**Calibration par deep-learning :**

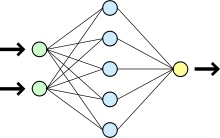
**Résumé :**

Nous avons ensuite essayé une approche moins standard de la calibration.Nous avons récolte un grand nombre d’exemple grâce a notre modèle ADS de simulation puis nous les avons utilisés pour l’entraînement de notre réseau neuronal grâce a un dérive de la méthode de la descente de gradient(Adam Optimizer). Afin de trouver une architecture efficace nous avons comparé plusieurs architecture différentes. Toutefois il a été complique d’explorer un grand nombre de structures car nous avions un nombre restreint d’exemple ce qui entre autre augmente le temps d’apprentissage et réduit son efficacité. Un autre facteur limitant a été le manque d’exemples réels que nous ne pouvons pas obtenir de manière automatiques en nombre assez grand. De ce fait nos résultats sont mitigés nous avons réussi a observer un grande progression au cours des différentes phases de l’apprentissage mais nous n’avons pas obtenus de résultats surpassant les méthodes classiques.

**Pour aller un peu plus loin:**

Ici nous allons rentrer dans la méthode d’exécution de ces stratégies et des clef de notre réussite relative et surtout les écueils que nous avons rencontrer.Nous essaierons d’expliquer tout les choix que nous avons faits et les raisons qui nous y ont amené. Pour

Fonctionnement de basse d’un réseau de neurones :



Un réseau de neurones est un ensemble de fonctions très simples(neurones) assemblées entres elles afin de former une fonction plus complexe. Cette architecture est inspirée de celle d’un cerveau, c’est de la qu’elle tire son nom.

C’est neurones sont assemblées en couche interconnectées entres elles. Chacune de ces neurones ont des paramètres que l’on peut ajuster cet ajustement est fait grâce à des exemples durant la phase appelé l’apprentissage. Cette phase utilise des méthodes dérivées de la descente de gradient dans notre cas la méthode Adam.

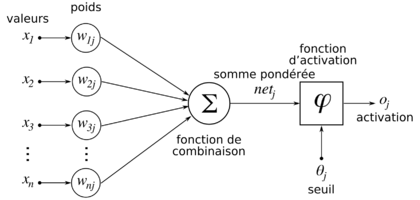
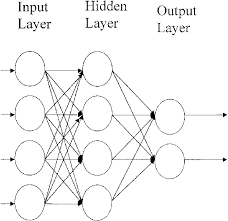


Schéma de fonctionnement d’un neurone.

Les paramètres ajustables par apprentissage sont multiples nous avons les seuils et les poids de chaque neurones. Nous pouvons aussi modifier la fonction d’activation de chaque neurones

Présentation du problème :

Nous avons donc quatre neurones d’entrées dans notre réseau qui représentes les quatre puissance de sortie de notre carte et 2 sorties qui représentent le rapport d’amplitude et le déphasage entre les 2 signaux.

Représentation du problème :

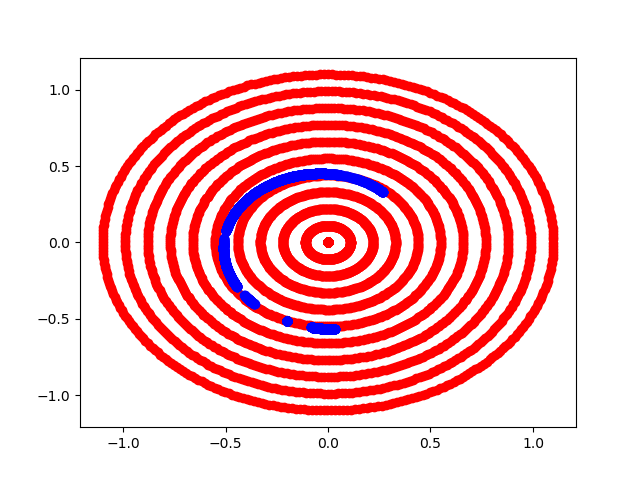
Afin de rendre notre interprétation des résultats plus aisée nous utiliserons une représentation polaire même si celle -ci ne correspond pas exactement au sorties de notre réseau. En effet la représentation la plus cohérente serait d’avoir l’amplitude sur un axe et le déphasage sur un autre. Mais par souci de cohérence au sein de ce rapport et pour une autre représentation nous resterons sur une représentation polaire.

Critère de performance :

Afin de pouvoir évaluer la performance de notre réseau de neurones, nous devons nous nous donner un critère de performance. Nous avons choisi la moyenne des écarts aux carré car elle prends en compte tout les écarts de tout les points et semble adapté à notre problème nous l’appellerons le loss afin de suivre la convention du domaine des réseaux neuronaux

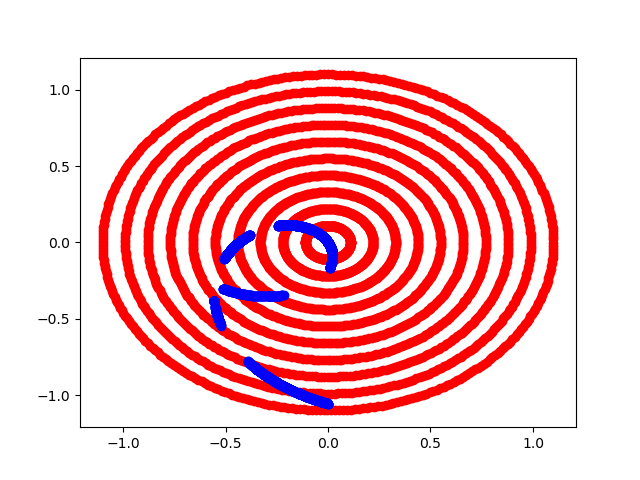
Choix de l’architecture du réseau :

Afin d’avoir une référence de performance pour nos prochain réseaux nous avons utilisé un réseau avec comme fonction d’activation une fonction d’activation l’identité.celui-ci devrai apprendre la meilleure régression linéaire et ainsi nous obtenons les résultats suivants.

  
Figure 1: réseau linéaire

Ce résultat s’approche de ce que obtiendrions avec une régression linéaire (moindres carrés)(figure 1).

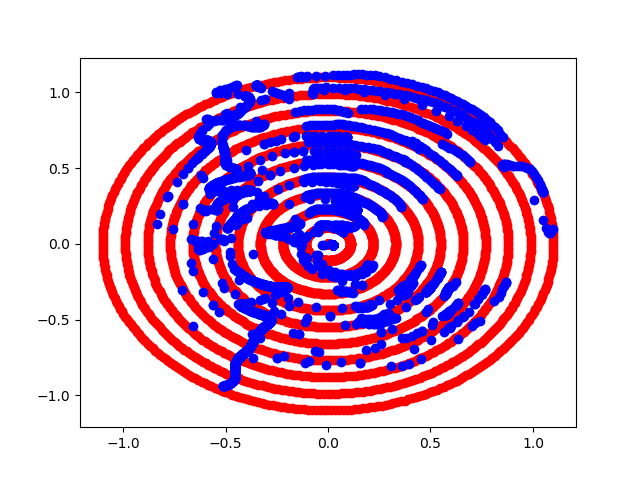
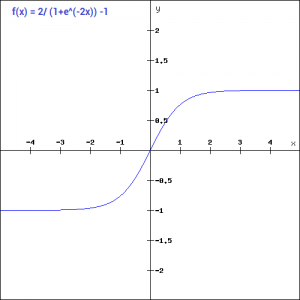
Ainsi nous avons des performance pour comparer nos autres réseaux.

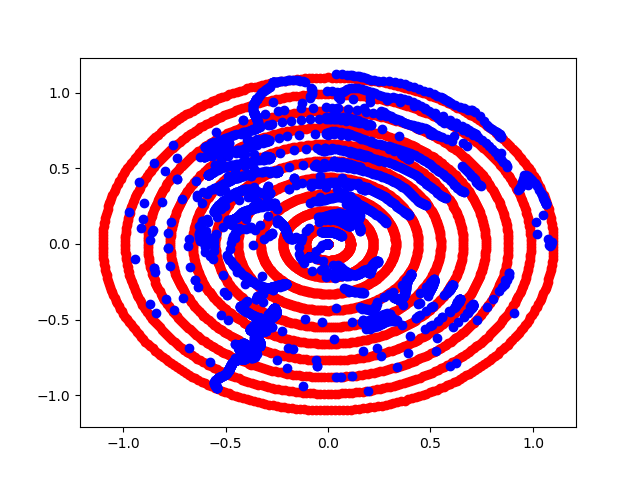
  
Figure 2: réseau non linéaire

Maintenant nous allons utiliser un réseau de même type mais en introduisant une fonction d’activation non linéaire et nous voyons que même après seulement quelques epoch (une epoch est un passage d’apprentissage sur toutes les données que nous avons ) le réseau ce déplace vers un état plus vraisemblable.

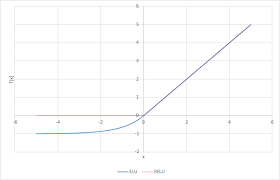
Nous pouvons nous intéresser aux différentes fonctions d’activations courantes voici rapidement les résultats obtenus par trois fonctions d’activation usuelles pour 21 000 epoch chacune.

Softsign :loss=0.28716





elu : loss=0.27736



relu :

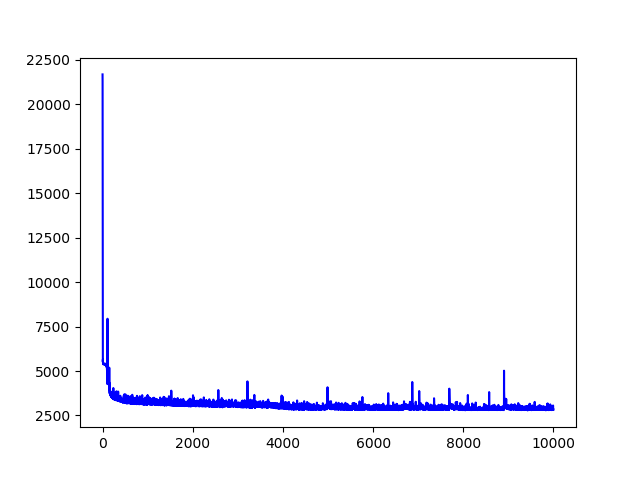
jjjj

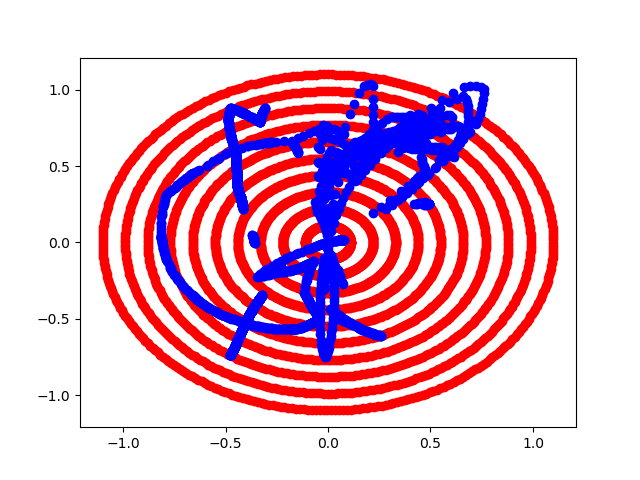
Ensuite nous nous sommes intéressé aux différentes mises en forme des données afin de faciliter l’apprentissage du réseau.

Mise en forme des données :

Afin d’obtenir les meilleurs résultats une pratique courante est de normaliser les entrées. C’est a dire de faire passer entrées dans une fonction qui annule la moyenne et rend l’écart type unitaire pour chaque entrée. Donc nous avons décidé de comparer les deux pratiques. Pour cella nous avons fait apprendre a deux réseaux un avec et un sans normalisation pendent 10 000 epoch. Voici les résultats.

Sans Normalisation





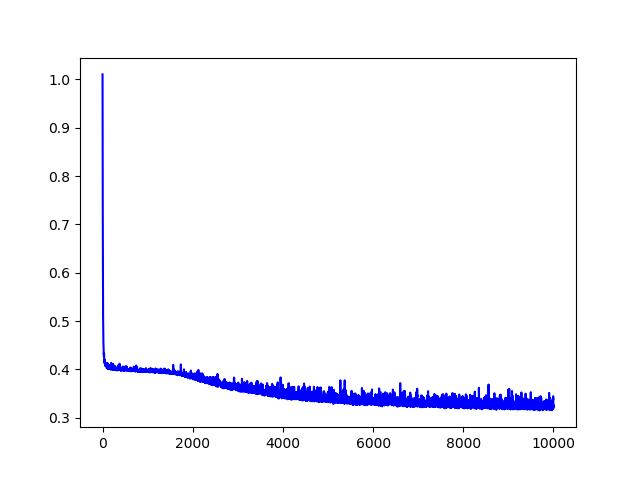
Ecart quadratique en fonction du nombre d’iterations.

Prédiction (en bleu)

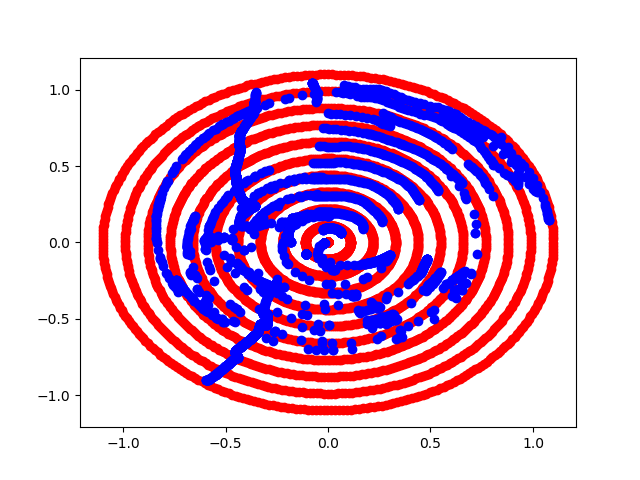
Valeurs attendu (en rouge)

Prédiction (en bleu)

Valeurs attendu (en rouge)



Avec Normalisation



Ecart quadratique en fonction du nombre d’iterations.

Prédiction (en bleu)

Valeurs attendu (en rouge)

Apprentissage :

L’apprentissage consiste a minimiser le loss pour cela nous utilisons la descente de gradient. Notre méthode d’apprentissage nous permet d’ajuster un paramètre qui représente la magnitude des changement que nous effectuons sur les paramètres du réseau. Ainsi si ce paramètre est grand le réseau va faire de grand changements imprécis alors que si ce paramètre est petit les paramètres du réseau aurons des changement plus petit mais plus précis. Ainsi le but et de commencer par une grande valeur et de la réduire en fonction de la précision que notre réseau atteint.

Problèmes :

Un des facteurs limitants de l’apprentissage a été le temps de calcul pour l’apprentissage en effet celui-ci varie de quelque milliseconde par epoch a quelques seconde en fonctions de nos paramètres d’apprentissage. Ceci a grandement réduit notre capacité a explorer un grand nombre d’architectures. Ainsi notre réseau avec le plus d’entraînement a mobilisé un pc plus de 60 heures même si nous entraînions notre réseau sur une carte graphique haut de gamme.

Un autre problème a été le manque d’exemple en effet nous avions un peu mois de 4000 exemples et ceux ci contenaient des symétries supplémentaires à celles du problème ce qui a pu fausser l’apprentissage.

Résultats :

Ainsi nos résultats n’ont pas été très concluant même avec beaucoup d’entraînement nos IA finissent par stagner. Et à la fin d’une centaine d’heure d’entraînement nous avons ces résultats :

Question ouvertes :

Pourquoi tout nos réseau nous ont une direction de prédilection pour l’apprentissage ?

**Calibration :**

1. minimiser les erreur par design
2. erreur toujours presentes
3. strategies de mitigations

recentrer

3 points

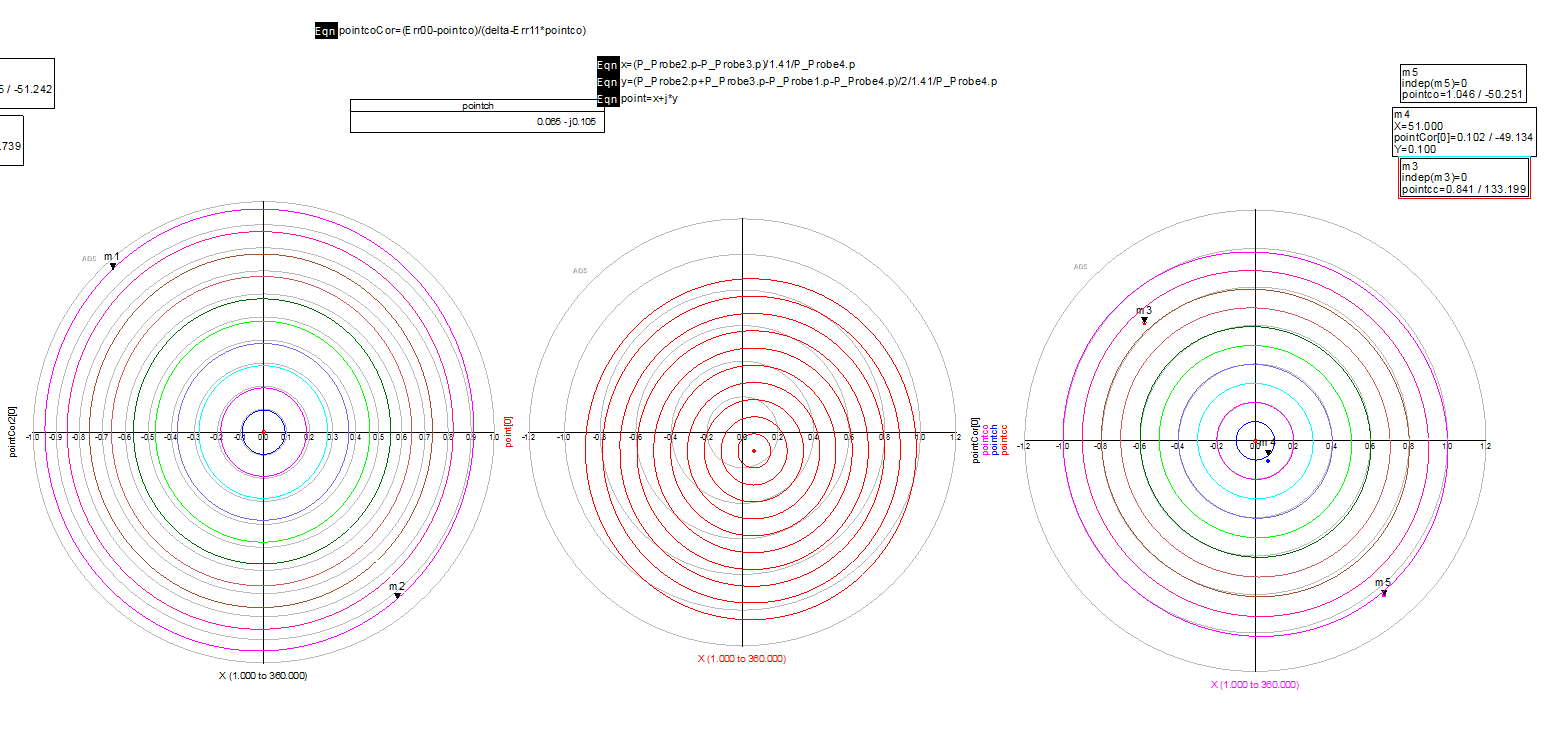
ia

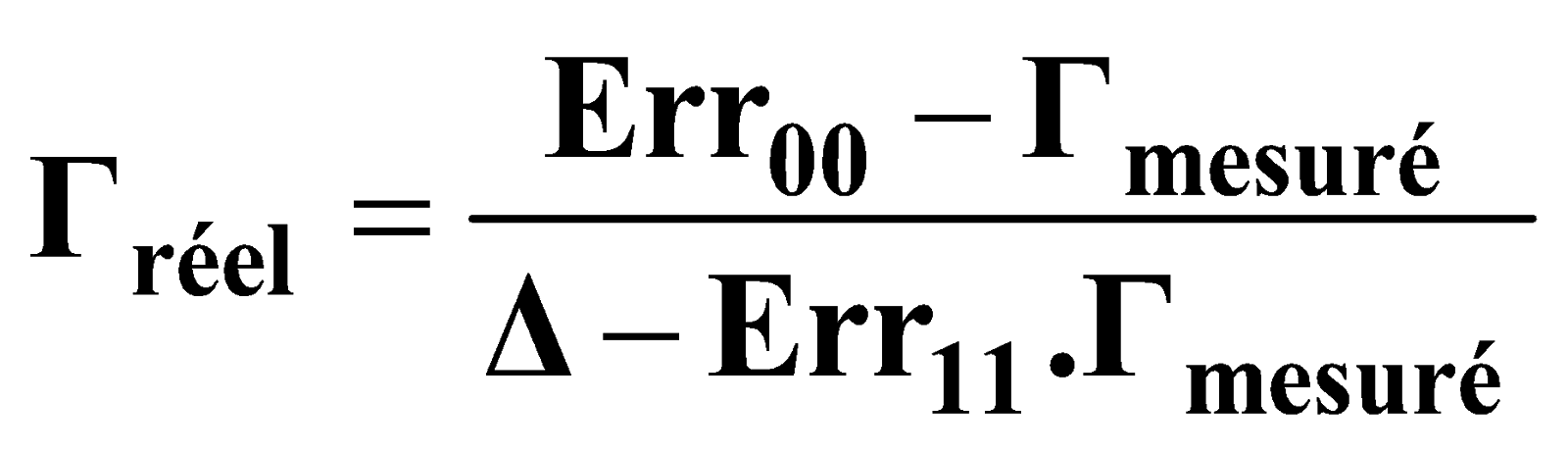
changement de point de vue

**Stratégies de calibration :**

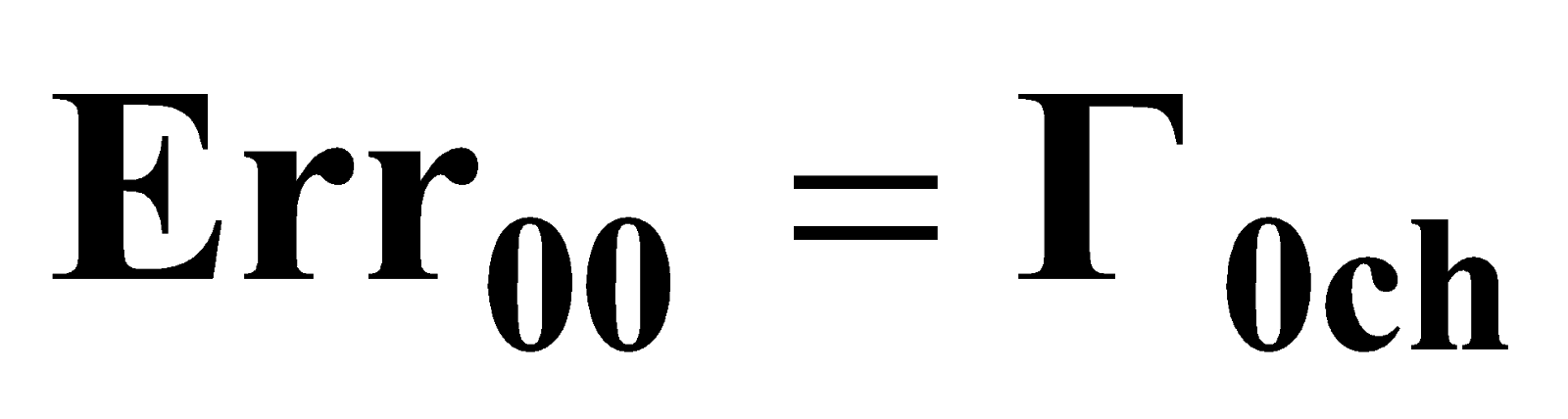
point de valeurs connues :

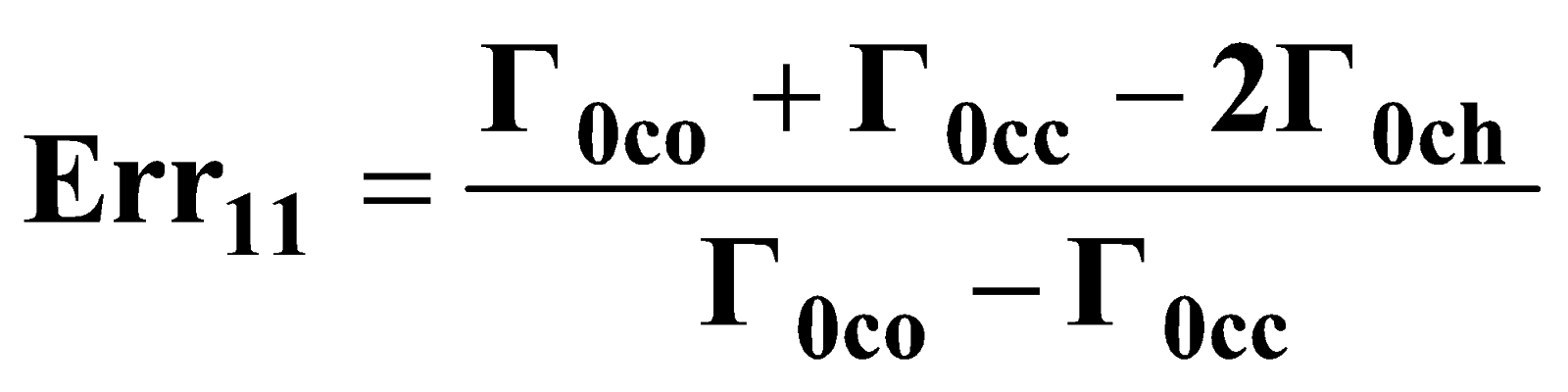
Nous pouvons corriger notre système grâce a un calibration c’est à dire grâce a des valeurs connues en effet une approche triviale et de mesurer 1 signal d’amplitude 1 sur le port B avec une charge 50Ω sur le port A. Ceci devrait nous donner le point (0,0) mais à cause des imperfections de notre réseau le point mesuré est différent de (0,0). Nous pouvons donc corriger ce décalage afin de réduire l’erreur. Mais comme nous pouvons le voir sur la figure suivante ce n’est pas suffisant. On peut observer une contraction en effet le cercle le plus extérieur n’est pas de module 1.

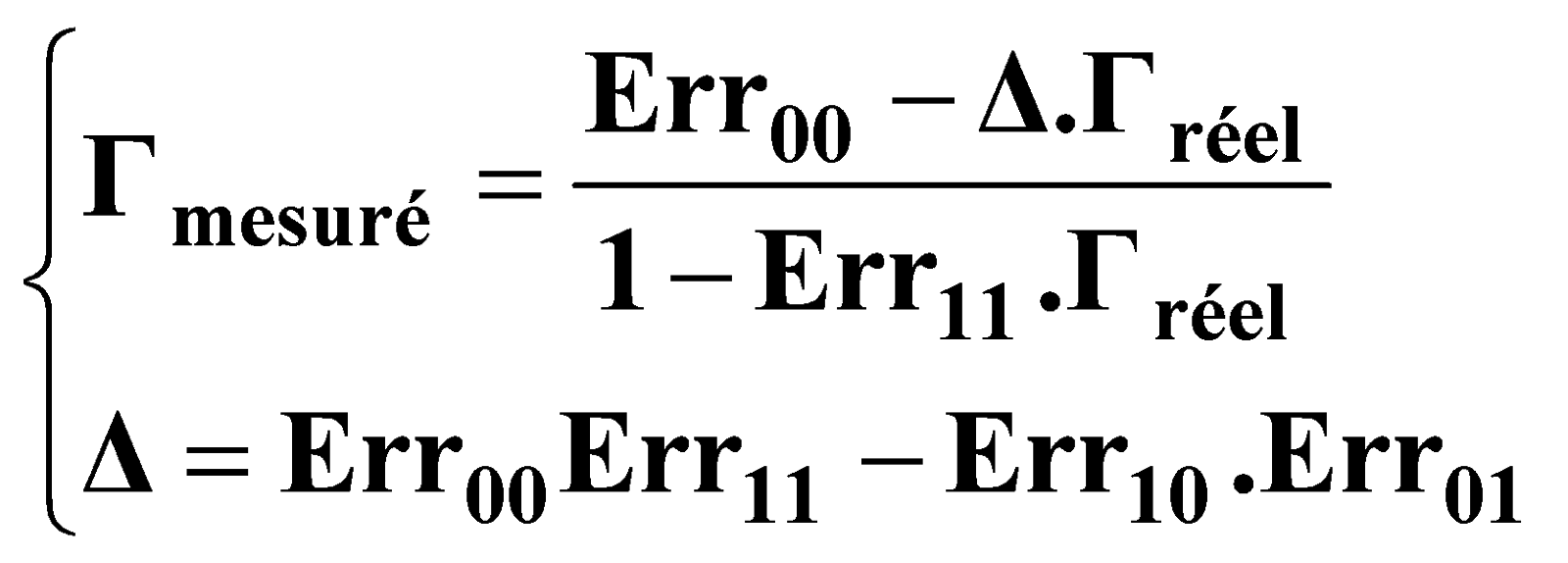
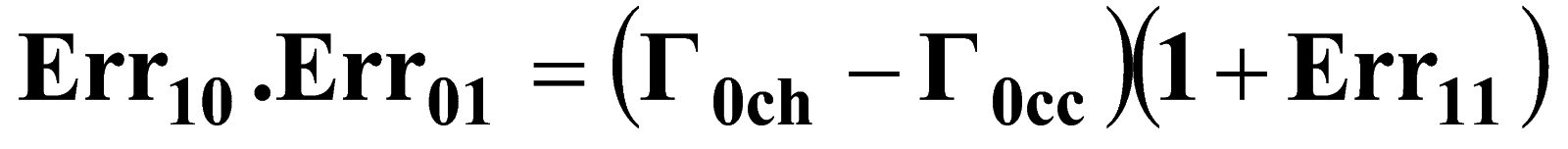
Pour corriger ce défaut Nous avons donc mis en place la correction 3 vecteurs qui au lieu de simplement prendre un seul point en prend 3 et effectue une déformation de l’espace des résultats pour replacer ces points nous utilisons les points 1,0 et -1 pour effectuer cette correction. Pour cella nous utilisons les équations suivantes.



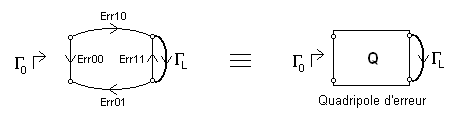
avec :



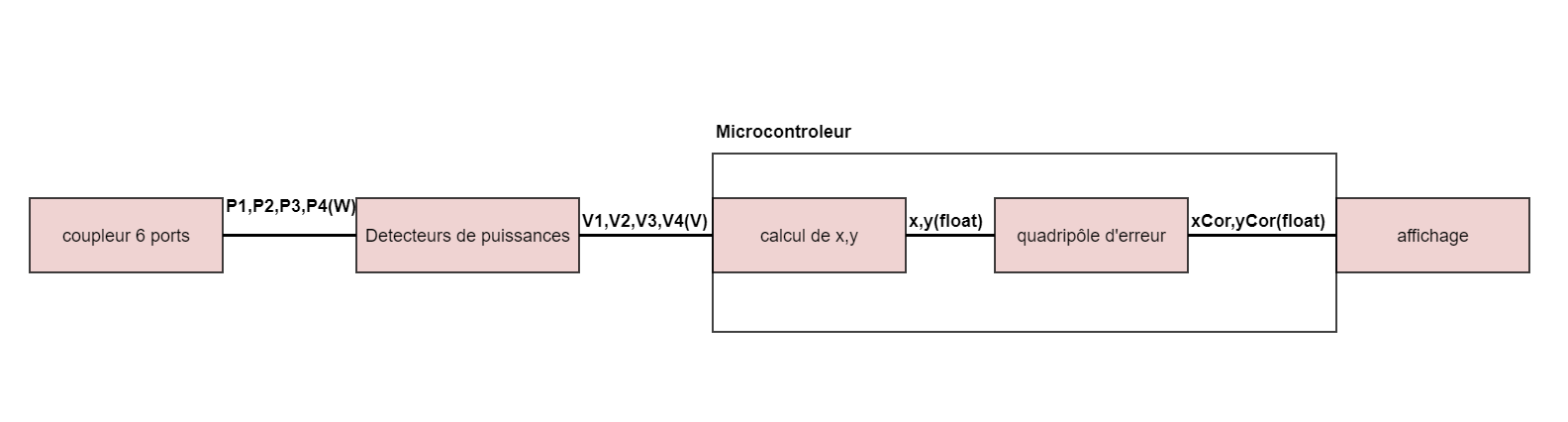


ou :

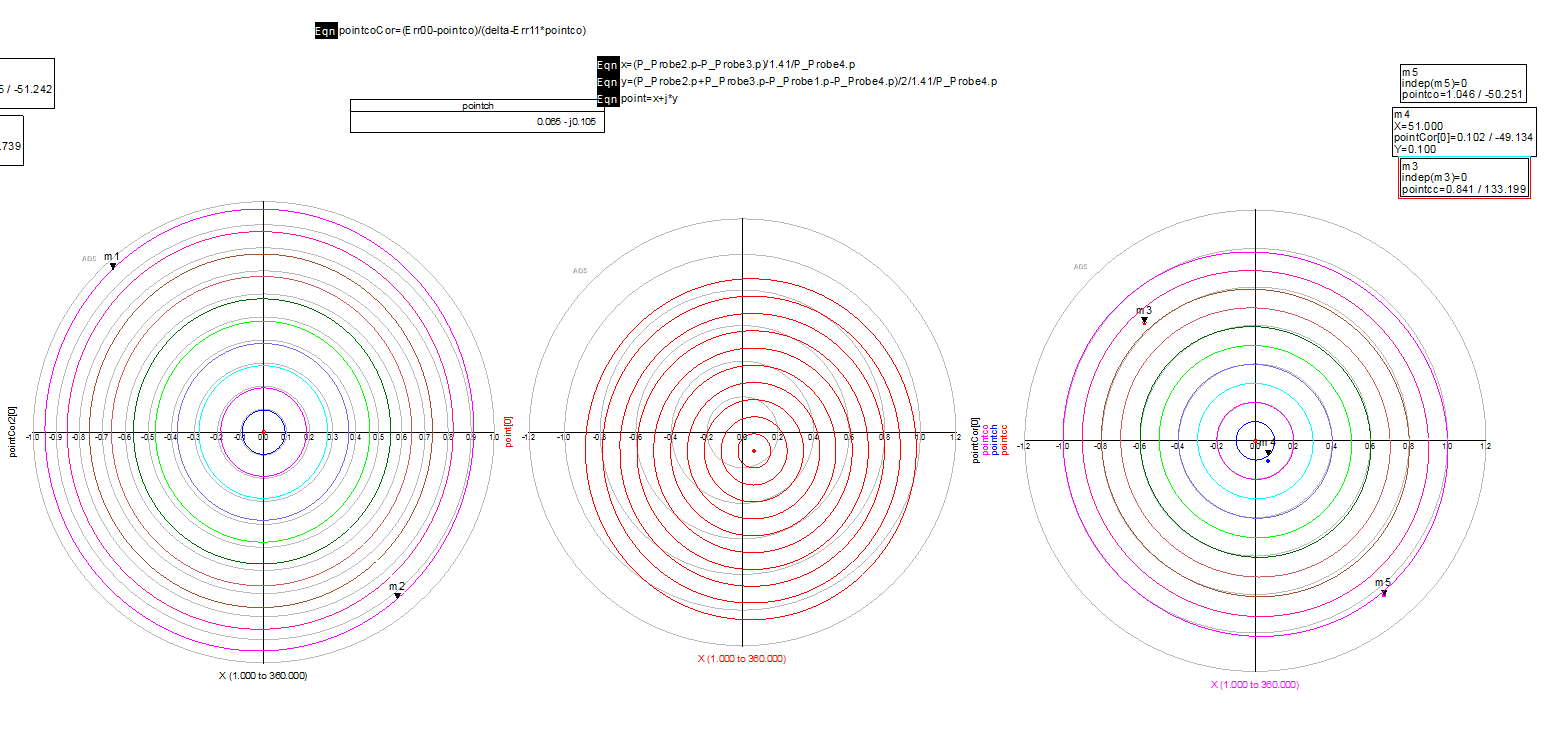
clarifier les notation avec l’outil formule de word.

Cette calibration peut être modélisée par un quadripôle fictif

ajoutés à la fin du système. Mais méthode de calibration nécessite de connaître 3 points afin de déterminer les paramètres S du quadripôle. Ainsi il faudra dans le produit final demander à l’utilisateur de mettre le système dans 3 états connu pour mesurer ces 3 états. Nous pouvons donc modéliser notre système complet comme suit.



Cependant malgré cette contrainte cette méthode est très intéressante car ces résultats sont bien plus précis.



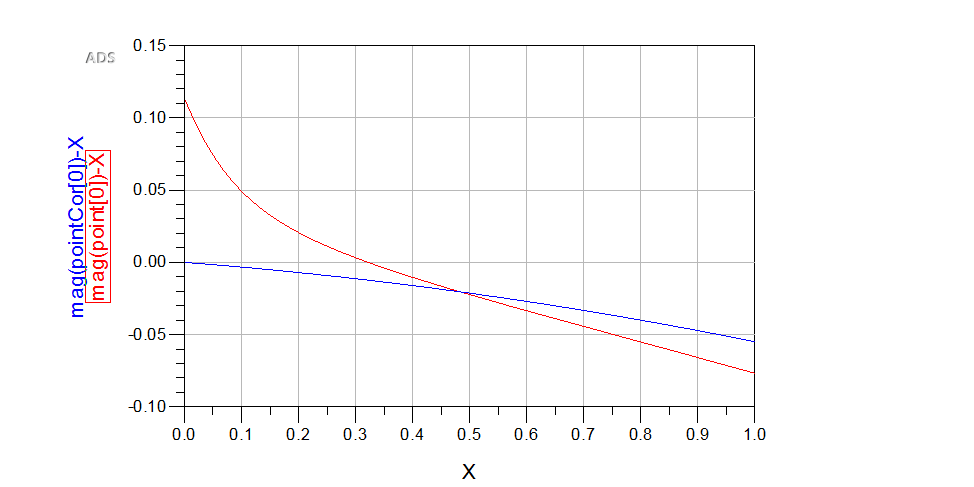
1 vecteur

1 vecteur

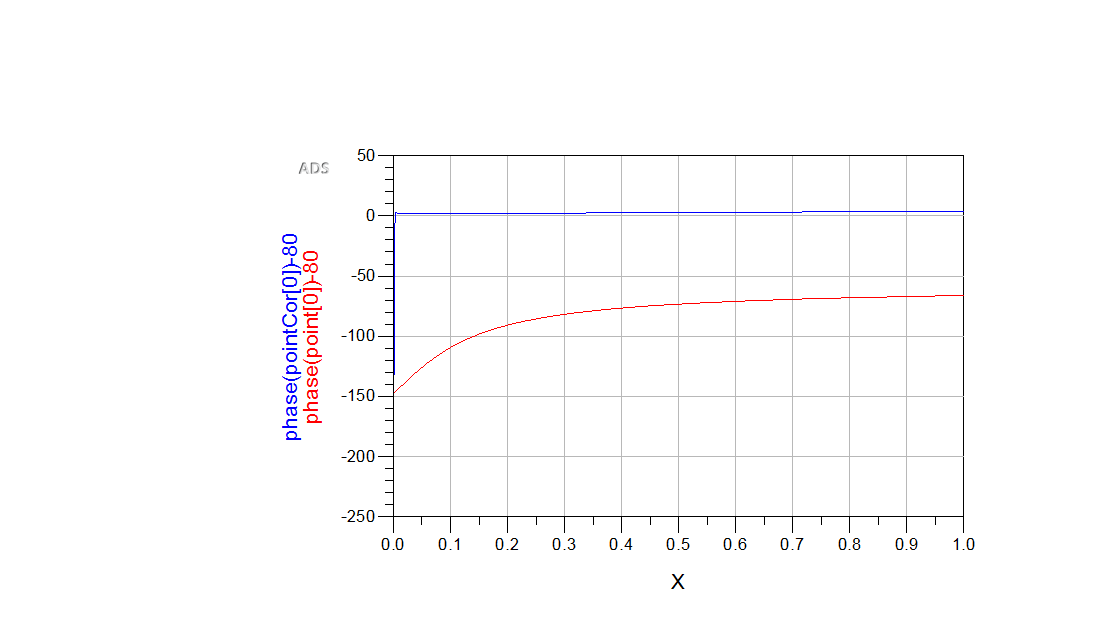
3 vecteur

Aucune correction

Nous allons donc comparer les erreurs entre ces 2 méthodes. Toutes les comparaisons d’erreur ont été faites à 80 degrés car c’est la direction d’erreur maximale et que nous cherchons un majorant d’erreur.

  
Figure 3: erreur en amplitude en fonction de l'amplitude

Ici nous mesurons l’erreur en Amplitude en fonction de l’amplitude pour un déphasage de 80 degré. Nous observons que la correction améliore beaucoup l’erreur dans la plupart des cas et réduit sa croissance. Nous allons maintenant passer a l’erreur de phase. Nous pouvons ainsi voir que le majorant d’erreur en amplitude est de 5 %.

  
Figure 4: erreur en phase en fonction de l'amplitude

Ici nous mesurons l’erreur de phase en fonction de l’amplitude à 80 degré. Nous observons que la correction permet de grandement réduire l‘erreur en amplitude et de la majorer par 1 degré pour les charges passives.

Nous avons donc une calibration qui demande peu de calcul et qui est facile à implémenter elle est donc celle qui sera retenue pour la maquette

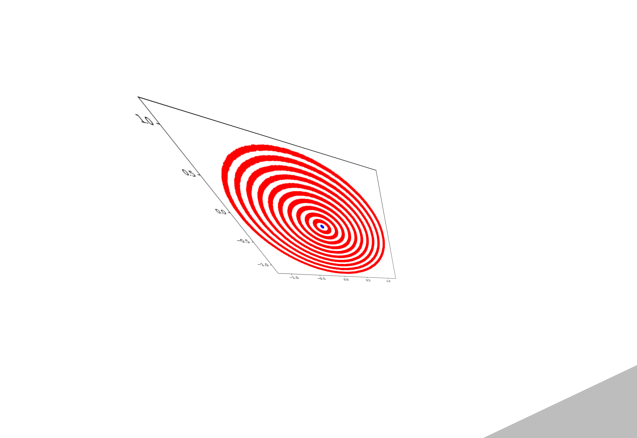
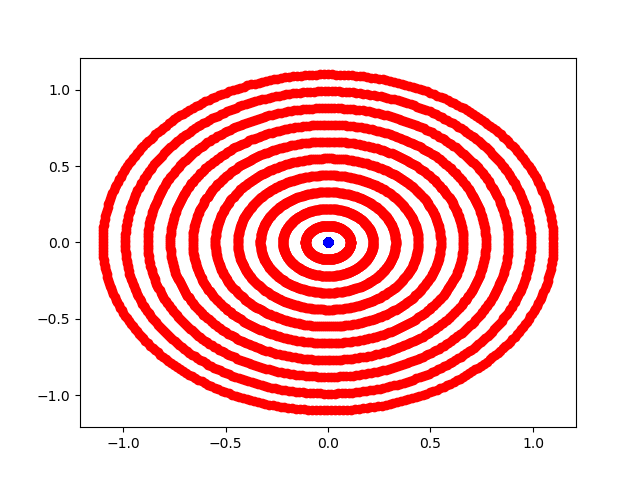
Calibration par changement de perspective :



Une autre calibration que nous avons essayé a été guidé par le constat suivant. La ou nous devrions observer des cercles nous observons des ellipses. Ce phénomène apparaît aussi dans la nature quand nous regardons un cercle mais pas de face. Nous avons donc essayé d’adapter un algorithme de correction de perspective à notre problème. L’idée est de prendre n(>4) point dont ont connais la position réelle et de construire une transformation qui permet de passer d’une image à l’autre.



Application aux images



Apparition d’ellipse dans la nature.

Application à notre problème

Maintenant regardons les résultats.