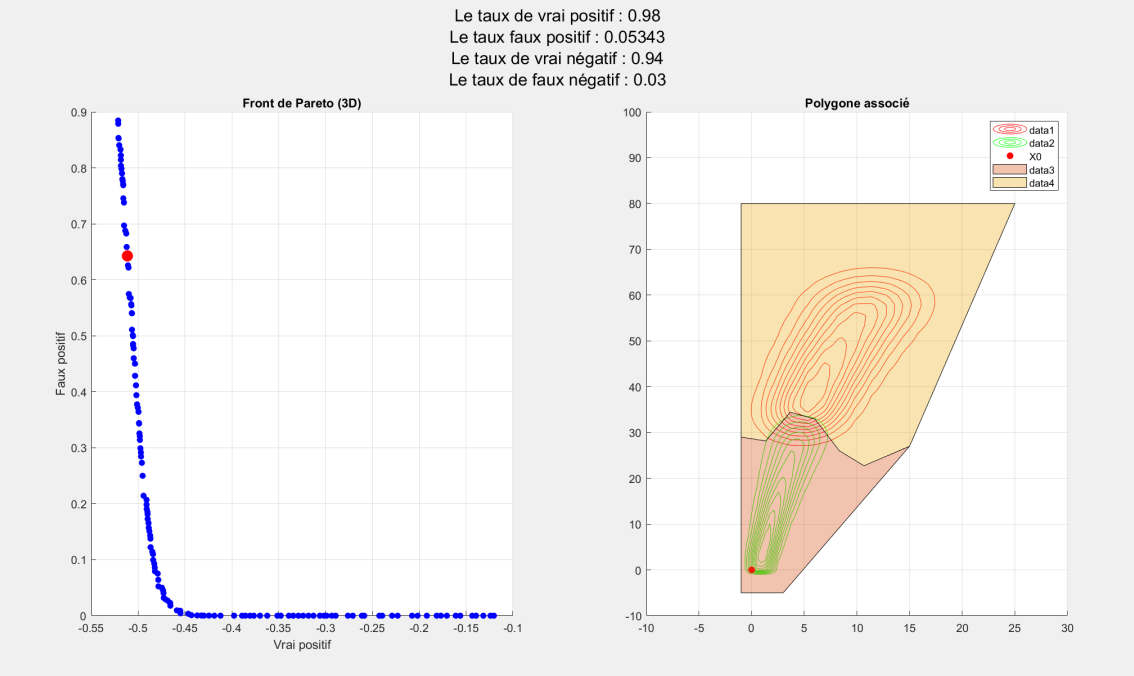


FIGURE 17 – Exemple de frontière pour maximiser les VP en minimisant les FP

6.4.1 Courbe ROC

Une courbe ROC illustre l'évolution du taux de faux positifs en fonction du taux de vrais positifs. En exploitant les points issus de la frontière de Pareto, il est possible de tracer différentes courbes ROC. Dans celle de la figure suivante, nous choisissons 6 points, une population de 300 avec 100 générations. Nous fixons les deux côtés de la frontière et la fonction d'optimisation est avec $J = [-0.6 * TVP; 0.4 * TFP]$ avec une pondération sur les objectifs permettant de favoriser un peu plus la maximisation des vrais positifs. On obtient une aire sous la courbe de 0.7642 ce qui se rapproche de 1 qui serait la solution idéale. En effet, une courbe ROC avec une aire sous la courbe proche de 1 signifie que l'algorithme est efficace pour distinguer les vrais positifs des négatifs. Ici $0.7664 > 0.5$ ce qui signifie que l'algorithme est meilleur qu'un tirage aléatoire.

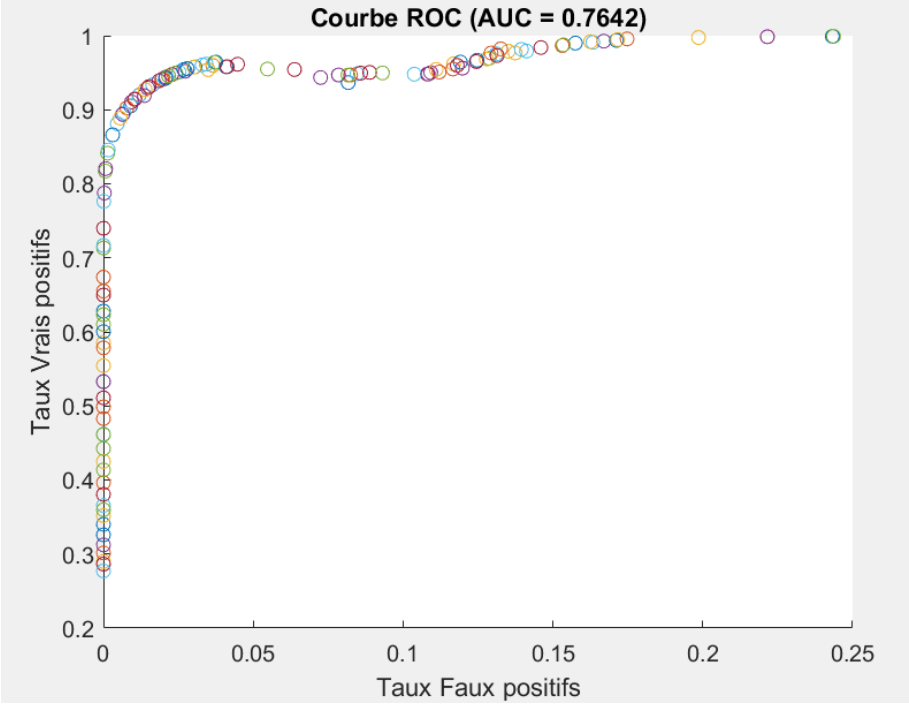


FIGURE 18 – Exemple de courbe ROC obtenue avec aire sous la courbe = 0.7642

6.4.2 Comparaison entre la méthode génétique et la méthode déterministe

Afin de mieux comparer les deux méthodes, nous pouvons concevoir un algorithme génétique intégrant la fonction d'optimisation de l'algorithme déterministe. Cependant, cette approche ne permet pas de générer un front de Pareto, car l'optimisation repose sur un unique objectif. A la différence avec l'algorithme déterministe le point d'extrémité droite de la frontière (20,30) est fixé, ce qui permet de définir ensuite des bornes inférieures essentielles au bon fonctionnement de l'algorithme génétique. Mais nous gardons les mêmes ω et ϵ que ceux de la partie 4 ainsi que 20 points sur la frontière.

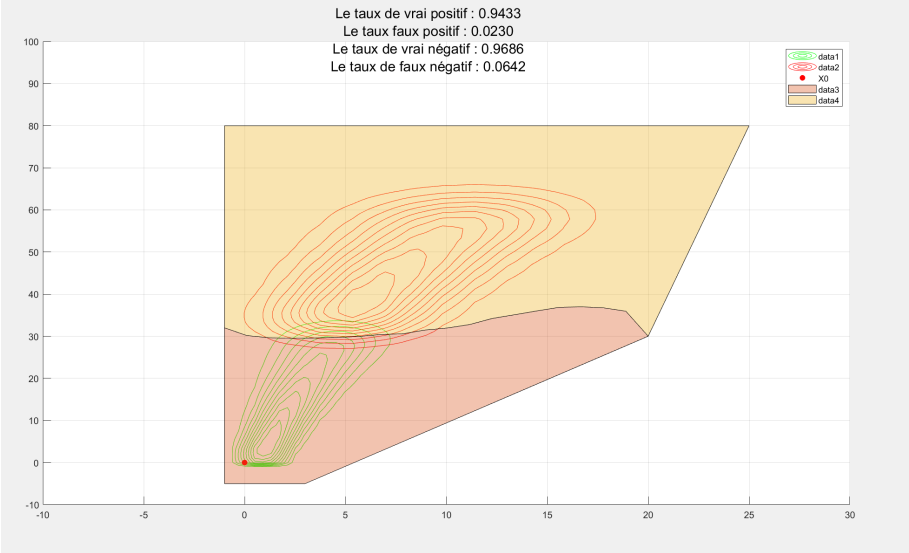


FIGURE 19 – Méthode NSGAII, même fonction objectif que la méthode déterministe mais en fixant les 2 points sur la frontière

Cette figure a été obtenue en considérant une population de 300 individus sur 70 générations. Nous observons que le résultat obtenu, avec un taux de vrais positifs de 0.94 et un taux de faux positifs de 0.02, est moins satisfaisant si l'on souhaite minimiser le taux de faux positifs que celui issu de la même fonction objectif appliquée à la méthode déterministe qui possède un taux de vrais positifs de 0.90 et de faux positif de 0.009. De plus, le temps de calcul est beaucoup plus lent avec la méthode génétique.

Cette différence de performance pourrait s'expliquer par l'imposition de bornes supérieures et inférieures dans l'algorithme génétique, introduisant ainsi des contraintes plus restrictives. Une autre hypothèse est que le nombre relativement faible de générations choisies n'a peut-être pas permis à l'algorithme génétique d'atteindre une solution optimale.

6.4.3 Conclusion sur la méthode génétique et idées pour aller plus loin

Le choix de la population initiale ainsi que du nombre de générations a un impact significatif sur le front de Pareto obtenu, mais influe également sur le temps de calcul, qui augmente considérablement avec ces paramètres.

L'un des principaux avantages de la méthode génétique réside dans sa flexibilité : elle permet de sélectionner un polygone correspondant aux taux de vrais positifs (VP), vrais négatifs (VN), faux positifs (FP) et faux négatifs (FN) souhaités, en se déplaçant simplement sur le front de Pareto. Dans le cadre de notre étude et des tests réalisés, cette approche a permis d'obtenir de meilleurs résultats pour l'optimisation simultanée des taux de vrais positifs (TVP) et de faux positifs (TFP) que par la fonction objectif f de la méthode déterministe.

Dans le cadre de la méthode génétique, il est essentiel que les nouvelles générations respectent certaines contraintes structurelles afin d'assurer le bon fonctionnement de l'algorithme. Par exemple, il est impératif que les polygones générés ne présentent pas d'intersections entre leurs côtés. Pour garantir cette condition, nous avons fixé le sommet noir et défini des bornes inférieures en fonction de ce point sur la frontière.

Une piste d'amélioration consisterait à concevoir un algorithme capable de déplacer dynamiquement ce sommet afin d'optimiser la configuration du polygone. Un algorithme a été réalisé qui permet de déplacer le sommet fixé qui est fonctionnel (voir annexe), mais il faut encore rendre ce déplacement dynamique. Pour l'instant nous pouvons changer x_{sommet} et y_{sommet} qui correspondent aux coordonnées du point noir et mettre celles que nous voulons.

Les coordonnées du polygone 1 sont en annexe pour les deux figures.

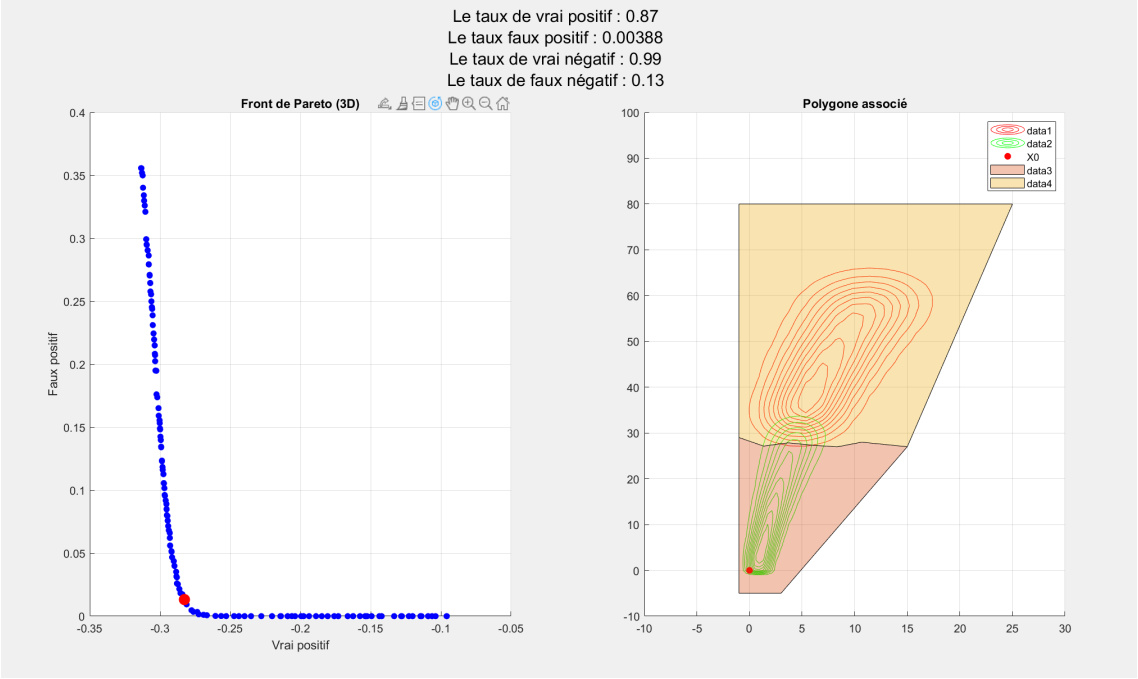


FIGURE 20 – Sommet fixe à (15,27)

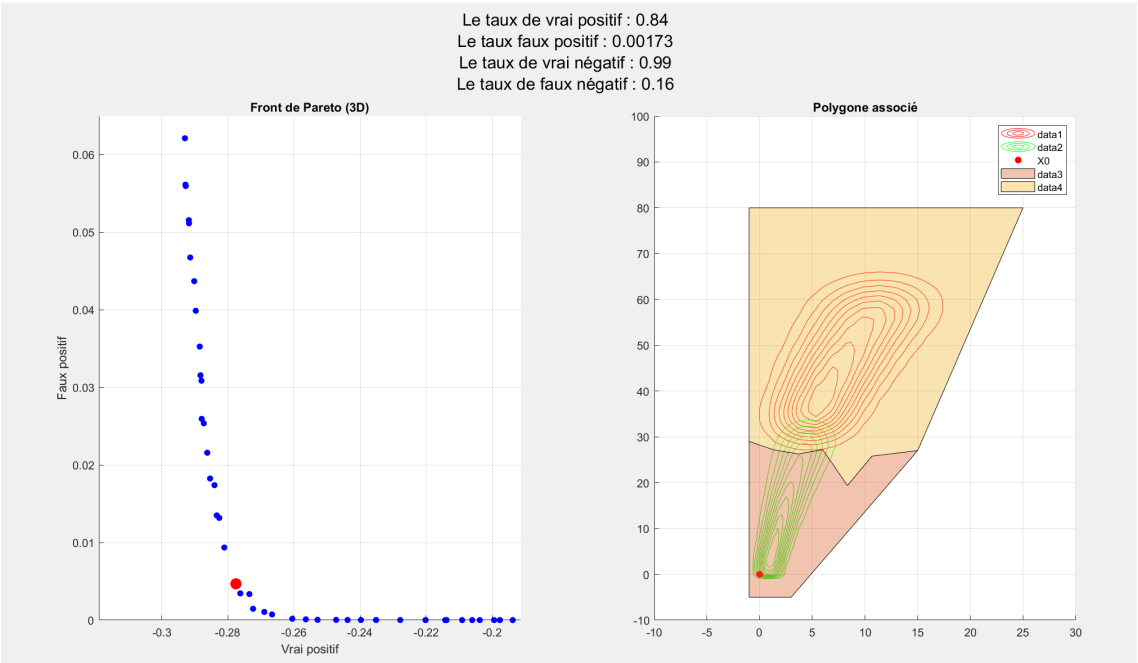


FIGURE 21 – Sommet fixe à (15,27)

7 Étude comparative des solutions

7.1 Comparaison des solutions des deux algorithmes

Les deux algorithmes ont des résultats similaires en matière de performances de détection de défauts et en matière de temps d'optimisation. La méthode déterministe tend à produire des polygones plus convexes, car elle intègre le périmètre dans sa fonction objectif, tandis que la méthode génétique offre une plus grande liberté dans le choix de la forme des polygones optimaux avec un Pareto. Ces algorithmes ont des paramètres que nous pouvons changer afin d'adapter leur fonctionnement aux exigences spécifiques du client et d'optimiser les compromis entre les différents critères de performance.

7.2 Comparaison avec la solution initiale

La solution actuellement adoptée dans les relais est la suivante :

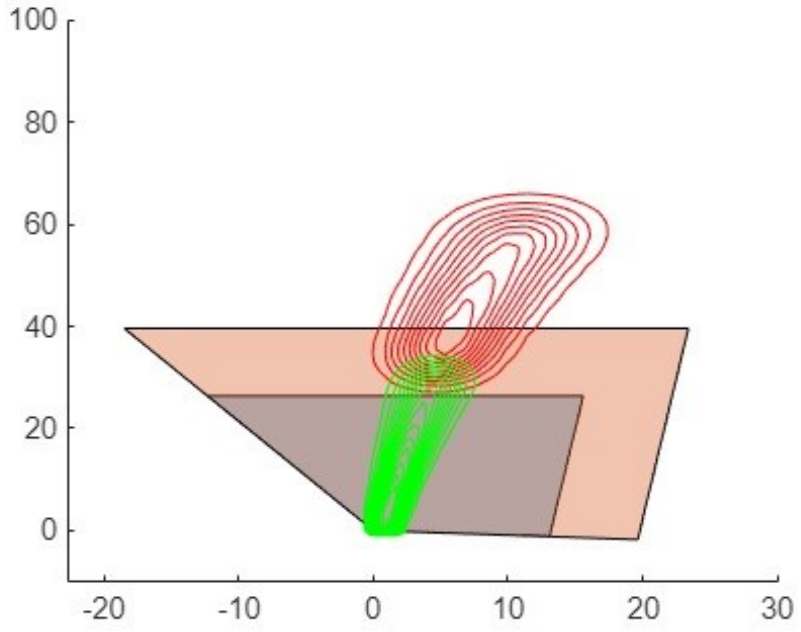


FIGURE 22 – Polygone de la littérature trouvé par calculs.
VP = 0.826 FP = 0.001

Aux vues de ses bonnes performances, notamment en terme de mini-

misation de FP, on comprend pourquoi cette solution est actuellement la plus répandue. Cependant les solutions que nous trouvons ont pour avantage d'avoir un FP du même ordre de grandeur tout en ayant un VP plus grand d'environ 8 %.
En pratique, c'est souhaitable car on va plus souvent couper immédiatement lorsqu'un court circuit va avoir lieu sur la ligne 1. La sécurité du réseau est renforcée.

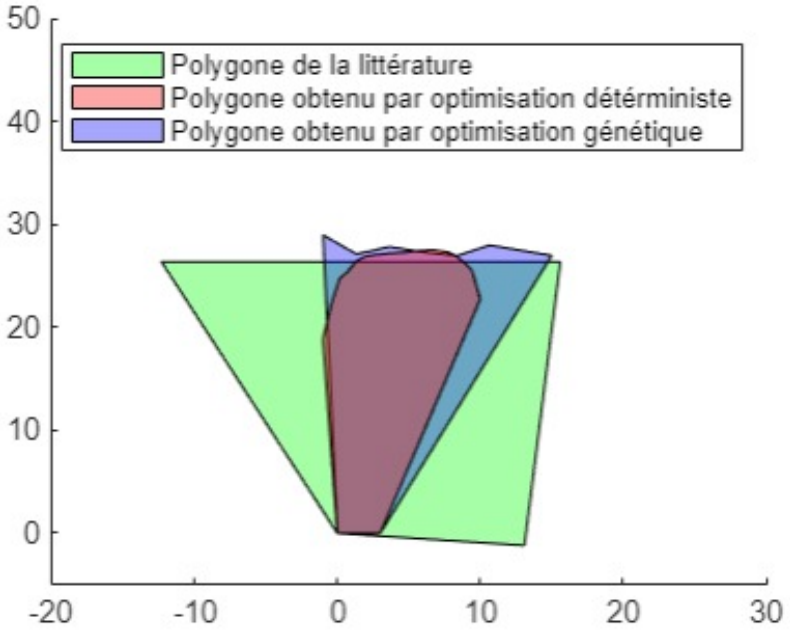


FIGURE 23 – Comparaison des trois zones de coupure instantanée obtenue.

7.3 Perspectives d'améliorations

On aurait voulu pouvoir combiner les méthodes d'optimisations en utilisant l'algorithme déterministe sur la population finale obtenue par l'algorithme génétique.

8 Annexes

8.1 Annexe 1 : Code Matlab - Densités de probabilité

8.1.1 Calcul de l'impédance équivalente du circuit selon la position du défaut

```
1
2  function [Vmes,Ilig]=calcul_tension_courant_relai_ddp(E,Xcc,
       Rcc,Zlig,C,RL,XL,pdf_R,pdf_X,pdf_C,lignedef,posdef,omega,
       Rdef)
3  % On calcule la tension et le courant mesures par le relai
4
5  %% 1) Calcul de Ztot
6  % Ztot est l'impedance reseau (generateur deconnecte, qui
       prend en compte
7  % le defaut)
8
9  switch lignedef
10     case 1 % defaut sur la ligne 1
11         %% Calcul des grandeurs electriques aval du defaut
12         ZL=1/(1/RL+1/(1j*XL)); % charge 3 (P et Q, Rl en
               parallele Xl)
13         Ztot=ZL; % tout au bout
14         % ligne 2-3
15         C3=C*random(pdf_C); % incertitude sur le C des lignes
16         X3=imag(Zlig)*random(pdf_X); % incertitude sur le X
               des lignes
17         R3=real(Zlig)*random(pdf_R); % incertitude sur le R
               des lignes
18         % on remonte de 1 ligne
19         Ztot=1/(1/Ztot+1j*C3*omega/2);
20         Ztot=Ztot+(R3+1j*X3);
21         % au noeud 2
22         Ztot=1/(1/Ztot+1j*C3*omega/2+1/ZL);
23         % ligne 1 -2
24         C2=C*random(pdf_C);
25         X2=imag(Zlig)*random(pdf_X);
26         R2=real(Zlig)*random(pdf_R);
27         Ztot=1/(1/Ztot+1j*C2*omega/2);
28         Ztot=Ztot+(R2+1j*X2);
29         % charge noeud 1
30         Ztot=1/(1/Ztot+1/ZL+1j*C2*omega/2);
31
32     %% Calcul des grandeurs electriques defaut et amont
```

```
        % ligne PCC-1 apres defaut
        % on a posdef[%] de la ligne en amont du defaut

        % defaut (position posedef fixe a l'exterieur de la
           fonction)
        % impedance de defaut aleatoire

        C1=C*random(pdf_C)*(1-posdef);
        X1=imag(Zlig)*random(pdf_X)*(1-posdef);
        R1=real(Zlig)*random(pdf_R)*(1-posdef);
        Ztot=1/(1/Ztot+1j*C1*omega/2);
        Ztot=Ztot+(R1+1j*X1);
        % defaut resistif en parallele (Rdef fixe en dehors de
            la fonction)
        Ztot=1/(1/Ztot+1/Rdef+1j*C1*omega/2);
        % on remonte avant defaut
        C1p=C1*(posdef)/(1-posdef);
        X1p=X1*(posdef)/(1-posdef);
        R1p=R1*(posdef)/(1-posdef);
        Ztot=1/(1/Ztot+1j*C1p*omega/2);a
        Ztot=Ztot+(R1p+1j*X1p);
        Ztot=1/(1/Ztot+1j*C1p*omega/2);
        % Point of Command Coupling (source de tension Vgrid)
        Ztot=Ztot+Rcc+1j*Xcc;

    case 2 % pareil que precedemment, sur la ligne 2
        ZL=1/(1/RL+1/(1j*XL)); % 1 charge
        Ztot=ZL; % tout au bout
        % ligne 2-3
        C3=C*random(pdf_C);
        X3=imag(Zlig)*random(pdf_X);
        R3=real(Zlig)*random(pdf_R);
        % on remonte de 1 ligne
        Ztot=1/(1/Ztot+1j*C3*omega/2);
        Ztot=Ztot+(R3+1j*X3);
        % au noeud 2
        Ztot=1/(1/Ztot+1j*C3*omega/2+1/ZL);
        % ligne 1 -2 ares defaut
        % on a posdef[%] de la ligne en amont du defaut
        C2=C*random(pdf_C)*(1-posdef);
        X2=imag(Zlig)*random(pdf_X)*(1-posdef);
        R2=real(Zlig)*random(pdf_R)*(1-posdef);
        Ztot=1/(1/Ztot+1j*C2*omega/2);
        Ztot=Ztot+(R2+1j*X2);
        % le defaut en //
```

```
        Ztot=1/(1/Ztot+1/Rdef+1j*C2*omega/2);
        %ligne 1-2 avant defaut
        C2p=C2*(posdef)/(1-posdef);
        X2p=X2*(posdef)/(1-posdef);
        R2p=R2*(posdef)/(1-posdef);
        Ztot=1/(1/Ztot+1j*C2p*omega/2);
        Ztot=Ztot+(R2p+1j*X2p);
        % le noeud 1
        Ztot=1/(1/Ztot+1/ZL+1j*C2p*omega/2);
        % ligne PCC-1
        C1=C*random(pdf_C);
        X1=imag(Zlig)*random(pdf_X);
        R1=real(Zlig)*random(pdf_R);
        Ztot=1/(1/Ztot+1j*C1*omega/2);
        Ztot=Ztot+(R1+1j*X1);
        Ztot=1/(1/Ztot+1j*C1*omega/2);
        % on remonte avant defaut
        % PCC
        Ztot=Ztot+Rcc+1j*Xcc;
    case 3 % pareil que precedemment, sur la ligne 3
        ZL=1/(1/RL+1/(1j*XL)); % 1 charge
        Ztot=ZL; % tout au bout
        % ligne 2-3 avant defaut
        % on a posdef[%] de la ligne en amont du defaut
        C3=C*random(pdf_C)*(1-posdef);
        X3=imag(Zlig)*random(pdf_X)*(1-posdef);
        R3=real(Zlig)*random(pdf_R)*(1-posdef);
        % on remonte de 1 ligne
        Ztot=1/(1/Ztot+1j*C3*omega/2);
        Ztot=Ztot+(R3+1j*X3);
        % au defaut en //
        Ztot=1/(1/Ztot+1j*C3*omega/2+1/Rdef);
        %ligne 2-3 avant defaut
        C3p=C3*posdef/(1-posdef);
        X3p=X3*posdef/(1-posdef);
        R3p=R3*posdef/(1-posdef);
        % on remonte de 1 ligne
        Ztot=1/(1/Ztot+1j*C3p*omega/2);
        Ztot=Ztot+(R3p+1j*X3p);
        % au noeud 2
        Ztot=1/(1/Ztot+1j*C3p*omega/2+1/ZL);
        % ligne 1 -2
        C2=C*random(pdf_C);
        X2=imag(Zlig)*random(pdf_X);
        R2=real(Zlig)*random(pdf_R);
```

```
        Ztot=1/(1/Ztot+1j*C2*omega/2);
        Ztot=Ztot+(R2+1j*X2);
        % charge noeud 1
        Ztot=1/(1/Ztot+1/ZL+1j*C2*omega/2);
        % ligne PCC-1
        C1=C*random(pdf_C);
        X1=imag(Zlig)*random(pdf_X);
        R1=real(Zlig)*random(pdf_R);
        Ztot=1/(1/Ztot+1j*C1*omega/2);
        Ztot=Ztot+(R1+1j*X1);
        Ztot=1/(1/Ztot+1j*C1*omega/2);
        % on remonte avant defaut
        % PCC
        Ztot=Ztot+Rcc+1j*Xcc;
end

%% Calcul des grandeurs vues par le relais
% relais situe au noeud 2
% On calcule Itot et on redescend pour avoir Vmes,  Imes

Itot=E/Ztot;
Ilig=Itot;
Vmes=E-(Rcc+1j*Xcc)*Ilig;
% tension au noeud 1 (sortie du generateur)
% le relais est au noeud 2 !

switch lignedef
    case 1
        % defaut au bout de posdef*ligne1
        % C1*posdef/2 a la tension Vmes
        Ilig=Ilig-Vmes*1j*C1p*omega/2;
        Vmes=Vmes-(R1p+1j*X1p)*Ilig;
        % tension au defaut
        Ilig=Ilig-Vmes*1j*C1p*omega/2;
        Ilig=Ilig-Vmes/Rdef;
        %C1*(1-posdef)/2 en //
        Ilig=Ilig-Vmes*1j*C1*omega/2;
        Vmes=Vmes-(R1+1j*X1)*Ilig;
        Ilig=Ilig-Vmes*1j*C1*omega/2;
        % au noeud 2 du relai !
    case 2
        Ilig=Ilig-Vmes*1j*C1*omega/2;
        Vmes=Vmes-(R1+1j*X1)*Ilig;
        Ilig=Ilig-Vmes*1j*C1*omega/2;
    case 3
```

```
166         Ilig=Ilig-Vmes*1j*C1*omega/2;
167         Vmes=Vmes-(R1+1j*X1)*Ilig;
168         Ilig=Ilig-Vmes*1j*C1*omega/2;
169         % au noeud 2 du relai !
170 end
171 end
```

Listing 1 – Code MATLAB pour le calcul de l'impédance équivalente du réseau selon la position du défaut

8.1.2 Génération des densités de probabilité via une simulation de Monte Carlo

```
1  clc; clear ; close all ; path=pwd();
2  [s,e]=regexp(path,'.+(?=[\\/]{1,2}gridModel)');
3  path=path(s:e); addpath(genpath(path)); clear s e path
4
5  %% Definition des parametres
6  Zlig = 3 + 1j * 33; % Impedance de ligne
7  U = 400e3; % Tension nominale
8  Scc = 800e6; % Puissance de court-circuit
9  XR_ratio = 30; % Rapport X/R du reseau
10 f0 = 50; % Frequence du reseau
11 Clig = 1e-6; % Capacite lineique de la ligne
12 Rdef = 1; % Resistance du defaut
13 PL = 200e6; % Puissance de charge
14 cosphiL = 0.95; % Facteur de puissance
15 N_MC = 10000; % Nombre de simulations Monte Carlo
16
17 %% Calcul des grandeurs electriques
18 E = U / sqrt(3); % Tension du reseau
19 Xcc = Scc / (3 * E^2); % Reactance de court-circuit
20 Rcc = Xcc / XR_ratio; % Resistance de court-circuit
21 RL = 3 * E^2 / PL; % Resistance lineique de la ligne
22 QL = sqrt((PL / cosphiL)^2 - PL^2); % Puissance reactive
23 XL = 3 * E^2 / QL; % Reactance lineique de la ligne
24 omega = 2 * pi * f0; % Pulsation
25
26 %% Definition des incertitudes
27 incerZ = 0.1; % Incertitude de mesure de l'impedance
28 incerC = 0.05; % Incertitude de mesure de la capacite
29 typedist = 'gauss'; % Choix du type de distribution pour les
      incertitudes sur R, X, C dans le MC
30
```

```
%% Lois de probabilite
switch typedist
    case 'uniform' % Densites uniformes
        pdf_R = makedist('Uniform', 'lower', 1 - incerZ, '
           upper', 1 + incerZ);
        pdf_X = makedist('Uniform', 'lower', 1 - incerZ, '
           upper', 1 + incerZ);
        pdf_C = makedist('Uniform', 'lower', 1 - incerC, '
           upper', 1 + incerC);
    case 'gauss' % Densites gaussiennes
        pdf_R = makedist('normal', 'mu', 1, 'sigma', incerZ /
           3);
        pdf_X = makedist('normal', 'mu', 1, 'sigma', incerZ /
           3);
        pdf_C = makedist('normal', 'mu', 1, 'sigma', incerC /
           3);
end

%% Definition du plan d'experience
ligne_def = [2 3]'; % Choix des lignes pour les defauts
N_pos = 10000; % Nombre de positions de defauts
posdefvec = rand(N_pos, 1); % Positions de defauts aleatoires
vecpos = (1:length(posdefvec))'; % Numerotation des defauts

plan_exp = kron(ligne_def, ones(length(posdefvec), 1));
plan_exp = [plan_exp, kron(ones(length(ligne_def), 1), vecpos)
   ];

%% Monte Carlo vectorise pour chaque experience
nExp = size(plan_exp,1); % Nombre d'experience
Zapp = zeros(length(plan_exp(:,1)), N_MC); % Initialisation de
    Zapp

for indexp = 1:nExp
    lignedef = plan_exp(indexp, 1); % Choix de la ligne indexp
    posdef = posdefvec(plan_exp(indexp, 2)); % Position du
       defaut
    disp(['Experience n  ', num2str(indexp), ' sur ', num2str(
       nExp)]);

    % Appel unique de la fonction (pour des parametres
       identiques)
    [Vmes0, Imes0] = calcul_tension_courant_relai_ddp(...
        E, Xcc, Rcc, Zlig, Clig, RL, XL, ...
        pdf_R, pdf_X, pdf_C, lignedef, posdef, omega, Rdef);
```

```
65
66      % Generation vectorisee des perturbations aleatoires pour
           N_MC simulations
67      r_R  = random(pdf_R, N_MC, 1);
68      r_X1 = random(pdf_X, N_MC, 1);
69      r_X2 = random(pdf_X, N_MC, 1);
70      r_C  = random(pdf_C, N_MC, 1);
71
72      % Calcul vectorise de Vmes et Imes
73      Vmes_vec = abs(Vmes0) .* r_R .* exp(1j * (angle(Vmes0) +
           r_X1));
74      Imes_vec = abs(Imes0) .* r_C .* exp(1j * (angle(Imes0) +
           r_X2));
75
76      % Calcul de l'impedance apparente pour toutes les
           simulations
77      Zapp(indexp, :) = Vmes_vec ./ Imes_vec;
78  end
79
80  %% Identification des defauts par ligne
81  indlig1 = find(plan_exp(:,1) == 3); % Indices des defauts sur
       ligne 1
82  indlig2 = find(plan_exp(:,1) == 2); % Indices des defauts sur
       ligne 2
83
84  Z1 = Zapp(indlig2,:); % Valeurs de Zapp pour ligne 1
85  Z2 = Zapp(indlig1,:); % Valeurs de Zapp pour ligne 2
86
87  %% Histogrammes 2D
88  nbins = 25; % Nombre de bins
89
90  % Histogramme pour la ligne 1
91  h1 = histogram2(real(Z1(:)), imag(Z1(:)), nbins);
92  val1 = zeros(size(h1.Values) + 2);
93  val1(2:end-1, 2:end-1) = h1.Values;
94  X1 = h1.XBinEdges; Y1 = h1.YBinEdges;
95
96  % Histogramme pour la ligne 2
97  h2 = histogram2(real(Z2(:)), imag(Z2(:)), nbins);
98  val2 = zeros(size(h2.Values) + 2);
99  val2(2:end-1, 2:end-1) = h2.Values;
100 X2 = h2.XBinEdges; Y2 = h2.YBinEdges;
101
102 %% Lissage des histogrammes
103 sigma = 1.2; % Ecart-type du lissage
```

```
kernelSize = 4; % Taille du noyau
h = fspecial('gaussian', kernelSize, sigma); % Creation du masque gaussien

% Application du masque de convolution a l'histogramme
val1_smooth = conv2(val1, h, 'same');
val2_smooth = conv2(val2, h, 'same');
% Suppression des artefacts aux bords
bord = 1;
val1_smooth(1:bord, :) = 0;
val1_smooth(end-bord+1:end, :) = 0;
val1_smooth(:, 1:bord) = 0;
val1_smooth(:, end-bord+1:end) = 0;

val2_smooth(1:bord, :) = 0;
val2_smooth(end-bord+1:end, :) = 0;
val2_smooth(:, 1:bord) = 0;
val2_smooth(:, end-bord+1:end) = 0;
%% Affichage des histogrammes
figure;

% Histogramme original pour Z1
subplot(2,2,1);
bar3(X1(1:end-1), val1(2:end-1, 2:end-1)');
title('Histogramme original - Ligne 1');
xlabel('Re(Z)');
ylabel('Im(Z)');
zlabel('Frequence');
grid on; view(3);

% Histogramme lisse pour Z1
subplot(2,2,2);
bar3(X1(1:end-1), val1_smooth(2:end-1, 2:end-1)');
title('Histogramme lisse - Ligne 1');
xlabel('Re(Z)');
ylabel('Im(Z)');
zlabel('Frequence lissee');
grid on; view(3);

% Histogramme original pour Z2
subplot(2,2,3);
bar3(X2(1:end-1), val2(2:end-1, 2:end-1)');
title('Histogramme original - Ligne 2');
xlabel('Re(Z)');
ylabel('Im(Z)');
```

```
148 zlabel('Frequence');
149 grid on; view(3);
150
151 % Histogramme lisse pour Z2
152 subplot(2,2,4);
153 bar3(X2(1:end-1), val2_smooth(2:end-1, 2:end-1)');
154 title('Histogramme lisse - Ligne 2');
155 xlabel('Re(Z)');
156 ylabel('Im(Z)');
157 zlabel('Frequence lissee');
158 grid on; view(3);
159
160
161 %% Interpolation des histogrammes (Brut)
162 [x1, y1] = ndgrid([X1(1), (X1(1:end-1) + X1(2:end))/2, X1(end)
    ], ...
163                   [Y1(1), (Y1(1:end-1) + Y1(2:end))/2, Y1(end)
                       ]); % Creation de la grille d'
                       interpolation pour Z1
164 f1 = griddedInterpolant(x1, y1, val1, 'linear', 'nearest'); %
    Interpolation de l'histogramme sur la grille
165
166 [x2, y2] = ndgrid([X2(1), (X2(1:end-1) + X2(2:end))/2, X2(end)
    ], ...
167                   [Y2(1), (Y2(1:end-1) + Y2(2:end))/2, Y2(end)
                       ]); % Creation de la grille d'
                       interpolation pour Z2
168 f2 = griddedInterpolant(x2, y2, val2, 'linear', 'nearest'); %
    Interpolation de l'histogramme sur la grille
169
170 %% Interpolation des histogrammes (Lisses)
171 [x1_smooth, y1_smooth] = ndgrid([X1(1), (X1(1:end-1) + X1(2:
    end))/2, X1(end)], ...
172                                 [Y1(1), (Y1(1:end-1) + Y1(2:
                                     end))/2, Y1(end)]);
173 f1_smooth = griddedInterpolant(x1_smooth, y1_smooth,
    val1_smooth, 'linear', 'nearest');
174
175 [x2_smooth, y2_smooth] = ndgrid([X2(1), (X2(1:end-1) + X2(2:
    end))/2, X2(end)], ...
176                                 [Y2(1), (Y2(1:end-1) + Y2(2:
                                     end))/2, Y2(end)]);
177 f2_smooth = griddedInterpolant(x2_smooth, y2_smooth,
    val2_smooth, 'linear', 'nearest');
178
```

```
%% Normalisation
dom1 = myPolygon([x1(1)-1, y1(1)-1; x1(1)-1, y1(end)+1; x1(end)+1, y1(end)+1; x1(end)+1, y1(1)-1], 0.2); % Domaine de la densite
intF1 = dom1.int(f1); % Calcul du volume interieur au domaine
f1 = griddedInterpolant(x1, y1, val1/intF1, 'linear', 'nearest'); % Division de la densite par le volume pour normaliser le volume total a 1

dom2 = myPolygon([x2(1)-1, y2(1)-1; x2(1)-1, y2(end)+1; x2(end)+1, y2(end)+1; x2(end)+1, y2(1)-1], 0.2); % Domaine de la densite
intF2 = dom2.int(f2); % Calcul du volume interieur au domaine
f2 = griddedInterpolant(x2, y2, val2/intF2, 'linear', 'nearest'); % Division de la densite par le volume pour normaliser le volume total a 1

% Normalisation des histogrammes lisses
intF1_smooth = dom1.int(f1_smooth);
f1_smooth = griddedInterpolant(x1_smooth, y1_smooth, val1_smooth/intF1_smooth, 'linear', 'nearest');

intF2_smooth = dom2.int(f2_smooth);
f2_smooth = griddedInterpolant(x2_smooth, y2_smooth, val2_smooth/intF2_smooth, 'linear', 'nearest');

%% Grille pour l'affichage
x = linspace(min([X1(:); X2(:)]) - 1, max([X1(:); X2(:)]) + 1, 300);
y = linspace(min([Y1(:); Y2(:)]) - 1, max([Y1(:); Y2(:)]) + 1, 300);
[xx, yy] = ndgrid(x, y);

%% Affichage des interpolations
figure;

% Avant lissage
subplot(1,2,1);
surf(xx, yy, f1(xx, yy), 'FaceColor', 'g', 'FaceAlpha', 0.5, 'EdgeAlpha', 0.2);
hold on;
surf(xx, yy, f2(xx, yy), 'FaceColor', 'r', 'FaceAlpha', 0.5, 'EdgeAlpha', 0.2);
scatter3(0, 0, 0, 'r', 'filled'); % Point X0
hold off;
```

```
legend({'Ligne 1 (Brut)', 'Ligne 2 (Brut)', 'Point X0'}, '
    Location', 'Best');
xlabel('R (Ohm)'); ylabel('X (Ohm)'); zlabel('Densite');
title('Histogrammes interpoles AVANT lissage');
grid on;

% Apres lissage
subplot(1,2,2);
surf(xx, yy, f1_smooth(xx, yy), 'FaceColor', 'g', 'FaceAlpha',
     0.5, 'EdgeAlpha', 0.2);
hold on;
surf(xx, yy, f2_smooth(xx, yy), 'FaceColor', 'r', 'FaceAlpha',
     0.5, 'EdgeAlpha', 0.2);
scatter3(0, 0, 0, 'r', 'filled'); % Point X0
hold off;
legend({'Ligne 1 (Lisse)', 'Ligne 2 (Lisse)', 'Point X0'}, '
    Location', 'Best');
xlabel('R (Ohm)'); ylabel('X (Ohm)'); zlabel('Densite');
title('Histogrammes interpoles APRES lissage');
grid on;

%% Sauvegarde des variables
save('f1.mat','f1_smooth');
save('f2.mat','f2_smooth');
```

Listing 2 – Code MATLAB pour la génération des densités de probabilité via une simulation de Monte Carlo

8.2 Annexe 2 : Algorithme Déterministe

```
1
2  clc; clear ; close all ; path=pwd();
3  [s,e]=regexp(path,'.+(?=[\\/]{1,2}shapOpt)');
4  path=path(s:e); addpath(genpath(path)); clear s e path
5  %% Initialisation
6  f1 = loadDDP1();
7  f2 = loadDDP2();
8
9
10
11 % Forme
12 nb = 20; % Nombre de poits de la frontiere.
13 noise = 0.5; % Bruit sur les ordonnees des points initiaux de
       la frontiere
14 absopt = linspace(-1,10,nb); % On fixe les abcsisses des
       points de la frontiere
15
16 % Fonction objectif
17 omega = 0.90; % w*VP + (1-w)*VN
18 omega2 = 0.08; % optim par rapport au perimetre ?
19 omega3 = 0.85; % optim avec FN ?
20 beta = 300 ; %Force de la contrainte sur FP et FN. Entre 150
       et 400
21 eps = 0.01;
22
23 %MultiStarts
24 nStarts=80;
25
26 %petit polygone
27 Y0=30;
28
29
30 %% Fonctions objectif et contraintes
31 function [f, g] = obj(xyOpti, f1, f2, absopt, omega, omega2,
       omega3, beta, eps)
32 % objective function
33
34 v_petit = [-1,3,absopt(end:-1:1);
35     -5,-5,xyOpti(end:-1:1)];
36
37 v_grand = [25,-1,absopt;
38     80,80,xyOpti];
39
```

```
40
41 try % try/Catch necessaire pour forcer l'algo a ne pas s'
       arreter quand il fait s'intersecter les cotes des polygones
42
43
44     sh_petit = myPolygon(reshape(v_petit,2,[]));
45     sh_grand = myPolygon(reshape(v_grand,2,[]));
46     P_petit = sh_petit.perim();
47     P_grand = sh_grand.perim();
48
49     VP = sh_petit.int(f1);
50     FN = sh_grand.int(f1);
51     FP = sh_petit.int(f2);
52     VN = sh_grand.int(f2);
53
54
55
56     f =omega2*(P_petit) + (1-omega2)*( -(omega)*VP - (1-omega)
          *VN + omega3*beta / 2 * (FP - eps)^2 + (1-omega3)*beta
          /2 * (FN- eps)^2 );
57
58 catch; f = NaN; g = NaN(size(xyOpti));
59 end
60 end
61
62 function [g, h]= constr(xyOpti)
63 g=0;
64 h=0;
65 end
66
67
68 %% Algorithms parameters
69 algo = "interior-point";
70 display = 'final';
71 sdgrad = false; % permet de resoudre avec les differences
       finies
72 maxfeval = 1000;
73 maxiter = 100;
74 options = optimoptions('fmincon', ...
75     'Display', display, ...
76     'Algorithm', algo, ...
77     'SpecifyObjectiveGradient', sdgrad, ...
78     'MaxFunctionEvaluations', maxfeval,...
79     'MaxIterations', maxiter,...
80     'ConstraintTolerance', 1e-6,...
```

```
    'ScaleProblem', true,...
    'EnableFeasibilityMode', true,...
    'SubproblemAlgorithm', "cg",...
    'StepTolerance', 1e-8,...
    'FunctionTolerance', 1e-8);

%% Algorithme d'optimisation
tic;

%une ligne par start
nVariables=nb;
x0_multi = zeros(nStarts,nVariables) ;

xopt_multi = zeros(nStarts,nVariables);
fopt_multi = zeros(nStarts,1);
exitflag_multi =  zeros(nStarts,1);


parfor cpt = 1:nStarts
    disp(strcat("Progress ",num2str(cpt/nStarts*100)," %")) % Affichage de l'avancement
    x0 = Y0.*ones(1,nb).*(1-noise +2*noise*rand(1,nb));
    x0_multi(cpt,:) = x0;

    % Definition de la fonction objectif en passant les arguments
    % (necessaire a cause de la parallelisation
    myObj = @(x) obj(x, f1, f2, absopt, omega, omega2, omega3, beta, eps);

    problem = struct();
    problem.options = options;
    problem.objective = myObj;

    while isnan(problem.objective(x0))
    x0 = Y0.*ones(1,nb).*(1-noise +2*noise*rand(1,nb));

    disp("pt invalide")
    end

    x0_multi(cpt,:) = x0;
    problem.x0=x0;
    problem.solver = 'fmincon';

    problem.nonlcon = @constr;
```

```
123     [xopt,fopt,exitflag] = fmincon(problem);
124     xopt_multi(cpt,:) = xopt;
125     fopt_multi(cpt,:) = fopt;
126     exitflag_multi(cpt)=exitflag;
127 end
128 t = toc; %Temps de calcul
129
130 %% Recuperation de la solution
131 [fopt_multi_sorted, idx] = sort(fopt_multi); % Trie V et
       recupere les indices de tri
132 xopt_multi_sorted = xopt_multi(idx, :); % Rearrange M en
       fonction des indices tries
133 sol = xopt_multi_sorted(1,:);
134
135
136 %% Affichage des valeurs des Objectifs
137
138 plot(sort(fopt_multi));
139 grid on;
140 ylabel("Objective function");
141 title("Repartition of the results")
142
143
144 %% Affichage de la solution
145
146 sol_petit = [-1,3,absopt(end:-1:1);
147              -5,-5,sol(end:-1:1)];
148
149 sol_grand = [25,-1,absopt;
150             80,80,sol];
151
152 shpSol_petit = myPolygon(sol_petit);
153 shpSol_grand = myPolygon(sol_grand);
154
155 polysh_petit=polyshape(sol_petit');
156 polysh_grand=polyshape(sol_grand');
157
158 x=linspace(-10,30,100);
159 y=linspace(-10,100,100);
160 [X,Y] = meshgrid(x,y);
161
162 polysh_petit.plot;hold on
163 polysh_grand.plot;
164 % surf(X,Y,f1(X,Y),'facecolor','g','facealpha',0.3, 'edgealpha
       ',0.2,'displayname','f1'); hold on
```

```
% surf(X,Y,f2(X,Y),'facecolor','r','facealpha',0.3, 'edgealpha',0.2,'displayname','f2'); hold off
% view(-40,40);
contour(X,Y,f2(X,Y),18,'r');
contour(X,Y,f1(X,Y),10,'g');

hold off;
axis normal
title(['Optimized Shape (t = ', num2str(t,3), ' s)'])


VP = shpSol_petit.int(f1);
FN = shpSol_grand.int(f1);
FP = shpSol_petit.int(f2);
VN = shpSol_grand.int(f2);

disp(strcat("VP = ",num2str(VP)))
disp(strcat("FP = ",num2str(FP)))
disp(strcat("VP2 = ",num2str(VN)))
disp(strcat("FN = ",num2str(FN)))
disp(strcat("Perimetre zone 1 = ",num2str(shpSol_petit.perim())))
disp(strcat("Grand P = ",num2str(shpSol_grand.perim())))


%% Chargmement des DDP obtenues par MC
function f1 = loadDDP1()
load('f1_4.mat')
f1=f1_smooth;
end

function f2 = loadDDP2()
load('f2_4.mat');
f2=f2_smooth;
end
```

Listing 3 – Code MATLAB pour l'optimisation de la forme du polygone en utilisant un algorithme déterministe.

8.3 Annexe 3 : Points des polygones optimisés.

8.3.1 Polygones de la littérature

```
zone_1 = [0, 13.09 , 15.6 , -12.3 ;
          0, -1.19 , 26.4 ,  26.4 ];
zone_2 = [0, 19.64 , 23.4 , -18.47;
          0, -1.78 , 39.6 , 39.6 ];
```

8.3.2 Exemples de polygones obtenus avec fmincon

```
absopt = linspace(-1,10,20);
solu = [18.9085 22.2563 24.7384 25.4458 26.4435 26.8806
   27.0428 27.1045 27.2190 27.2552 27.2886 27.4259 27.4730
   27.5425 27.4554 27.3389 26.9984 26.3434 25.5472 22.7413];
sol_petit = [0,3,absopt(end:-1:1);
             0,0,solu(end:-1:1)];
```

8.3.3 Exemples de polygones obtenus avec NSGAII

```
zone_1 = [-1.0000 , -5.0000, 3.0000 , -5.0000, 15.0000 ,
   27.0000, 10.6667, 22.7399 , 8.3333   26.0058, 6.0000
   32.9578, 3.6667   34.4220, 1.3333, 28.1524 , -1.0000 ,
   29.0000]

zone_1 = [-1.0000 , -5.0000 , 3.0000 , -5.0000 , 15.0000 ,
   27.0000 , 10.6667, 27.9954 , 8.3333 , 26.9707 , 6.0000 ,
   27.3208 , 3.6667 , 27.8544 , 1.3333 , 27.1380 , -1.0000 ,
   29.0000]

zone_1 = [-1.0000 , -5.0000 , 3.0000 , -5.0000, 15.0000 ,
   27.0000 , 10.6667 , 25.7997 , 8.3333 , 19.3683 , 6.0000 ,
   27.2052 , 3.6667   26.2720 , 1.3333 , 27.1691 , -1.0000 ,
   29.0000]
```

8.4 Annexe 4 : Algorithme Génétique

8.4.1 Algorithme Génétique Pareto

```
%% Test optim Ines

addpath(genpath('C:\Users\Ines Ben Cherifa\Desktop\St7\st7-shapopt-grid-protection\shapOpt\src'));
addpath(genpath('C:\Users\Ines Ben Cherifa\Desktop\St7\st7-shapopt-grid-protection\shapOpt\data'));
clear ; close all ; path=pwd();
[s,e]=regexp(path,'.+(?=[\\/]{1,2}shapOpt)');
path=path(s:e); addpath(genpath(path)); clear s e path

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%1) DDP

% declaration des densit s f1 et f2

f1= loadDDP1();
f2 = loadDDP2();


%%%%%%%%%2) Optimisation

nb = 6; % le nombre de points de la frontiere
absopt = linspace(-1, 20, nb);

lb = zeros(1, nb); % Creation du vecteur des bornes inferieures, qui va permettre de generer des polygones qui n'ont pas de cotes qui s'interceptent

for i = 0:nb-1  % La borne inferieure depend du point de la frontiere considere
    valeur_test = (-1 + i * 26 / (nb - 1));

    if valeur_test < 3
        lb(i+1) = -5 + 1e-6;
    else
        lb(i+1) = (35 / 22) * valeur_test - (212 / 22) + 1e-6;
    end
end

ub = 40*ones(1,nb); % Borne superieure fixe
options = optimoptions('gamultiobj', 'PopulationSize', 400,
    ...
```

```
    'MaxGenerations', 50, ...
    'PlotFcn', {@gaplotpareto}, ...
    'Display', 'iter',...
    'UseParallel',true);
% modif pour parallelisation
myObj = @(x) multi_objective(x, nb, f1, f2);

[x, fval] = gamultiobj(myObj, nb, [], [], [], [], lb, ub,
   options);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 3) Traces

figure;
ax1 = subplot(1,2,1);
% scatter3(fval(:,1), fval(:,2), fval(:,3), 50, 'b', 'filled')
   ;
scatter(fval(:,1), fval(:,2), 50, 'b', 'filled');
title('Front de Pareto (3D)');
xlabel('Vrai positif');
ylabel('Faux positif');
% zlabel('Vrai negatif');
grid on;
hold on;
% view(3); % Affichage en 3D
% rotate3d(ax1, 'on'); % Activation de la rotation

% Subplot droite : Polygone
ax2 = subplot(1,2,2);
% subplot(1,2,2);
title('Polygone associe');
axis equal;
hold on;
%view(3); % Activer la vue 3D pour l'affichage
rotate3d(ax2, 'on');

% Attente de clics utilisateur uniquement sur le front de
   Pareto
while true
    % Attendre un clic
    waitforbuttonpress; % Attend un appui (clic ou touche
       clavier)

    % Verifier si le clic a eu lieu dans ax1 (subplot gauche)
    if gca == ax1
```

```
77          % Recuperer les coordonnees du clic
78          point = get(gca, 'CurrentPoint');
79          x_click = point(1,1);
80          y_click = point(1,2);
81
82          % Trouver le point du front le plus proche
83          distances = sqrt((fval(:,1) - x_click).^2 + (fval(:,2)
                - y_click).^2);
84          [~, idx] = min(distances);
85
86          % supprimer le dernier point
87          try
88          delete(h2)
89          end
90
91          % Mettre en evidence le point selectionne en rouge
92          h2 = scatter(fval(idx,1), fval(idx,2), 100, 'r', '
               filled');
93
94          % Extraire les coordonnees du polygone
95          sol = x(idx,:);  % Seuls les points de la frontiere
               sont optimises
96
97          sol_petit = [-1,3, absopt(end:-1:1); -5,-5 ,flip(sol)
               ];
98          % Fixer le 3e point de sol_petit a (20, 30)
99       sol_petit(1, 3) = 25; % Coordonnee x du 3e point
100      sol_petit(2, 3) = 30; % Coordonnee y du 3e point
101
102     sol_petit(1, end) = -1; % Coordonnee x du dernier point
103      sol_petit(2, end) = 27; % Coordonnee y du dernier point
104         x1 = sol_petit';
105
106         disp(x1)
107
108         forme1 = myPolygon(x1);
109
110
111         sol_grand = [25,-1,absopt;80,80,sol];
112
113 % Fixer l'avant-dernier point de sol_grand a (20, 30)
114 sol_grand(1, end) = 25; % Coordonnee x de l'avant-dernier
       point
115 sol_grand(2, end) = 30; % Coordonnee y de l'avant-dernier
       point
```

```
116 sol_grand(1, 3) = -1; % Coordonnee x du 3eme point
117 sol_grand(2, 3) = 27; % Coordonnee y du 3eme point
118
119         x2 = sol_grand';
120
121         forme2 = myPolygon(x2);
122
123
124         J1 = forme1.int(f1);
125         J2 = forme1.int(f2);
126         J3 = forme2.int(f2);
127         J4= forme2.int(f1);
128
129
130         % Afficher le polygone correspondant
131         subplot(ax2); % Se placer sur le bon subplot
132         cla;
133         hold on;
134
135         xx=linspace(-10,30,100);
136         y=linspace(-10,100,100);
137
138         [X,Y] = meshgrid(xx,y);
139         contour(X,Y,f2(X,Y), 10,'r');
140         contour(X,Y,f1(X,Y), 10,'g');
141         scatter(0,0,'r','filled','displayname','X0');%hold off
142         legend(); grid on;
143
144         sgtitle(sprintf('Le taux de vrai positif : %.2f\nLe
                taux faux positif : %.5f\nLe taux de vrai negatif :
                 %.2f\nLe taux de faux n gatif : %.2f', J1, J2,J3,
                J4));
145         axis normal
146         polysh_petit=polyshape(x1);
147         polysh_petit.plot;
148         polysh_grand=polyshape(x2);
149         polysh_grand.plot;
150         hold off
151     end
152 end
153
154
155
156
157 %% tracer une courbe ROC
```

```
158
159 figure;
160 hold on; % Permet de tracer plusieurs courbes sur le meme
    graphique
161 xlabel('Taux Faux positifs');
162 ylabel('Taux Vrais positifs');
163 title('Courbe ROC');
164 T2_vect = []; % Vecteur pour stocker tous les faux positifs
165 T1_vect = [];
166
167 for i = 1:size(x, 1) % Parcourir chaque ligne de la matrice x
168     sol = x(i, :);
169         sol_petit = [-1,3, absopt(end:-1:1); -5,-5 ,flip(sol)
               ];
170         % Fixer le 3e point de sol_petit a (20, 30)
171     sol_petit(1, 3) = 20; % Coordonnee x du 3e point
172     sol_petit(2, 3) = 30; % Coordonnee y du 3e point
173 %
174     sol_petit(1, end) = -1; % Coordonnee x de l'avant-dernier
          point
175 sol_petit(2, end) = 27; % Coordonnee y de l'avant-dernier
    point
176     x1 = sol_petit';
177     forme1 = myPolygon(x1);
178
179
180       sol_grand = [25,-1,absopt;80,80,sol];
181
182 % Fixer l'avant-dernier point de sol_grand a (20, 30)
183 sol_grand(1, end) = 20; % Coordonnee x de l'avant-dernier
    point
184 sol_grand(2, end) = 30; % Coordonnee y de l'avant-dernier
    point
185 sol_grand(1, 3) = -1; % Coordonnee x du 3eme point
186 sol_grand(2, 3) = 27; % Coordonnee y du 3eme point
187     x2 = sol_grand';
188     forme2 = myPolygon(x2);
189     J1 = forme1.int(f1);
190     J2 = forme1.int(f2);
191     J3 = forme2.int(f2);
192     J4= forme2.int(f1);
193     T1= J1/(J1+J4);
194     T2=J2/(J2+J3);
195     T2_vect = [T2_vect, T2];
196     T1_vect = [T1_vect, T1];
```

```
197     plot(T2, T1, 'o', 'DisplayName', sprintf('Ligne %d', i));
198 end
199 
200 [T2_sorted, idx] = sort(T2_vect);
201 T1_sorted = T1_vect(idx); % Reorganiser T1 en fonction de T2
       trie
202 
203 % Calculer l'aire sous la courbe (AUC) avec trapz
204 AUC = trapz(T2_sorted, T1_sorted);
205 AUC= 1- AUC;
206 title(sprintf('Courbe ROC (AUC = %.4f)', AUC));
207 text(0.6, 0.1, sprintf('AUC = %.4f', AUC), 'FontSize', 12, '
       FontWeight', 'bold', 'Color', 'red');
208 
209 hold off;
210 
211 
212 
213 
214 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
215 %% Fonctions utiles
216 
217 %% A) DDP
218 
219 function f1 = loadDDP1()
220 load('f1_4.mat')
221 f1=f1_smooth;
222 end
223 
224 function f2= loadDDP2()
225 load('f2_4.mat');
226 f2=f2_smooth;
227 end
228 
229 %% B) Optimisation
230 
231 function [J, ConstrVals] = multi_objective(x, nb, f1, f2)
232     %nb =6;
233     absopt = linspace(-1,20,nb);
234 
235      sol = x;  % Seuls les points de la frontiere sont
           optimises
236 
237 sol_petit = [-1,3, absopt(end:-1:1);
238              -5,-5 ,flip(sol)];
```

```
239
240 sol_petit(1, 3) = 25; % Coordonnee x du 3e point
241 sol_petit(2, 3) = 30;
242
243 sol_petit(1, end) = -1; % Coordonnee x de l'avant-dernier
    point
244 sol_petit(2, end) = 27; % Coordonnee y de l'avant-dernier
    point
245 % x1 = order_points(sol_petit');
246 forme1 = myPolygon(sol_petit',2);
247
248 sol_grand = [25,-1,absopt;80,80,sol];
249
250 sol_grand(1, end) = 25; % Coordonnee x de l'avant-dernier
    point
251 sol_grand(2, end) = 30; % Coordonnee y de l'avant-dernier
    point
252 sol_grand(1, 3) = -1; % Coordonnee x du 3eme point
253 sol_grand(2, 3) = 27; % Coordonnee y du 3eme point
254
255 % x2 = order_points(sol_grand');
256 forme2 = myPolygon(sol_grand',2);
257
258     J1 = forme1.int(f1); % VP (?)
259     J2 = forme1.int(f2); % FP (?)
260     J3 = forme2.int(f1); % FN (?)
261     J4 = forme2.int(f2); % VN (?)
262     P_petit = forme1.perim();
263     T1= J1/(J1+J4);       % VP normalise
264     T2= J2/(J2+J3);
265     % J = [-0.6*T1,0.4*T2]; % avec ponderation
266     J = [-T1,T2];  % sans ponderation
267
268     % ConstrVals = [P_petit-345];
269     ConstrVals = []; % pas de contraintes sur le perimetre
270 end
```

Listing 4 – Code MATLAB, optimisation de la forme en utilisant un algorithme génétique

8.4.2 Algorithme Génétique avec la fonction objectif de la méthode déterministe

```
addpath(genpath('C:\Users\Ines Ben Cherifa\Desktop\St7\st7-shapopt-grid-protection\shapOpt\src'));
clear ; close all ; path=pwd();
[s,e]=regexp(path,'.+(?=[\\/]{1,2}MainScripts)');
path=path(s:e); addpath(genpath(path)); clear s e path
%%
f1= loadDDP1();
f2 = loadDDP2();

function f1 = loadDDP1()
load('f1_4.mat')
f1=f1_smooth;
end

function f2= loadDDP2()
load('f2_4.mat');
f2=f2_smooth;
end

%%

nb = 20;
absopt = linspace(-1, 20, nb);

lb = zeros(1, nb); % Initialisation du vecteur

for i = 0:nb-1
    valeur_test = (-1 + i * 26 / (nb - 1));

    if valeur_test < 3
        lb(i+1) = -5 + 1e-6; % i+1 car MATLAB indexe a partir de 1
    else
        lb(i+1) = (35 / 22) * valeur_test - (212 / 22) + 1e-6;
    end
end

ub = 50*ones(1,nb); % Borne superieure fixe
options = optimoptions('gamultiobj', 'PopulationSize', 300, 'MaxGenerations', 50, 'Display', 'iter');
myObj = @(x) multi_objective(x, nb, f1, f2);

```

```
[x, fval] = gamultiobj(myObj, nb, [], [], [], [], lb, ub,
   options);

function [J, ConstrVals] = multi_objective(x, nb, f1, f2)
    omega = 0.95; % w*VP + (1-w)*VN
    omega2 = 0.08; %optim par rapport au perimetre ?
    omega3 = 0.85; %optim avec FN ?
    beta = 250;
    eps = 0.01;
    absopt = linspace(-1,20,nb);
    sol=x;
    v_petit = [-1,3, absopt(end:-1:1);
            -5,-5 ,flip(sol)];
    v_petit(1, 3) = 20;
    v_petit(2, 3) = 30;
    v_petit(1, end) = -1; % Coordonnee x de l'avant-dernier
       point
v_petit(2, end) = 31; % Coordonnee y de l'avant-dernier point

    v_grand =[25,-1,absopt;80,80,sol];
    %Reorganiser les sommets
    v_grand(1, end) = 20; % Coordonnee x de l'avant-dernier
       point
v_grand(2, end) = 30; % Coordonnee y de l'avant-dernier point
v_grand(1, 3) = -1; % Coordonnee x de l'avant-dernier point
v_grand(2, 3) = 31; % Coordonnee y de l'avant-dernier point

    sh_petit = myPolygon(reshape(v_petit,2,[]));
    sh_grand = myPolygon(reshape(v_grand,2,[]));
    P_petit = sh_petit.perim();
    P_grand = sh_grand.perim();


    VP = sh_petit.int(f1); %vrai positif
    FN = sh_grand.int(f1);%-sh_petit.int(@f1); %faux negatifs
    FP = sh_petit.int(f2);
    VN = sh_grand.int(f2);%-sh_petit.int(@f2);


    % disp(strcat("VP = ",num2str(VP)))
    % disp(strcat("FP = ",num2str(FP)))
    % disp(strcat("VN = ",num2str(VN)))
    % disp(strcat("FN = ",num2str(FN)))

```

```
    f =omega2*(P_petit) + (1-omega2)*( -(omega)*VP - (1-omega)*VN + omega3*beta / 2 * (FP - eps)^2 + (1-omega3)*beta /2 * (FN- eps)^2 );

    J = f;

    ConstrVals = [];
    end
%%
figure;
    sol = x(1,:);  % Seuls les points de la frontiere sont optimises

        sol_petit = [-1,3, absopt(end:-1:1); -5,-5 ,flip(sol)];
        % Fixer le 3e point de sol_petit a (20, 30)
    sol_petit(1, 3) = 20; % Coordonnee x du 3e point
    sol_petit(2, 3) = 30; % Coordonnee y du 3e point
       sol_petit(1, end) = -1; % Coordonnee x de l'avant-dernier point
sol_petit(2, end) = 32; % Coordonnee y de l'avant-dernier point
        x1 = sol_petit';

        forme1 = myPolygon(x1,2);


        sol_grand = [25,-1,absopt;80,80,sol];

% Fixer l'avant-dernier point de sol_grand a (20, 30)
sol_grand(1, end) = 20; % Coordonnee x de l'avant-dernier point
sol_grand(2, end) = 30; % Coordonnee y de l'avant-dernier point
sol_grand(1, 3) = -1; % Coordonnee x de l'avant-dernier point
sol_grand(2, 3) = 32; % Coordonnee y de l'avant-dernier point

        x2 = sol_grand';

        forme2 = myPolygon(x2,2);

        J1 = forme1.int(f1);
        J2 = forme1.int(f2);
        J3 = forme2.int(f2);
        J4= forme2.int(f1);
```

```
        xx=linspace(-10,30,100);
        y=linspace(-10,100,100);
hold on
        [X,Y] = meshgrid(xx,y);

        contour(X,Y,f1(X,Y), 10,'g');
        contour(X,Y,f2(X,Y), 10,'r');
        scatter(0,0,'r','filled','displayname','X0');%hold off
        legend(); grid on;

        sgtitle(sprintf('Le taux de vrai positif : %.4f\nLe
           taux faux positif : %.4f\nLe taux de vrai n gatif
           : %.4f\nLe taux de faux negatif : %.4f', J1, J2,J3,
           J4));
        axis normal
        polysh_petit=polyshape(x1);
        polysh_petit.plot;
        polysh_grand=polyshape(x2);
        polysh_grand.plot;
        hold off
```

Listing 5 – Code MATLAB

8.4.3 Morceau de code à changer afin de modifier le sommet fixe

```
absopt = linspace(-1, x_sommet-2, nb);

x_sommet= 15;
y_sommet=27;
a=(-5-y_sommet)/(3-x_sommet);
b= y_sommet - a*x_sommet;

lb = zeros(1, nb); % Initialisation du vecteur
for i = 0:nb-1
    valeur_test = (-1 + i * (x_sommet + 1) / (nb - 1));

    if valeur_test < 3
        lb(i+1) = -5 + 1e-6; % i+1 car MATLAB indexe a partir
           de 1
    else
        lb(i+1) = a * valeur_test + b + 1e-6;
    end
end
```

Listing 6 – Code MATLAB

8.5 Annexe 5 : Autre approche pour garantir la convexité des polygones

On a rapidement pensé à considérer d'autres paramètres que le périmètre seul pour obtenir une figure convexe. Notamment le rapport $\frac{perimetre}{volume}$ du polygone. En effet, en minimisant ce paramètre, on maximise le volume en minimisant le périmètre avec, pour rapport minimal, une figure proche d'un disque.

Cette expression a le mérite d'avoir un gradient explicitement calculable à partir des dérivées de formes :

$$\nabla \cdot \frac{p}{V} = \frac{p\nabla \cdot V - V\nabla \cdot p}{V^2}$$

Ainsi, on peut directement changer la fonction objectif de notre optimisation tout en gardant une résolution par dérivées de formes pour limiter le temps de calcul. Au préalable, bien que la fonction ne soit pas trop complexe, les résultats donnés par ce gradient ont été comparés à ceux obtenus par différences finies.

Le code a donc été modifié de sorte que ce rapport remplace le périmètre dans la fonction objectif :

```
function [f, Ex, Ey, Sx, Sy] = f1(x,y)
% 2D normal distribution
Ex = 0; Sx = 0.2;
Ey = 1; Sy = 0.2;
fx = exp( -0.5*((x-Ex)/Sx).^2 ) / (Sx * sqrt(2*pi));
fy = exp( -0.5*((y-Ey)/Sy).^2 ) / (Sy * sqrt(2*pi));
f = fx .* fy ;
end

function g = grad_f1(x,y)
% gradient of f1
[f, Ex, Ey, Sx, Sy] = f1(x,y);
g = f .* [(2*Ex - 2*x)/(2*Sx^2), (2*Ey - 2*y)/(2*Sy^2)];
end


function [f, Ex, Ey, Sx, Sy] = f2(x,y)
% 2D normal distribution
```

```
Ex = 0; Sx = 0.2;
Ey = 1.3; Sy = 0.2;
fx = exp( -0.5*((x-Ex)/Sx).^2 ) / (Sx * sqrt(2*pi));
fy = exp( -0.5*((y-Ey)/Sy).^2 ) / (Sy * sqrt(2*pi));
f = fx .* fy ;
end

function g = grad_f2(x,y)
% gradient of f2
[f, Ex, Ey, Sx, Sy] = f2(x,y);
g = f .* [(2*Ex - 2*x)/(2*Sx^2), (2*Ey - 2*y)/(2*Sy^2)];
end


% plot
[x,y] = meshgrid(-1:0.05:1, -0.1:0.05:1.9);
surf(x,y,f1(x,y),'facecolor','g','facealpha',0.5, 'edgealpha'
    ,0.2,'displayname','f1'); hold on
surf(x,y,f2(x,y),'facecolor','r','facealpha',0.5, 'edgealpha'
    ,0.2,'displayname','f2');
scatter(0,0,'r','filled','displayname','X0'); hold off
legend(); grid on; title("Probability densities");

nCote = 10;
noise = 0;
radius = 0.9;
t = (2*pi/nCote:2*pi/nCote:2*pi)-pi/2; t = t(1:end-1);
R = radius * (1 + noise*(rand(1,nCote-1)-0.5));
X0 = [0;0]; center = X0 + [0; radius];
problem.x0 = R.*[cos(t);sin(t)] + center;
shpInit = myPolygon([X0, problem.x0]);
global r0
r0 = ratio(shpInit)
global p0
p0 = shpInit.perim()
shpInit.plot(); hold on
surf(x,y,f1(x,y),'facecolor','g','facealpha',0.3, 'edgealpha'
    ,0.2,'displayname','f1');
surf(x,y,f2(x,y),'facecolor','r','facealpha',0.3, 'edgealpha'
    ,0.2,'displayname','f2'); hold off
view(-40,40); axis normal

function [f, g] = obj(v)
% objective function
global r0
```

```
omega = 0; % optimal choice depends on number of points
beta = 1000;
eps = 0.01;
X0 = [0;0]; vall = [X0 , v]; % we fix X0
try
    sh = myPolygon(reshape(vall,2,[]));
    r = ratio(sh); F1 = sh.int(@f1); F2 = sh.int(@f2);
    % f = omega * r/r0 - (1-omega) * F1 + beta / 2 * (F2 - eps)^2 ;
    f = omega * sh.perim()/p0 - (1-omega) * F1 + beta / 2 * (F2 - eps)^2 ;f = omega * sh.perim()/p0 - (1-omega) * F1 + beta / 2 * (F2 - eps)^2 ;
    % g = omega *  grad_r(sh)/r0 - (1-omega) * gradF(sh,sh.dInt(@f1, @grad_f1)) ...
    %     + beta * (F2 - eps) * gradF(sh,sh.dInt(@f2, @grad_f2));
    % g = g(:); g = g(3:end);
catch; f = NaN; g = NaN(size(v));
end
end

function rapport = ratio(sh)

    % Calcul des grandeurs
    V = sh.vol();
    p = sh.perim();

    % Calcul du rapport
    rapport = p/V;
end

function gradient_ratio = grad_r(sh)
    % Calcul des d riv es de formes
    dV = sh.dVol();
    dp = sh.dPerim();

    % Calcul des gradients
    gradV = sh.grad(dV);
    gradp = sh.grad(dp);

    % Calcul des grandeurs
    V = sh.vol();
    p = sh.perim();

    % Calcul du gradient final
```

```
100     gradient_ratio = (V * gradp - p * gradV) / V^2;
101 end
102 
103 function gradient = gradF(sh,dF)
104     gradient = sh.grad(dF);
105 end
106 
107 algo = "interior-point";
108 display = 'iter';
109 sdgrad = false;
110 maxfeval = 1000;
111 maxiter = 100;
112 options = optimoptions('fmincon', ...
113     'Display', display, ...
114     'Algorithm', algo, ...
115     'SpecifyObjectiveGradient', sdgrad, ...
116     'MaxFunctionEvaluations', maxfeval,...
117     'MaxIterations', maxiter,...
118     'ConstraintTolerance', 1e-6,...
119     'ScaleProblem', true,...
120     'EnableFeasibilityMode', true,...
121     'SubproblemAlgorithm', "cg",...
122     'StepTolerance', 1e-8,...
123     'FunctionTolerance', 1e-8);
124 problem.options = options;
125 problem.solver = 'fmincon';
126 problem.objective = @obj;
127 
128 tic;
129 sol = fmincon(problem);
130 t = toc;
131 
132 shpSol = myPolygon([[0;0],sol]);
133 shpSol.plot();
134 hold on
135 surf(x,y,f1(x,y),'facecolor','g','facealpha',0.3, 'edgealpha'
    ,0.2,'displayname','f1');
136 surf(x,y,f2(x,y),'facecolor','r','facealpha',0.3, 'edgealpha'
    ,0.2,'displayname','f2'); hold off
137 view(-40,40); axis normal
138 title(['Optimized Shape (t = ', num2str(t,3), ' s)'])
139 subtitle(['int(F1) = ', num2str(shpSol.int(@f1),'%.2e'),...
140     ' | $\frac{p}{V}$ = (', num2str(ratio(shpSol),'%.2e)'),...
141     '$\Omega^{-1}$ | int(F2) = ', num2str(shpSol.int(@f2),'%.2
        e')], 'Interpreter', 'latex')
```

Listing 7 – Code MATLAB

Ce code permet, en partant d'un décagone régulier, d'obtenir un décagone convexe aux performances de détection comparables à ce qui a été trouvé en utilisant le périmètre pour garantir la convexité.

On notera qu'après plusieurs tentatives infructueuses, on a normalisé le rapport en le divisant par le rapport initial.

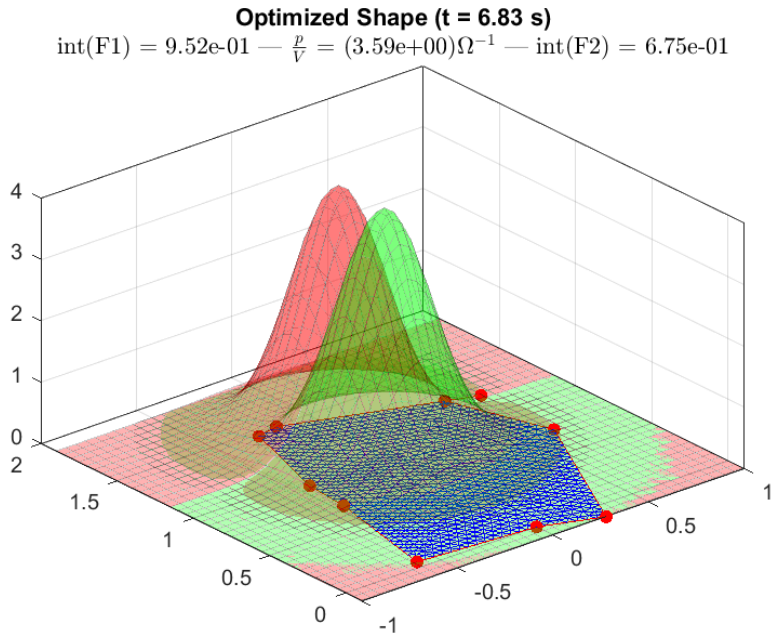


FIGURE 24 – Décagone obtenu avec la fonction objectif contenant le rapport $\frac{p}{V}$

Malheureusement, le code fourni prenant en compte le périmètre ne compile plus et ne permet pas d'obtenir une figure comparative.