



Une école de l'IMT

# **Les RPM {Ratio of Polynomial Modelling} et leurs applications aux statistiques de Mellin**

---

Jean-Marie Nicolas

**2019D001**

février 2019

Département Image, Données, Signal  
Groupe IMAGES : *Image, Modélisation,  
Analyse, GEométrie, Synthèse*

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>L'approximation RPM et la recherche numérique des RPC (<i>Rational Polynomial Coefficients</i>)</b>	<b>4</b>
2.1	Les RPM ( <i>Ratio of Polynomial Modelling</i> ) . . . . .	4
2.2	Cas monovariable . . . . .	6
2.2.1	Le cadre formel . . . . .	6
2.2.2	Les solutions classiques : solution directe . . . . .	7
2.2.3	Les solutions classiques : solution pondérée (méthode itérative) . . . . .	8
2.2.4	Les solutions classiques : solution pondérée (méthode itérative) + schéma "à la Tikhonov" . . . . .	9
2.2.5	Un paramètre pour la mise en œuvre : la valeur moyenne . . . . .	9
2.3	Cas à deux variables . . . . .	10
<b>3</b>	<b>Exemple détaillé : l'approximation polynomiale de la fonction Polygamma <math>\Psi(1, x)</math></b>	<b>12</b>
3.1	La fonction Polygamma $\Psi(1, x)$ et son développement en série entière . . . . .	12
3.2	Méthode directe . . . . .	14
3.3	Méthode itérative "à la Tichonov" . . . . .	14
3.4	Comparaison des modèles étudiés . . . . .	16
<b>4</b>	<b>Applications aux statistiques de Mellin : approximation et inversion des fonctions Polygamma</b>	<b>17</b>
4.1	Le rôle des fonction Polygamma en statistiques de Mellin . . . . .	17
4.2	Fonctions Polygamma : modèle RPM . . . . .	17
<b>5</b>	<b>Conclusion</b>	<b>22</b>
<b>A</b>	<b>Quelques relations utiles pour l'imagerie cohérente</b>	<b>22</b>
A.1	Loi Gamma et loi K (lois en intensité) . . . . .	22
A.1.1	Utilitaire de la loi Gamma : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$ . . . . .	22
A.1.2	Utilitaire des caustiques de la loi K : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$ . . . . .	23
A.2	Loi de Rayleigh Nakagami et loi K en amplitude . . . . .	23
A.2.1	Inversion du coefficient de variation de la loi de Rayleigh Nakagami . . . . .	23
A.2.2	Inversion du coefficient de variation de la loi de Rayleigh Nakagami Inverse . . . . .	24
A.2.3	Utilitaire de la loi de Rayleigh-Nakagami : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$ . . . . .	24
A.2.4	Utilitaire de la caustique des lois K en amplitude : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$ . . . . .	25
A.2.5	De la loi de Nakagami à la loi Gamma . . . . .	25
A.2.6	De la loi Gamma à la loi de Nakagami . . . . .	26
A.3	Loi de Rice (type 2) : inversion du coefficient de variation . . . . .	26
A.4	Loi de Weibull : inversion du coefficient de variation . . . . .	26
A.5	Gamma Généralisée : inversion de la fonction $\frac{\Psi(2, L)}{\Psi(1, L)^{1.5}}$ . . . . .	26
A.6	Lois à 3 variables . . . . .	27
A.6.1	Loi K . . . . .	27
A.6.2	Loi de Fisher . . . . .	27

A.6.3	Loi Beta . . . . .	28
<b>B</b>	<b>Les codes de Sarlab dédiés aux RPM : rpclab</b>	<b>28</b>
B.1	Cas 1-D . . . . .	29
B.1.1	Appels simplifiés . . . . .	29
B.1.2	Récupération du RPM . . . . .	30
B.1.3	Appels simplifiés avec paramètres optionnels . . . . .	30
B.1.4	Procédure “interne” générale . . . . .	31
B.2	Cas 2-D . . . . .	32
B.2.1	Appels simplifiés . . . . .	32
B.2.2	Récupération du RPM 2D . . . . .	32
B.2.3	Appels simplifiés avec paramètres optionnels . . . . .	33
B.3	Exemple de l’inversion et de l’approximation d’une fonction . . . . .	33
B.3.1	Approximation d’une fonction . . . . .	33
B.3.2	Inversion d’une fonction . . . . .	36

# 1 Introduction

En imagerie cohérente (comme l'imagerie Radar à Ouverture Synthétique, l'échographie médicale, l'imagerie sonar, ...), les statistiques de Mellin (voir par exemple [1]) ont apporté de nouvelles méthodes d'estimation mieux adaptées à des données définies sur  $\mathbb{R}^+$  (l'amplitude des images) et soumises aux effets du bruit multiplicatif (le chatolement ou *speckle*). Le formalisme ne requiert en particulier qu'une certaine familiarité avec une catégorie de fonctions spéciales : la fonction Gamma  $\Gamma(x)$  et ses dérivées, c'est à dire la fonction Digamma  $\Psi(x)$  (dérivée logarithmique de la fonction Gamma) et les fonctions Polygamma  $\Psi(r, x)$  (dérivées  $r + 1$ -ème de la fonction Digamma). Leurs définitions (données ici pour une variable  $x$  réelle et strictement positive) n'ont rien de vraiment compliqué :

$$\begin{aligned}\Gamma(x) &= \int_0^\infty t^{x-1} e^{-t} dt \\ \Psi(x) &= \frac{d \log \Gamma(x)}{dx} = \frac{\Gamma'(x)}{\Gamma(x)} \\ \Psi(r, x) &= \frac{d^r \Psi(x)}{dx^r}\end{aligned}$$

La fonction Gamma, notée  $\Gamma(x)$ , est bien connue de par sa propriété fondamentale :

$$\Gamma(x + 1) = x \Gamma(x)$$

et si on se restreint aux valeurs entières, on a une relation impliquant la fonction factorielle universellement connue :

$$n! = \Gamma(n + 1)$$

On peut aussi noter une caractéristique des fonctions Polygamma qui sont strictement monotones (positives pour  $r$  impair et négatives pour  $r$  pair), et qui convergent vers 0 pour  $x \rightarrow \infty$  et vers l'infini pour  $x \rightarrow 0^+$  (le signe dépend de  $r$ ).

Si ce formalisme est très satisfaisant sur le plan analytique, il peut poser problème dès lors que l'on souhaite manipuler les expressions des statistiques de Mellin, et, en particulier, dans le cas où on souhaite, dans un code informatique, inverser une relation. En effet, si fonctions Gamma, Digamma et Polygamma existent dans les logiciels actuels (Matlab, Python) ou sont faciles à implémenter dans d'autres<sup>1</sup>, la forme inverse de ces fonctions est encore à inventer. Même sur le plan analytique, s'il existe bien une fonction inverse aux fonctions circulaires (les arcsinus par exemple), aux fonctions de type exponentiel (les arcsinus hyperbolique par exemple), la famille des fonctions Gamma en est totalement dépourvu. Tout au plus on peut manipuler formellement les fonctions  $\Gamma^{-1}(y)$ ,  $\Psi^{-1}(y)$  et  $\Psi^{-1}(r, y)$  définies par :

$$\begin{aligned}x &= \Gamma^{-1}(y) \Leftrightarrow y = \Gamma(x) \\ x &= \Psi^{-1}(y) \Leftrightarrow y = \Psi(x) \\ x &= \Psi^{-1}(r, y) \Leftrightarrow y = \Psi(r, x)\end{aligned}$$

mais, en pratique, on ne peut que déplorer qu'il n'existe aucune implémentation logicielle de ces fonctions inverses dans aucun langage.

Il est alors raisonnable de se pencher sur les diverses approximations possibles pour ces fonctions, tant dans leurs expressions directes qu'inverses. Par exemple, ces fonctions étant

---

1. en C, les *Numerical Recipes* s'appuient sur l'approximation de Lanczos et proposent une expression pour la fonction Gamma, que l'on peut dériver pour obtenir un code pour les fonctions Digamma et Polygamma.

infiniment dérivables sur  $\mathbb{R}^+$ , on peut envisager des développements de type Taylor : c'est cette approche qui avait été utilisée dans le rapport [1], mais, à l'usage, les relations obtenues étaient parfois peu pratiques à programmer. De plus, la relation inverse était difficile à établir.

L'objectif de ce rapport est donc de proposer une nouvelle approche, fortement inspirée des développements polynomiaux, mais plus généraux : le RPM (*Ratio of Polynomial Modelling*), bien connu en imagerie satellitaire puisque ce type de modèle est essentiel dans les techniques de recalage d'images optiques sur les grilles de géoréférencement. Après une présentation du RPM, nous appliquerons cette approche à la famille des fonctions Gamma et Polygamma, tant pour leur approximation que pour leur inversion. D'autres cas de figures propres à l'imagerie cohérente seront étudiés.

Notons que si l'essentiel des applications proposées dans ce rapport sont relatifs aux lois statistiques sur  $\mathbb{R}^+$ , il est bien évidemment possible d'utiliser ce formalisme pour d'autres cas rencontrés dans d'autres disciplines scientifiques (modélisation de la trajectoire d'un satellite par exemple).

## 2 L'approximation RPM et la recherche numérique des RPC (*Rational Polynomial Coefficients*)

### 2.1 Les RPM (*Ratio of Polynomial Modelling*)

Il existe de nombreux cas où l'on souhaite disposer d'une expression analytique approximant au mieux une fonction sur un certain nombre de données pour lesquelles on connaît la valeur de cette fonction. Cette fonction pouvant être connue sous forme explicite ou implicite. De manière plus formelle :

- on connaît un certain nombre de valeurs d'entrées  $x_n, n \in [1, N]$
- pour chaque  $x_n$ , on connaît une valeur  $y_n$

On a ainsi un ensemble de valeurs  $\{(x_n, y_n), n \in [1, N]\}$  que l'on peut voir comme une base d'apprentissage aussi bien dans le cas de la recherche d'une fonction directe permettant de passer d'un  $x_n$  à un  $y_n$  que dans le cas de l'inversion, c'est à dire la recherche d'une fonction permettant de passer d'un  $y_n$  à un  $x_n$ . Dans le cadre de la télédétection, on parle souvent de cet ensemble de données comme d'une *boîte à chaussure*.

Il existe bien entendu de nombreuses expressions analytiques  $g$  telle que  $g(x_n) \simeq y_n \forall n \in [1, N]$ . La plus connue est la forme polynomiale pour laquelle on dispose d'un polynôme de degré  $L$  défini par ses coefficients  $a_i, i \in [1, L]$  : pour tout  $x_n, n \in [1, N]$ , on a

$$y_n \simeq \sum_{i=0}^L a_i x_n^i \quad (1)$$

La détermination des coefficients  $a_i$  du polynôme reposant principalement dans le sens que l'on donne à la relation " $\simeq$ ".

En particulier, si l'on recherche à minimiser l'erreur quadratique moyenne calculée sur les échantillons, la détermination du polynôme est alors un système linéaire, ce qui a toujours fait le succès de cette méthode.

Cependant, la méthode polynomiale n'a pas que des adeptes car son comportement asymptotique est dictée par le coefficient le plus élevé, ce qui conduit à des résultats parfois trop éloignés de la réalité sous jacente. De plus un certain doigté est de mise pour choisir le degré du polynôme : en particulier, si la valeur extrême de  $L$  est égale au nombre des entrées  $N$ , il est

bien connu que dans ce cas l'erreur quadratique moyenne est nulle, mais le polynôme n'a aucun pouvoir de généralisation pour des valeurs d'entrée autres que le jeu initial des  $x_n, n \in [1, N]$ .

Si l'on ose des rapprochements provocateurs, on peut noter qu'en filtrage de signaux 1-D l'opérateur de convolution (qui correspond au plus élémentaire des méthodes de filtrage linéaire) a une transformée en Z polynomiale. Les limitations de ce type de filtrage ont conduit naturellement à la conception des filtres récursifs qui ont des transformées en Z s'écrivant comme des rapports de transformées en Z d'opérateurs de convolution : ajouter ainsi un dénominateur à la transformée en Z d'un filtre initial a ouvert de nouveaux horizons dans le design des filtres linéaires.

La genèse des RPM (Rational Polynomial Model) a pour point de départ l'approximation de fonctions grâce à l'ajout d'un dénominateur polynomial à la relation 1 (polynôme de degré  $N$ ). On a au final un numérateur de degré  $L$  et un dénominateur de degré  $M$ .

$$y_n \simeq \sum_{i=0}^L a_i x_n^i \rightarrow y_n \simeq \frac{\sum_{i=0}^L a_i x_n^i}{1. + \sum_{i=1}^M b_i x_n^i} \quad (2)$$

Les coefficients  $a_i$  et  $b_i$  sont les RPC (*Rational Polynomial Coefficients*)

Puisque le RPM a été défini pour avoir un dénominateur égal à l'unité à l'origine, le comportement asymptotique du RPM ainsi construit sera dicté par le polynôme du numérateur à l'origine et par le polynôme du dénominateur à l'infini, ce qui ouvre bien évidemment des perspectives intéressantes en terme d'interprétation de modèle.

Il semblerait que l'usage des RPM soit principalement apparu dans le domaine de la télédétection pour associer des pixels acquis par des systèmes imageurs à des positions spécifiques sur la Terre. Comme le modèle physique de l'acquisition est d'une complexité redoutable (la Terre n'ayant plus rien de sphérique pour les très hautes résolutions actuelles), mais qu'il est quasiment toujours possible d'avoir un jeu de données associant à un point de la Terre la position du pixel sur l'image, on peut alors définir un RPM donnant en sortie le pixel (en valeur décimale) correspondant à un triplet (longitude, latitude, altitude) donné. On a ainsi la position d'un pixel  $(ix, iy)$  sous la forme de RPM de degré 3 prenant en compte longitude ( $x_{ref}$ ), latitude ( $y_{ref}$ ) et altitude ( $z_{ref}$ ) :

$$ix = \frac{\sum_i \sum_j \sum_k a_{ijk} x_{ref}^i y_{ref}^j z_{ref}^k}{\sum_i \sum_j \sum_k b_{ijk} x_{ref}^i y_{ref}^j z_{ref}^k} \quad iy = \frac{\sum_i \sum_j \sum_k c_{ijk} x_{ref}^i y_{ref}^j z_{ref}^k}{\sum_i \sum_j \sum_k d_{ijk} x_{ref}^i y_{ref}^j z_{ref}^k}$$

avec  $0 \leq i + j + k \leq 3$ , ce qui donne dans chaque cas 20 coefficients au numérateur et 20 coefficients au dénominateur. Ce type de modèle de recalage semble avoir été généralisé en imagerie satellitaire dès l'apparition, en 1999, des premiers satellites THR (très haute résolution, c'est à dire métrique et submétrique). A l'heure actuelle, toutes les agences spatiales fournissent leurs données associées à un RPM permettant ce passage en coordonnées image<sup>2</sup>.

Un certain nombre de publications scientifiques sont dédiées à cette modélisation spécifique à la télédétection<sup>3</sup>, en particulier l'article de Tao [3]. Il est alors aisé de généraliser ce modèle

---

2. Cela peut dépendre du type de produit demandé, la tendance actuelle étant de fournir les images géoréférencées.

3. Il faut cependant noter qu'elles sont parsemées de petites erreurs que le lecteur est censé corriger...

à des cas moins spécifiques : c'est ce qui fera l'objet de la section 2.2 qui reprend et détaille dans un cadre plus général les bases du modèle RPM décrites dans [3]. S'il faut retenir un point important de ces travaux menés dans un cadre restreint (la télédétection), c'est la difficulté à trouver les "bons" RPC (Rational polynomial Coefficients) : en effet, la méthode la plus usitée est fondée sur un schéma "à la Tichonov" qui nécessite l'introduction d'un paramètre a priori petit :  $\varepsilon$ . Ce paramètre joue un grand rôle dans la qualité de la convergence du résultat puisqu'il permet de s'affranchir dans certains cas d'un conditionnement de matrice médiocre. Les bons auteurs s'accordent à dire que la valeur à choisir nécessite un certain doigté, mais il n'existe pas à l'heure actuelle de méthode analytique permettant d'en optimiser la valeur.

## 2.2 Cas monovariabale

### 2.2.1 Le cadre formel

La mise en œuvre d'un modèle RPM pour une fonction  $F(x)$  nécessite la construction d'un jeu de données  $(x_n, U_n), n \in [1, N]$  (la "boîte à chaussure") tel que la fonction  $F(x)$  que l'on recherche vérifie le mieux possible :

$$U_n = F(x_n) \quad \forall n \in [1, N]$$

A noter que cette fonction peut ne pas être explicite : seule la donnée élémentaire  $x_n \rightarrow U_n$  est requise, ce qui peut être aussi obtenu par expérience.

On fait alors l'hypothèse que la fonction recherchée soit un rapport de deux polynômes, celui du numérateur étant de degré  $L$  et celui du dénominateur étant de degré  $M$ . On obtient alors pour toute valeur  $x_n$  une valeur approchée  $\tilde{U}_n$  :

$$\tilde{U}_n = \frac{a_0 + \sum_{i=1}^L a_i x_n^i}{1. + \sum_{i=1}^M b_i x_n^i} \quad (3)$$

que l'on trouve chez les bons auteurs sous la forme vectorielle suivante ( $\odot$  représente le produit scalaire) :

$$\tilde{U}_n = \frac{(1 \ x_n \ x_n^2 \dots x_n^L) \odot (a_0 \ a_1 \ a_2 \dots a_L)^t}{(1 \ x_n \ x_n^2 \dots x_n^M) \odot (1 \ b_1 \ b_2 \dots b_M)^t} \quad \forall n \in [1, N]$$

En notant  $\mathbf{x}$  le vecteur  $(x_1, x_2, \dots, x_N)$ ,  $\mathbf{x}^k$  le vecteur  $(x_1^k, x_2^k, \dots, x_N^k)$ ,  $\mathbf{1}$  le vecteur unité de dimension  $N$ , et en utilisant la notation de la division "point à point" (comme le permet Matlab avec l'opération `./` et Python avec l'opération `numpy.divide`), on obtient formellement le vecteur  $\tilde{\mathbf{U}}$  :

$$\tilde{\mathbf{U}} = \left( (\mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^L) \odot (a_0 \ a_1 \ a_2 \dots a_L)^t \right) ./ \left( (\mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^M) \odot (1 \ b_1 \ b_2 \dots b_M)^t \right) \quad (4)$$

L'objectif est donc de trouver le jeu des  $L + M + 1$  coefficients  $(a_i, i \in [0, L])$  et  $(b_i, i \in [1, M])$  qui donne, pour le vecteur  $\mathbf{x}$  du jeu de données initial, un vecteur  $\tilde{\mathbf{U}}$  aussi proche que possible du vecteur initial  $\mathbf{U}$  du jeu de données initial.

Pour cela, à partir de la relation 4, on peut écrire :

$$\tilde{\mathbf{U}} * \left( (\mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^M) \odot (1 \ b_1 \ b_2 \dots b_M)^t \right) = \left( (\mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^L) \odot (a_0 \ a_1 \ a_2 \dots a_L)^t \right)$$

l'opération “.” correspondant à la multiplication “point à point” (comme le permet Matlab avec l'opération “.” et Python avec l'opération `numpy.multiply`), et on en déduit :

$$\tilde{\mathbf{U}} = -\tilde{\mathbf{U}} * \left( (\mathbf{x} \mathbf{x}^2 \dots \mathbf{x}^M) \odot (b_1 \ b_2 \dots b_M)^t \right) + \left( (\mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^L) \odot (a_0 \ a_1 \ a_2 \dots a_L)^t \right)$$

ce qui donne :

$$\begin{aligned} \tilde{\mathbf{U}} &= - \left( (\tilde{\mathbf{U}} * \mathbf{x}, \tilde{\mathbf{U}} * \mathbf{x}^2, \dots, \tilde{\mathbf{U}} * \mathbf{x}^M) \odot (b_1 \ b_2 \dots b_M)^t \right) \\ &\quad + \left( (\mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^L) \odot (a_0 \ a_1 \ a_2 \dots a_L)^t \right) \\ &= \left( \mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^L - \tilde{\mathbf{U}} * \mathbf{x}, \tilde{\mathbf{U}} * \mathbf{x}^2, \dots, -\tilde{\mathbf{U}} * \mathbf{x}^M \right) \odot (a_0 \ a_1 \ a_2 \dots a_L \ b_1 \ b_2 \dots b_M)^t \end{aligned}$$

En regroupant les coefficients  $a_i$  et  $b_i$  dans un unique vecteur  $\mathbf{J}$  de dimension  $L + M + 1$  tel que :

$$\mathbf{J} = (a_0, a_1, a_2, \dots, a_L, b_1, b_2, \dots, b_M)^t \quad (5)$$

on obtient pour finir l'expression suivante :

$$\tilde{\mathbf{U}} = \left( \mathbf{1} \ \mathbf{x} \ \mathbf{x}^2 \dots \mathbf{x}^L - \tilde{\mathbf{U}} * \mathbf{x}, -\tilde{\mathbf{U}} * \mathbf{x}^2, \dots, -\tilde{\mathbf{U}} * \mathbf{x}^M \right) \odot \mathbf{J} \quad (6)$$

C'est ce vecteur  $\mathbf{J}$  qu'il faut chercher et qui contient les coefficients du RPM.

Introduisons maintenant la matrice  $\mathbf{M}$ , composée de  $L + M + 1$  colonnes et  $N$  lignes telle que la ligne  $n$  de cette matrice soit définie par :

$$\mathbf{M}[n, :] = \left( 1, x_n^1, x_n^2, \dots, x_n^L, -U_n x_n^1, -U_n x_n^2, \dots, -U_n x_n^M \right)$$

En l'introduisant dans l'équation 6, on peut réécrire notre problème en disant que l'on recherche le vecteur  $\mathbf{J}$  (composé du jeu de coefficients  $a_i, b_j$ ) tel que  $\tilde{\mathbf{U}}$  défini par :

$$\tilde{\mathbf{U}} = \mathbf{M} \odot \mathbf{J} \quad (7)$$

soit, le plus possible, proche de  $\mathbf{U}$ . Ceci signifie que le vecteur d'erreur  $\mathbf{V}$  :

$$\mathbf{V} = \mathbf{U} - \mathbf{M} \odot \mathbf{J} \quad (8)$$

doit être le plus proche possible du vecteur nul.

### 2.2.2 Les solutions classiques : solution directe

Considérons la relation 7 :

$$\tilde{\mathbf{U}} = \mathbf{M} \odot \mathbf{J}$$

Comme dans notre problème la matrice  $\mathbf{M}$  n'est pas une matrice carrée, il faut très classiquement réécrire la relation 7 en la multipliant par  $\mathbf{M}^t$  :

$$\mathbf{M}^t \tilde{\mathbf{U}} = \mathbf{M}^t \mathbf{M} \odot \mathbf{J} \quad (9)$$

La matrice  $\mathbf{M}^t \mathbf{M}$  étant carrée, elle est potentiellement inversible.

Si elle est effectivement inversible, on obtient alors très classiquement la solution dite directe pour  $\mathbf{J}$  par la relation :

$$\mathbf{J} = \left( \mathbf{M}^t \mathbf{M} \right)^{-1} \mathbf{M}^t \mathbf{U} \quad (10)$$

On obtient ainsi une solution à notre problème, le seul risque résidant dans un mauvais conditionnement de la matrice  $\mathbf{M}^t \mathbf{M}$ .



### 2.2.3 Les solutions classiques : solution pondérée (méthode itérative)

En s'inspirant des méthodes linéaires où l'on peut utiliser la matrice de covariance comme matrice des poids, on va définir une matrice des poids  $\mathbf{W}$  uniquement à partir des coefficients  $b_i$  du dénominateur de notre solution RPM, ces vecteurs définissant le vecteur  $\mathbf{B}$ . On définit alors un nouveau vecteur  $\mathbf{WB}$  par ses composantes  $wb_n$  :

$$wb_n = (1 \ x_n \ x_n^2 \dots x_n^M) \odot (1 \ b_1 \ b_2 \dots b_M)^t \quad (11)$$

$\mathbf{WB}$  est un vecteur de dimension  $N$ . Formellement, on peut définir son inverse :

$$\left( \frac{1}{\mathbf{WB}} \right)_n = \frac{1}{wb_n} = \frac{1}{(1 \ x_n \ x_n^2 \dots x_n^M) \odot (1 \ b_1 \ b_2 \dots b_M)^t}$$

On définit ainsi la matrice des poids  $\mathbf{W}$ , de dimension  $N \times N$  sous la forme :

$$\mathbf{W} = \begin{pmatrix} \left( \frac{1}{\mathbf{WB}} \right)_1 & 0 & \dots & 0 \\ 0 & \left( \frac{1}{\mathbf{WB}} \right)_2 & \dots & 0 \\ 0 & . & .. & 0 \\ 0 & 0 & \dots & \left( \frac{1}{\mathbf{WB}} \right)_n \end{pmatrix} \quad (12)$$

Cette matrice dépend du jeu de coefficients solution, ce qui aura des conséquences importantes car on ne pourra plus utiliser le modèle direct. Elle va être appliquée en guise de matrice de pondération, ce qui conduit à une nouvelle expression de l'équation 9 :

$$\mathbf{M}^t \tilde{\mathbf{U}} = \mathbf{M}^t \mathbf{M} \odot \mathbf{J} \quad \rightarrow \quad \mathbf{M}^t \mathbf{W}^2 \tilde{\mathbf{U}} = \mathbf{M}^t \mathbf{W}^2 \mathbf{M} \odot \mathbf{J} \quad (13)$$

On remarque que l'on ne peut lui associer un modèle direct pour sa résolution puisque, ne connaissant pas la matrice  $\mathbf{W}$ , on ne peut envisager de l'inverser.

Aussi, pour résoudre la relation 13, il est d'usage d'appliquer la méthode itérative suivante :

— On initialise le système (étape [0]) en prenant pour matrice  $\mathbf{W}_{[0]}$  la matrice identité :

$$\mathbf{W}_{[0]} = \mathbf{E}$$

— ensuite on effectue les itérations selon le schéma suivant :

— connaissant à l'issue de l'étape  $[s-1]$  on va écrire l'expression vérifiée à l'étape  $[s]$  par la solution  $\mathbf{J}_{[s]}$  :

$$\mathbf{M}^t \mathbf{W}_{[s-1]}^2 \tilde{\mathbf{U}} = \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} \odot \mathbf{J}_{[s]} \quad (14)$$

que l'on peut alors résoudre par un schéma direct identique à celui du paragraphe 2.2.2, ce qui donne  $\mathbf{J}_{[s]}$  :

$$\mathbf{J}_{[s]} = \left( \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} \right)^{-1} \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{U} \quad (15)$$

— A partir de  $\mathbf{J}_{[s]}$ , on extrait le vecteur  $\mathbf{B}_{[s]}$ , ce qui, grâce à la relation 12, donne la matrice  $\mathbf{W}_{[s]}$ .

— Un test d'arrêt s'insère dans ce schéma.

Les expérimentations montrent que ce modèle pondéré n'apporte guère d'améliorations par rapport au modèle direct.

### 2.2.4 Les solutions classiques : solution pondérée (méthode itérative) + schéma “à la Tikhonov”

La régularisation “à la Tikhonov” s’exprime par la réécriture de l’équation 13 en y insérant un terme de régularisation (avec  $h$  petit), ce qui donne l’expression suivante :

$$\mathbf{M}^t \mathbf{W}^2 \tilde{\mathbf{U}} = \left( \mathbf{M}^t \mathbf{W}^2 \mathbf{M} + h^2 \mathbf{E} \right) \odot \mathbf{J} \quad (16)$$

Il suffit ensuite d’appliquer le même schéma itératif que dans le cas précédent :

- on initialise le système (étape [0]), c’est à dire la matrice  $\mathbf{W}_{[0]}$  et le vecteur solution  $\mathbf{J}_{[0]}$ , en prenant
- pour matrice  $\mathbf{W}_{[0]}$  la matrice identité :

$$\mathbf{W}_{[0]} = \mathbf{E}$$

- pour vecteur solution  $\mathbf{J}_{[0]}$ , soit le vecteur solution du problème direct (équation 10), soit le vecteur nul (cas le plus usuel).
- ensuite, on effectue les itérations selon le schéma suivant :
- connaissant à l’issue de l’étape  $[s - 1]$  la solution  $\mathbf{J}_{[s-1]}$  et la matrice  $\mathbf{W}_{[s-1]}$ , on va écrire l’expression vérifiée à l’étape  $[s]$  par la solution  $\mathbf{J}_{[s]}$  (relation 16) :

$$\mathbf{M}^t \mathbf{W}_{[s-1]}^2 \tilde{\mathbf{U}} = \left( \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} + h^2 \mathbf{E} \right) \odot \mathbf{J}_{[s]} \quad (17)$$

que l’on pourrait certes envisager de résoudre par un schéma direct identique à celui du paragraphe 2.2.2, ce qui donnerait  $\mathbf{J}_{[s]}$  :

$$\mathbf{J}_{[s]} = \left( \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} + h^2 \mathbf{E} \right)^{-1} \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{U} \quad (18)$$

mais qui, au vu des résultats que l’on obtiendrait, doit être avantageusement remplacé par le schéma itératif suivant (méthode de Tichonov) :

$$\mathbf{J}_{[s]} = \mathbf{J}_{[s-1]} + \left( \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} + h^2 \mathbf{E} \right)^{-1} \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{V}_{[s-1]} \quad (19)$$

expression dans laquelle on fait intervenir le terme d’erreur  $\mathbf{V}$  (relation 8) :

$$\mathbf{V}_{[s-1]} = \mathbf{U} - \mathbf{M} \mathbf{J}_{[s-1]} \quad (20)$$

- A partir de  $\mathbf{J}_{[s]}$ , on extrait le vecteur  $\mathbf{B}_{[s]}$ , ce qui, grâce à la relation 12, donne la matrice  $\mathbf{W}_{[s]}$ .
- Un test d’arrêt s’insère dans ce schéma.

Notons que le choix du paramètre  $h$  n’est pas trivial : selon les bons auteurs, il semble qu’il faille tester pour différentes valeurs de  $h$  et conserver celle qui donne le “meilleur” résultat [3].

### 2.2.5 Un paramètre pour la mise en œuvre : la valeur moyenne

Si on considère une approximation polynomiale quelconque selon une variable  $x$ , elle n’est en pratique valide que dans le domaine des valeurs de  $x$  couvert par les données en entrée, c’est à dire le jeu de données  $(x_n, U_n), n \in [1, N]$  utilisé pour définir cette approximation. On a donc un intervalle  $[x_{\min}, x_{\max}]$  dans lequel l’expression trouvée est utilisable, et, dans des cas de figure

“sympathiques”, cette expression pourra être étendue au delà de cet intervalle. On a donc pour les données tests dans le cas d’une modélisation RPM :

$$U = F(x) = \frac{a_0 + \sum_{i=1}^L a_i x^i}{1. + \sum_{i=1}^M b_i x^i}$$

En privilégiant une “valeur moyenne”  $\hat{x}$  dans l’intervalle  $[x_{\min}, x_{\max}]$ , on définit une valeur moyenne  $\hat{U}$  :

$$\hat{U} = \frac{a_0 + \sum_{i=1}^L a_i \hat{x}^i}{1. + \sum_{i=1}^M b_i \hat{x}^i}$$

Dans ce cas là, la modélisation RPM peut se formuler en privilégiant cette valeur moyenne et on peut la réécrire sous la forme :

$$U = \frac{\hat{U} + \sum_{i=1}^L c_i (x - \hat{x})^i}{1. + \sum_{i=1}^M d_i (x - \hat{x})^i} \quad (21)$$

Cette expression permet parfois une meilleure appréhension de la modélisation choisie, et, surtout, peut être privilégiée lors de la recherche numérique du modèle car il évite des erreurs numériques liées à des valeurs de  $x$  non nulles et des puissances de la variable  $x$  élevées.

Enfin notons que contrairement à la modélisation présentée dans laquelle l’expression du dénominateur commençait par la valeur 1, on peut tout à fait écrire la modélisation RPM sous la forme équivalente :

$$U = \frac{1 + \sum_{i=1}^L c'_i (x - \hat{x})^i}{\frac{1}{\hat{U}} + \sum_{i=1}^M d'_i (x - \hat{x})^i} \quad (22)$$

avec :

$$c'_i = \frac{c_i}{\hat{U}} \quad d'_i = \frac{d_i}{\hat{U}}$$

### 2.3 Cas à deux variables

Ce paragraphe traite le cas où l’on dispose de deux mesures,  $(U_1, U_2)$ , en chaque point, ce point de mesure étant défini par deux variables  $(x, y)$ . Le RPM le plus classique consiste alors à écrire deux polynômes (numérateur et dénominateur) prenant en compte les puissances des variables  $x$  et  $y$  ainsi que des termes croisés. Une forme générale est donnée dans l’expression

suivante sous sa forme la plus générale :

$$U_1 = \frac{\sum_{i=0}^N \sum_{j=0}^M a_{ij} x^i y^j}{\sum_{i=0}^N \sum_{j=0}^M b_{ij} x^i y^j}$$

$$U_2 = \frac{\sum_{i=0}^N \sum_{j=0}^M c_{ij} x^i y^j}{\sum_{i=0}^N \sum_{j=0}^M d_{ij} x^i y^j}$$

en prenant le plus souvent  $b_{00} = 1$  et  $d_{00} = 1$ .

Le cas opérationnel qui nous intéresse est celui où  $N = M = 3$  (comme pour le recalage d'image satellitaire<sup>4</sup>. Il y a alors 20 coefficients au numérateur et 19 coefficients au dénominateur.). avec la formulation suivante :

$$\tilde{U}_1 = \frac{a_0 + a_1x + a_2y + a_3xy + a_4x^2 + a_5y^2 + a_6x^3 + a_7x^2y + a_8xy^2 + a_9y^3}{1 + b_1x + b_2y + b_3xy + b_4x^2 + b_5y^2 + b_6x^3 + b_7x^2y + b_8xy^2 + b_9y^3}$$

$$\tilde{U}_2 = \frac{c_0 + c_1x + c_2y + c_3xy + c_4x^2 + c_5y^2 + c_6x^3 + c_7x^2y + c_8xy^2 + c_9y^3}{1 + d_1x + d_2y + d_3xy + d_4x^2 + d_5y^2 + d_6x^3 + d_7x^2y + d_8xy^2 + d_9y^3}$$

Le cas monodimensionnel s'applique aux deux dimensions moyennant un conditionnement spécifique des matrices. On a ainsi :

— pour le vecteur cible :

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_1 \\ \hline \mathbf{U}_2 \end{bmatrix} \quad (23)$$

— pour le vecteur erreur :

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_1 \\ \hline \mathbf{V}_2 \end{bmatrix} \quad (24)$$

— pour la matrice  $\mathbf{W}$

$$\mathbf{W} = \left[ \begin{array}{c|c} \mathbf{W}_1 & 0 \\ \hline 0 & \mathbf{W}_2 \end{array} \right] \quad (25)$$

— pour la matrice  $\mathbf{M}$

$$\mathbf{M} = \left[ \begin{array}{c|c} \mathbf{M}_1 & 0 \\ \hline 0 & \mathbf{M}_2 \end{array} \right] \quad (26)$$

---

4. à ceci près qu'en imagerie on utilise trois variables :  $x$ ,  $y$  et  $z$ , par exemple, longitude, latitude et altitude

— enfin pour le vecteur  $\mathbf{J}$

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \end{bmatrix} \quad (27)$$

c'est à dire, dans notre cas :

$$\mathbf{J} = (a_0, a_1, \dots, a_9, b_1, b_2, \dots, b_9, c_0, c_1, \dots, c_9, d_1, d_2, \dots, d_9)^t \quad (28)$$

Le schéma “à la Tichonov”, toujours fondé sur la relation 16 :

$$\mathbf{M}^t \mathbf{W}^2 \tilde{\mathbf{U}} = (\mathbf{M}^t \mathbf{W}^2 \mathbf{M} + h^2 \mathbf{E}) \odot \mathbf{J}$$

se décompose comme suit :

- Initialisation du système (étape [0]) en prenant
- pour matrice  $\mathbf{W}$  la matrice identité :

$$\mathbf{W}_{[0]} = \mathbf{E}$$

- pour le vecteur solution  $\mathbf{J}$ , le vecteur nul (cas le plus courante).
- connaissant, à l'étape  $[s-1]$ , la solution  $\mathbf{J}_{[s-1]}$  ainsi que la matrice  $\mathbf{W}_{[s-1]}$ , on écrit l'expression vérifiée à l'étape  $[s]$  par la solution  $\mathbf{J}_{[s]}$  (16) :

$$\mathbf{M}^t \mathbf{W}_{[s-1]}^2 \tilde{\mathbf{U}} = (\mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} + h^2 \mathbf{E}) \odot \mathbf{J}_{[s]}$$

que l'on résout par le schéma itératif suivant (méthode de Tichonov, relation 19) :

$$\mathbf{J}_{[s]} = \mathbf{J}_{[s-1]} + (\mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{M} + h^2 \mathbf{E})^{-1} \mathbf{M}^t \mathbf{W}_{[s-1]}^2 \mathbf{V}_{[s-1]}$$

expression dans laquelle on fait intervenir le terme d'erreur  $\mathbf{V}$  (relation 20) :

$$\mathbf{V}_{[s-1]} = \mathbf{U} - \mathbf{M} \mathbf{J}_{[s-1]}$$

- connaissant  $\mathbf{J}_{[s]}$ , on en déduit  $\mathbf{W}_{[s]}$ .
- on applique un test d'arrêt.

Comme dans le cas 1-D, il peut être intéressant d'opérer autour d'une “valeur moyenne” définie pour une paire  $(\hat{x}, \hat{y})$ .

### 3 Exemple détaillé : l'approximation polynomiale de la fonction Polygamma $\Psi(1, x)$

#### 3.1 La fonction Polygamma $\Psi(1, x)$ et son développement en série entière

Nous allons prendre un cas d'école : la fonction Polygamma  $\Psi(1, x)$ , c'est à dire :

$$\Psi(1, x) = \frac{d\Psi(x)}{dx}$$

avec

$$\Psi(x) = \frac{d \log \Gamma(x)}{dx} = \frac{\Gamma'(x)}{\Gamma(x)}$$

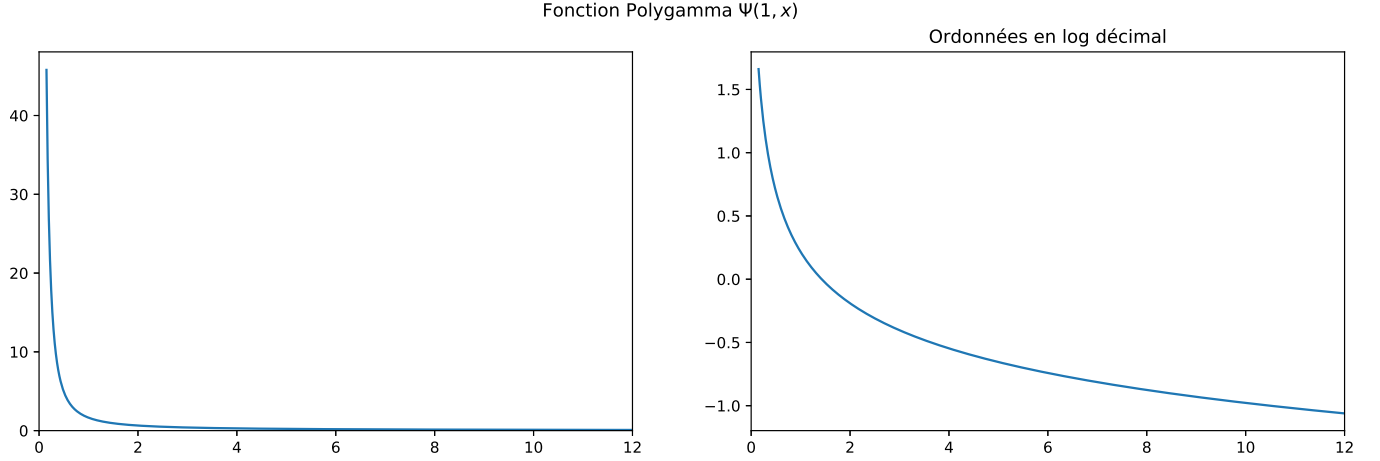


FIGURE 1 – Fonction Polygamma  $\Psi(1, x)$  en échelle linéaire (à gauche) et en échelle semi logarithmique (à droite).

C'est une fonction strictement monotone, représentée figure 1.

Une approximation en série classique de la fonction  $\Psi(1, x)$  s'obtient par le biais de l'expression suivante :

$$\Psi(1, x) = \sum_{k=0}^{\infty} \frac{1}{(x+k)^2}$$

Sur la figure 2, on représente la fonction et son développement en série avec  $k_{\max} = 100$  (en pointillé) ainsi que l'erreur relative. On peut remarquer que ce développement en série est très performant au voisinage de l'origine, mais qu'il donne quand même 10% d'erreur pour  $x \simeq 11.5$  (50% d'erreur pour  $k_{\max} = 10$ ).

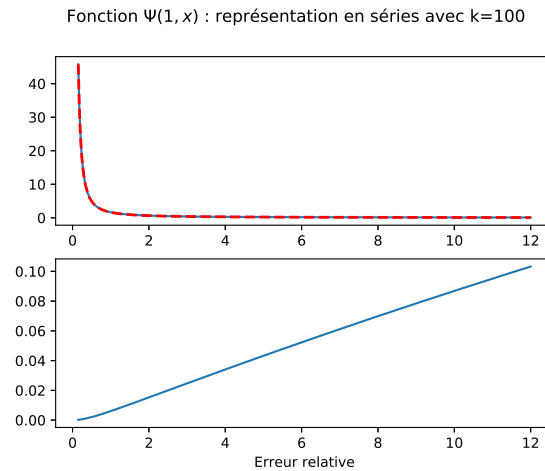


FIGURE 2 – Développement en série de la fonction Polygamma  $\Psi(1, x)$  (en haut). Erreur relative de ce modèle (en bas).

Sur ce cas de figure facile, nous allons analyser les résultats obtenus pour la méthode directe et la méthode itérative “à la Tichonov” dans diverses configurations de RPM :

— le cas  $L = 1, M = 1$

— le cas  $L = 2, M = 2$

— le cas  $L = 3, M = 3$

Les bornes de l'analyse sont fixées à  $[0.20; 10.]$ .

Pour chaque modèle étudié, seront tracés dans un premier graphe la fonction (en trait continu) et son approximation (en trait pointillé) et dans un second graphe la valeur de l'erreur relative.

### 3.2 Méthode directe

— un premier modèle, avec un numérateur de degré 1 et un dénominateur de degré 1 :

$$\frac{1.000000 - 0.027280 x}{-0.126225 + 0.765172 x}$$

L'erreur relative maximale est de l'ordre de 20%.

— un second modèle, avec un numérateur de degré 2 et un dénominateur de degré 2 :

$$\frac{1.000000 + 2.550788 x + 0.002247 x^2}{0.047378 - 0.464109 x + 2.586889 x^2}$$

L'erreur relative maximale est de l'ordre de 0.8% (sauf en début de courbe pour de faibles valeurs de  $x$ ).

— un dernier modèle, avec un numérateur de degré 3 et un dénominateur de degré 3 :

$$\frac{1.000000 + 2.227731 x + 3.335376 x^2 - 0.000080 x^3}{-0.006725 + 0.085115 x + 0.578380 x^2 + 3.333423 x^3}$$

L'erreur relative maximale est de l'ordre de 0.06%.

On peut remarquer les points suivants :

- comme on pouvait s'y attendre, la qualité de l'approximation s'améliore avec le degré des polynômes.
- avec seulement 3 coefficients, le maximum de l'erreur relative est comparable à celui obtenu par un développement en série. Avec 5 coefficients, le résultat est bien meilleur qu'avec le développement en série (avec 100 coefficients).
- les maxima des erreurs relatives sont obtenus pour des valeurs proches de l'origine.

### 3.3 Méthode itérative “à la Tichonov”

— un premier modèle, avec un numérateur de degré 1 et un dénominateur de degré 1 :

$$\frac{1.000000 - 0.109237 x}{-0.063933 + 0.504235 x}$$

L'erreur relative maximale est de l'ordre de 40% (sauf en fin de courbe pour de grandes valeurs de  $x$ ).

— un second modèle, avec un numérateur de degré 2 et un dénominateur de degré 2 :

$$\frac{1.000000 + 1.558122 x + 0.015469 x^2}{0.011845 - 0.148716 x + 1.696399 x^2}$$

L'erreur relative maximale est de l'ordre de 1% (sauf en fin de courbe pour de grandes valeurs de  $x$ ).

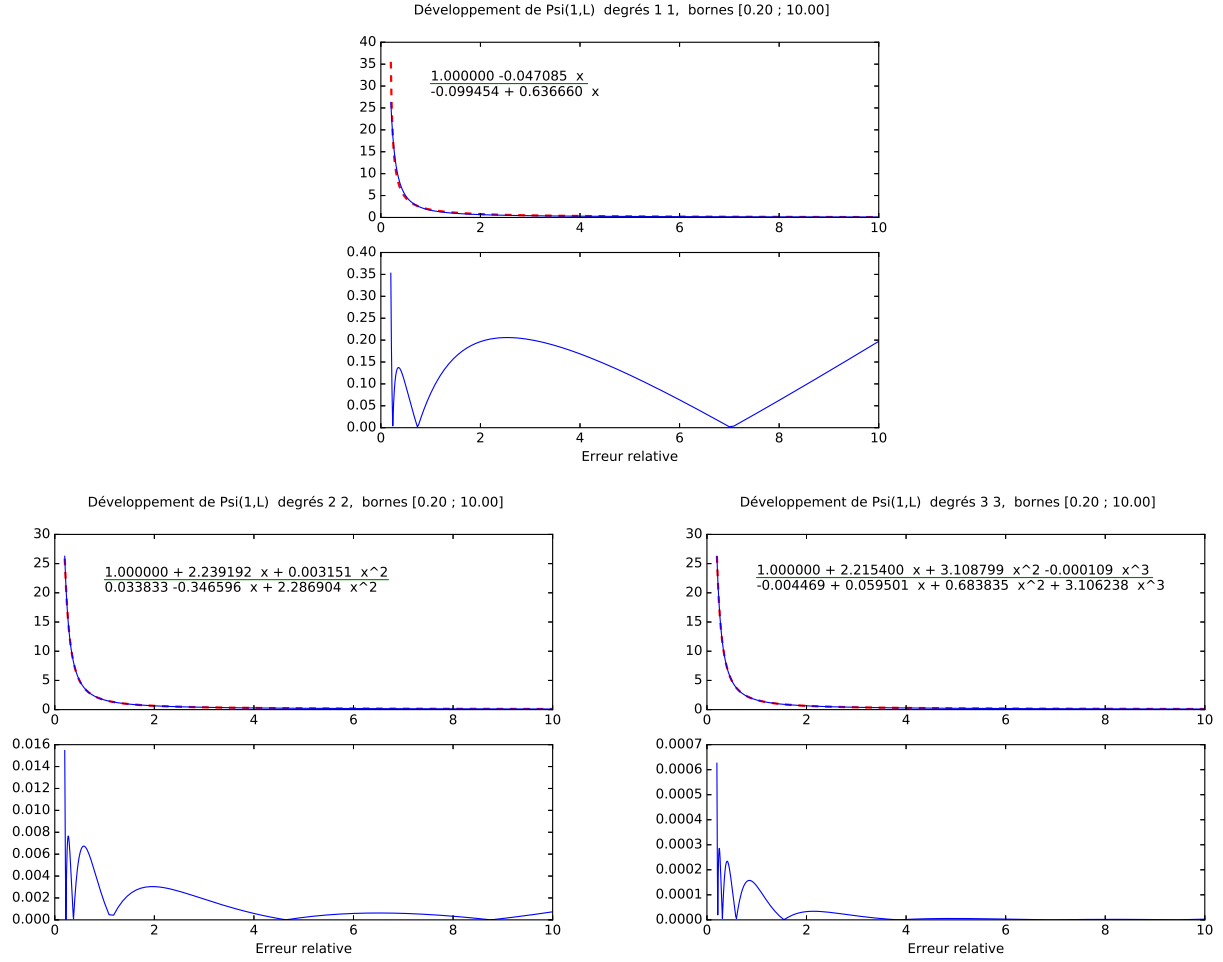


FIGURE 3 – Approximation polynomiale de la fonction polygamma  $\Psi(1, x)$ . Méthode directe. Trois cas sont étudiés :  $(L=1, M=1)$ ,  $(L=2, M=2)$  et  $(L=3, M=3)$ . Sont tracés la courbe initiale et la courbe approximée, ainsi que l'erreur relative.

— un dernier modèle, avec un numérateur de degré 3 et un dénominateur de degré 3 :

$$\frac{1.000000 + 2.223736 x + 3.197486 x^2 - 0.000097 x^3}{-0.005144 + 0.068075 x + 0.645944 x^2 + 3.195180 x^3}$$

L'erreur relative maximale est de l'ordre de 0.03%.

On peut remarquer les points suivants :

- comme on pouvait s'y attendre, la qualité de l'approximation s'améliore avec le degré des polynômes.
- avec seulement 5 coefficients, le maximum de l'erreur relative est beaucoup plus petit qu'avec le développement en série (avec 100 coefficients).
- les maxima des erreurs relatives sont obtenus pour des valeurs proches de l'origine lorsque les degrés des polynômes sont élevés.



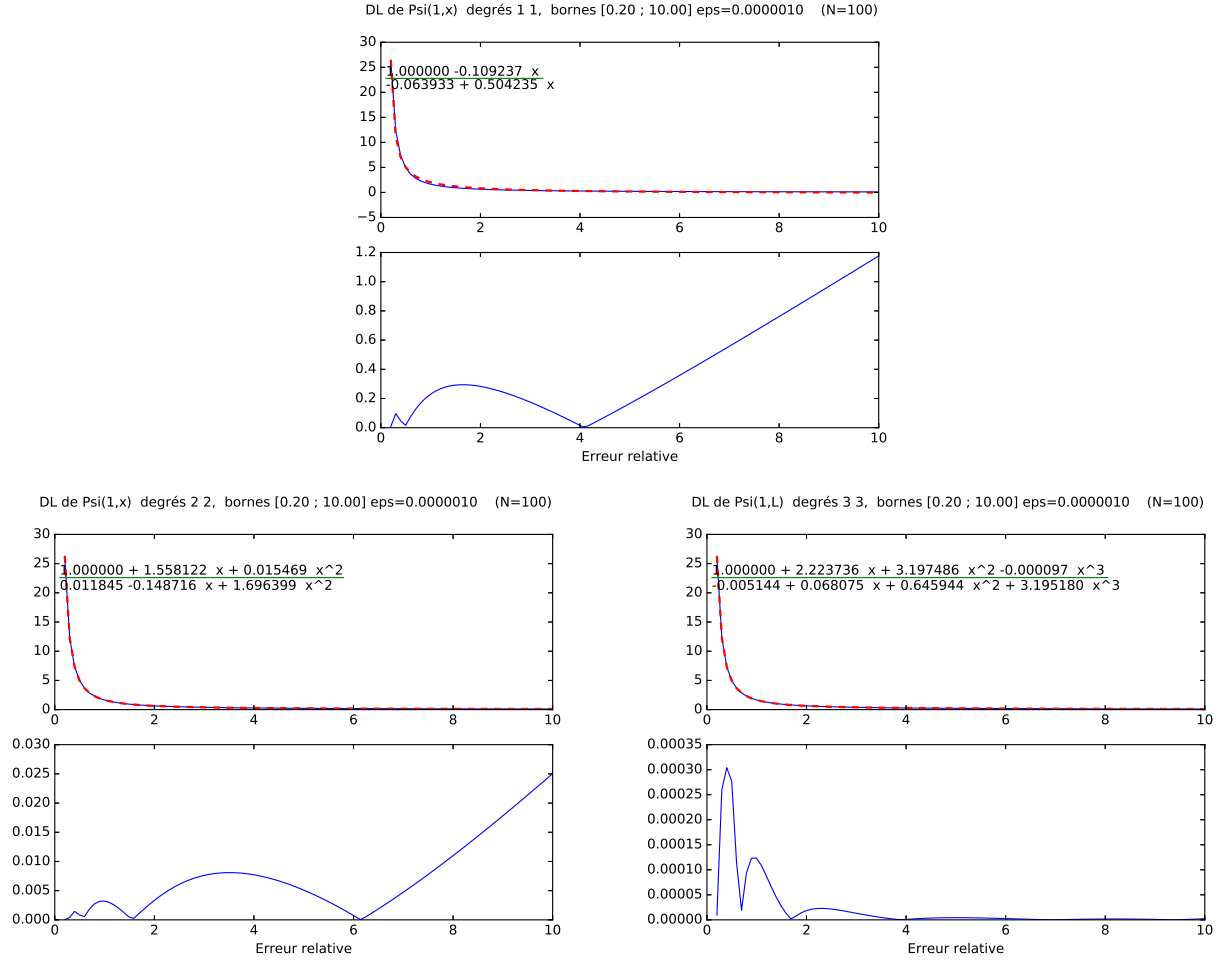


FIGURE 4 – Approximation polynomiale de la fonction polygamma  $\Psi(1, x)$ . Méthode itérative “à la Tichonov”. Trois cas sont étudiés :  $(L=1, M=1)$ ,  $(L=2, M=2)$  et  $(L=3, M=3)$ . Sont tracés la courbe initiale et la courbe approximée, ainsi que l’erreur relative.

### 3.4 Comparaison des modèles étudiés

Si le modèle direct peut avoir certaines qualités pour des polynômes à très faible nombre de coefficients, on remarque que le modèle “à la Tikhonov” est beaucoup plus efficace pour des polynômes de degré égal ou supérieur à 3. Diverses expérimentations effectuées par ailleurs sur de polynômes de degrés élevés confirment cette impression.

Par rapport au développement en séries de la fonction  $\Psi(1, x)$ , un RPM présente un caractère beaucoup plus universel tout en requérant beaucoup moins de coefficients. On peut tout de même noter que ce développement en série entière, qui nécessite comme souvent énormément de coefficients pour être utilisé, peut s’approximer par d’autres types de développement plus sophistiqués mais aussi plus spécialisés. Cependant, l’utilisation d’un RPM permet d’obtenir une approximation dans une gamme de valeurs usuelles définie par l’utilisateur, ce qui est très utile quand on a un a priori sur ce domaine.

Enfin, approximation et inversion sont abordées de la même manière, d’autant que cette fonction  $\Psi(1, x)$  est strictement monotone. Il est alors envisageable d’avoir un RPM pour l’approximation et un RPM pour l’inversion qui permet de retrouver l’identité si on utilise ces deux

RPM en *pipe line* : il suffit pour cela de construire correctement le jeu de données pour l'inversion à partir des résultats de l'approximation.

## 4 Applications aux statistiques de Mellin : approximation et inversion des fonctions Polygamma

Cette section va donner un certain nombre de modèles RPM dédié à l'approximation et à l'inversion de fonctions Polygamma dédiées à l'imagerie cohérente. Les relations proposées sont donc dédiées à ce domaine et ne doivent en aucun cas être prises comme des références absolues. Certains choix, spécifiques à l'imagerie cohérente, ont été pris, comme les bornes de définition des modèles : il va sans dire que, selon les cas étudiés, d'autres modélisations existent et peuvent tout à fait s'avérer beaucoup plus judicieuses que celles proposées dans ce document.

### 4.1 Le rôle des fonction Polygamma en statistiques de Mellin

Le domaine applicatif de ce document est celui des statistiques d'images cohérentes, c'est à dire celles dont le mécanisme d'acquisition conduit à l'existence d'un bruit multiplicatif (chaotement).

Plus spécifiquement, nous allons aborder, tant pour l'approximation que pour l'inversion, le cas des fonctions Digamma et Polygamma.

La fonction  $\Psi(r, x)$  (fonction Polygamma d'ordre  $r$ ) est la dérivée  $r + 1$ -ème logarithmique de la fonction Gamma. A ce titre, il est possible d'en donner des approximations polynomiales, par exemple grâce au développement de Lanczos de la fonction Gamma. Comme nous l'avons déjà vu, elles peuvent avoir des expressions sous forme de séries entières (paragraphe 3). Le cas  $r = 0$ , c'est à dire la fonction  $\Psi(0, x)$ , est celui de la fonction Digamma, qui est le plus souvent notée  $\Psi(x)$ .

Ces fonctions sont très usitées en imagerie cohérente. En effet, dans le cadre des lois Gamma et Nakagami, on rencontre les systèmes vérifiés par les log-cumulants d'ordre 1 ( $\tilde{\kappa}_1$ ) et 2 ( $\tilde{\kappa}_2$ ) en fonction des paramètres de ces lois ( $\mu$  pour le paramètre d'échelle et  $L$  pour le paramètre de forme) :

— pour la loi Gamma

$$\begin{cases} \tilde{\kappa}_1 &= \log(\mu) + \Psi(L) - \log(L) \\ \tilde{\kappa}_2 &= \Psi(1, L) \end{cases} \quad (29)$$

— pour la loi de Nakagami

$$\begin{cases} \tilde{\kappa}_1 &= \log(\mu) + \frac{1}{2}(\Psi(L) - \log(L)) \\ \tilde{\kappa}_2 &= \frac{1}{4}\Psi(1, L) \end{cases} \quad (30)$$

Avoir des RPM de ces fonctions, tant pour leur approximation que pour leur inversion, s'avère d'une grande utilité dans le traitement de ce type d'images, et un des objectifs que nous fixerons à ces RPM est d'avoir des expressions robustes ainsi qu'un nombre limité de coefficients.

### 4.2 Fonctions Polygamma : modèle RPM

L'objectif est de proposer des RPM utilisables de manière courante évitant des appels chronophages à des fonctions dites spéciales, ou des résolutions de système à minimiser. Plusieurs critères vont donc être utilisés :

— la minimisation de l'erreur quadratique utilisée dans le processus de recherche des RPC ;

$\Psi(x)$	$\simeq \frac{1.000000 + 0.712948 x - 0.837487 x^2 - 0.081094 x^3}{-0.009079 - 0.916387 x + 0.436530 x^2 - 0.016771 x^3}$
$\Psi(x) - \log(x)$	$\simeq \frac{1.000000 + 3.068163 x + 3.359825 x^2 - 0.000013 x^3}{0.006222 - 1.132967 x - 5.022964 x^2 - 6.718980 x^3}$
$\Psi(1, x)$	$\simeq \frac{1.000000 + 2.227731 x + 3.335376 x^2 - 0.000080 x^3}{-0.006725 + 0.085115 x + 0.578380 x^2 + 3.333423 x^3}$
$\Psi(2, x)$	$\simeq \frac{1.000000 + 1.014673 x + 0.008057 x^2 - 0.000352 x^3}{0.006304 - 0.072868 x + 0.303145 x^2 - 1.079702 x^3}$
$\Psi(3, x)$	$\simeq \frac{1.000000 + 0.145391 x^2 + 0.001224 x^4 - 0.000002 x^6}{-0.000018 + 0.000505 x^2 + 0.165628 x^4 + 0.009387 x^6}$

TABLE 1 – Approximation de fonctions Polygamma

- la minimisation de l’erreur relative en tout point de la base de données ;
- la minimisation de l’erreur quadratique moyenne relative.

Pour trouver le “meilleur” RPM, on va utiliser une méthode gloutonne consistant à :

- balayer tous les cas possibles de RPM pour des degrés de polynômes (numérateur et dénominateur) variant de la valeur 1 à une valeur  $N_{pol}$  donnée.
- balayer plusieurs valeurs pour le paramètre  $\varepsilon$  pilotant l’optimisation “à la Tichonov” (incluant la valeur nulle, ce qui revient à chercher à résoudre le problème direct)
- considérer les 3 cas possibles dans lesquels la variable du polynôme est  $x$ ,  $\sqrt{x}$  et  $x^2$ .

Nous avons choisi que les RPM vérifient comme critère la minimisation de l’erreur quadratique moyenne relative. Par souci de rationalisation, le nombre de coefficients a été, en général, volontairement limité à 7 ( $N_{pol}=3$ ). Enfin, les expressions données dans ce document ont été automatiquement générées en Latex : au lecteur de vérifier les choix à effectuer pour limiter le nombre de chiffres significatifs de certains coefficients.

Les résultats sont synthétisés dans deux tableaux : le tableau 1 pour l’approximation et le tableau 2 pour l’inversion. Dans le cas de l’inversion, quelques figures synthétiques illustrent la qualité de l’inversion ainsi que l’erreur relative.

$x = \Psi(1, L)$	$L \simeq \frac{1.000000 + 0.899712 x + 0.106635 x^2 + 0.000891 x^3}{-0.000085 + 1.001475 x + 0.390803 x^2 + 0.015221 x^3}$ <p>Figure 5 page 20</p>
$x = \Psi(2, L)$	$L \simeq \frac{1.000000 - 1.040182 x^{0.5} + 0.130305 x - 0.000886 x^{1.5}}{-0.000143 - 1.002479 x^{0.5} + 0.524941 x - 0.030360 x^{1.5}}$ <p>Figure 5 page 20</p>
$x = \Psi(3, L)$	$L \simeq \frac{1.000000 + 14.721010 x^{0.5} + 21.765669 x + 3.263541 x^{1.5} + 0.023304 x^2}{0.019003 + 2.541448 x^{0.5} + 14.565046 x + 7.348012 x^{1.5} + 0.288223 x^2}$ <p>Figure 6 page 20</p>
$x = \Psi(L)$	$L \simeq \frac{1.000000 + 0.053442 x + 0.046578 x^2 + 0.002517 x^3}{0.683940 - 0.445950 x + 0.118939 x^2 - 0.012645 x^3}$ <p>Figure 6 page 20</p>
$x = \Psi(L) - \log(L)$	$L \simeq \frac{1.000000 - 31.158014 x + 27.024720 x^2 - 0.292582 x^3}{0.000118 - 1.994320 x + 61.750886 x^2 - 32.866199 x^3}$ <p>Figure 7 page 21</p>

TABLE 2 – Inversion de fonctions Polygamma

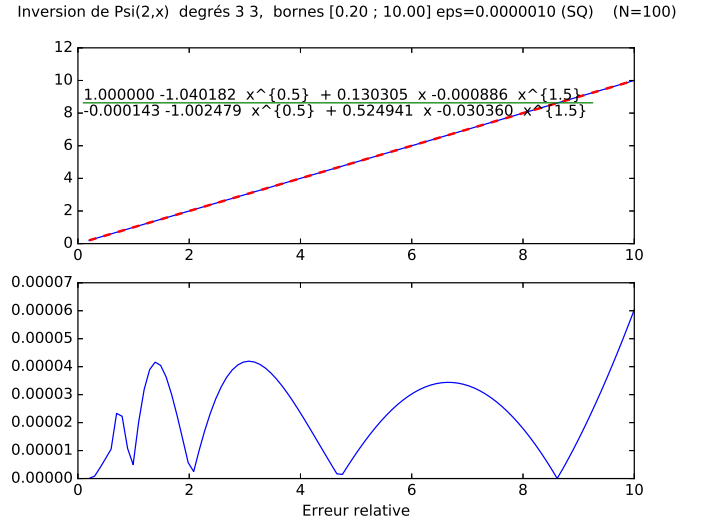
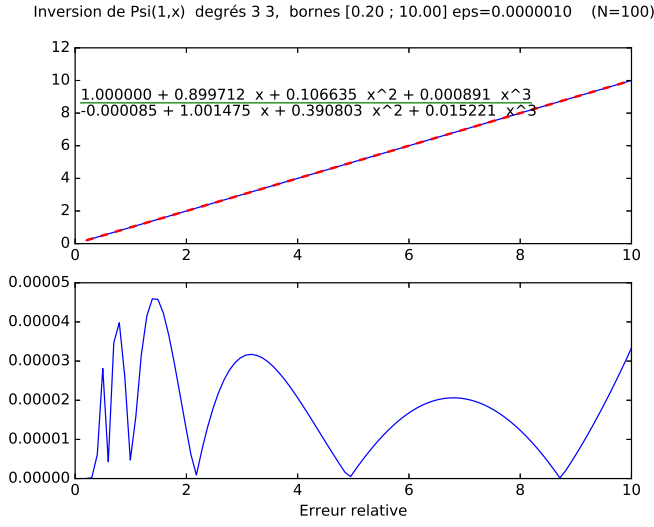


FIGURE 5 – A gauche : Inversion de  $\Psi(1,x)$  : polynômes de degré 3. A droite : Inversion de  $\Psi(2,x)$  : polynôme de degré 3 en  $\sqrt{x}$ .

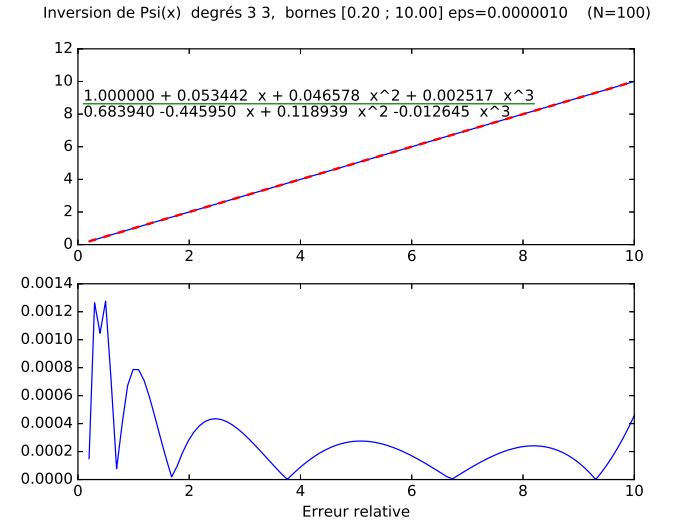
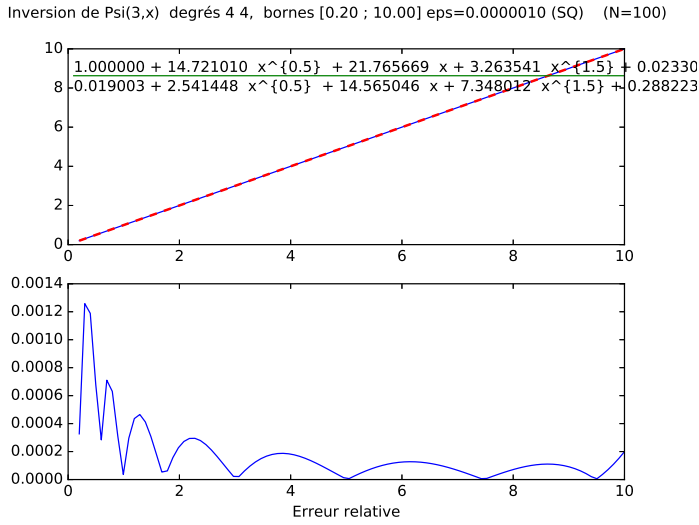


FIGURE 6 – A gauche : inversion de  $\Psi(3,x)$  : polynôme de degré 4 en  $\sqrt{x}$ . A droite : inversion de  $\Psi(x)$  : polynôme de degré 3.

Inversion de  $\Psi(x) - \log(x)$  degrés 3 3, bornes [0.20 ; 10.00] eps=0.0000010 (N=100)

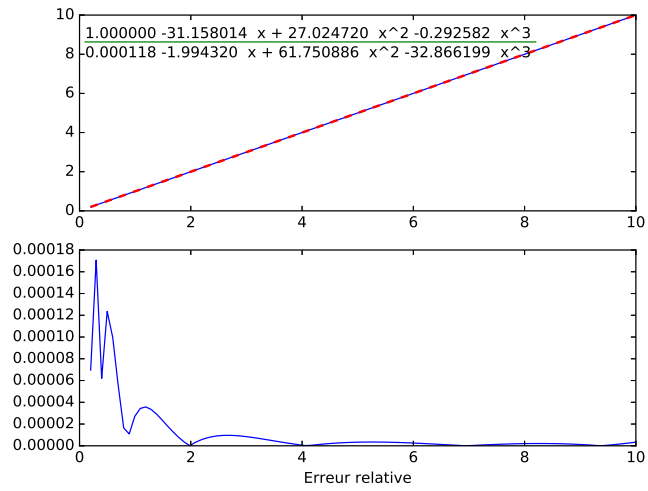


FIGURE 7 – Inversion de  $\Psi(x) - \log(x)$  : polynome de degré 3

## 5 Conclusion

La modélisation par ratio de polynômes (RPM) s'avère un outil mathématiquement bien fondé et d'utilisation pratique. Elle peut aussi bien s'utiliser pour approximer des fonctions "compliquées" aussi bien que pour inverser ces mêmes fonctions dont on ne connaît pas la forme analytique de l'inverse. On ne peut que s'étonner d'un emploi le plus souvent cantonné au recalage d'images satellitaires alors que ce type de modèle est tout à fait bien adapté à de nombreux cas rencontrés dans d'autres cas pratiques.

On trouvera en annexe A des expressions directement utilisables pour le traitement d'images en imagerie cohérente, et en annexe B un descriptif rapide des outils `rpclab` (Python) relatifs à la plateforme `sarlab` du groupe "images" du département IDS de Télécom ParisTech.

## A Quelques relations utiles pour l'imagerie cohérente

Cette annexe propose quelques expressions utiles pour manipuler des lois usuelles en imagerie cohérente : loi Gamma, loi K, loi de Rayleigh Nakagami et loi de Rayleigh Nakagami Inverse, loi de Rice, loi de Weibull et loi Gamma Généralisée, ainsi que des exemples de lois ayant deux facteurs de forme (loi de Fisher et loi Beta). Les paramètres choisis (définition de la "boîte à chaussure", valeur du paramètre  $\varepsilon$ , ...) correspondent à des domaines utiles en imagerie cohérente, ce qui rend ces expressions opérationnelles dans la plupart des cas rencontrés en traitement d'images cohérentes. Les expressions proposées ont été automatiquement générées en Latex, au lecteur de fixer un nombre de chiffres significatifs réalistes pour son application.

Comme indiqué au paragraphe 4, d'autres modélisations existent et peuvent tout à fait s'avérer beaucoup plus judicieuses que celles proposées dans ce document.

### A.1 Loi Gamma et loi K (lois en intensité)

#### A.1.1 Utilitaire de la loi Gamma : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$

Les deux log-cumulants de la loi Gamma  $\mathcal{G}[\mu, L]$  sont liés par une relation biunivoque ne dépendant que du paramètre  $L$  puisque l'on a :

$$\begin{aligned}\tilde{\kappa}_2 &= \Psi(1, L) \\ \tilde{\kappa}_3 &= \Psi(2, L)\end{aligned}$$

Il est tout à fait possible de trouver un RPC donnant directement  $\tilde{\kappa}_3$  à partir de  $\tilde{\kappa}_2$  sans passer par l'étape d'inversion du second log-cumulant. On obtient :

$$\hat{\tilde{\kappa}}_{3,G} \simeq \frac{-0.000143 - 0.993792 \hat{\tilde{\kappa}}_2^2 - 0.249196 \hat{\tilde{\kappa}}_2^4 - 0.004537 \hat{\tilde{\kappa}}_2^6}{1.000000 + 0.313414 \hat{\tilde{\kappa}}_2^2 + 0.009037 \hat{\tilde{\kappa}}_2^4 + 0.000008 \hat{\tilde{\kappa}}_2^6} \quad (31)$$

Cette expression permet, dans le diagramme  $\tilde{\kappa}_2 - \tilde{\kappa}_3$ , de savoir si une loi est dans la zone des lois de Fisher ou des lois Beta. En effet, connaissant  $\tilde{\kappa}_2$  et en appliquant la relation 35, on obtient la valeur test  $\tilde{\tilde{\kappa}}_{3,G}$  de sorte que :

- si  $\tilde{\kappa}_3 < \tilde{\tilde{\kappa}}_{3,G}$ , alors on a une loi Beta
- si  $\tilde{\tilde{\kappa}}_{3,G} \leq \tilde{\kappa}_3 \leq -\tilde{\tilde{\kappa}}_{3,G}$ , alors on a une loi de Fisher
- si  $\tilde{\kappa}_3 > -\tilde{\tilde{\kappa}}_{3,G}$ , alors on a une loi Beta inverse

### A.1.2 Utilitaire des caustiques de la loi K : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$

Le cas des lois  $\mathcal{K}$  s'exprimant sous la forme  $\mathcal{K}[\mu, L, L]$  donne un cas limite des lois  $\mathcal{K}$  (l'autre étant le cas  $\lim_{M \rightarrow \infty} \mathcal{K}[\mu, L, M] = \mathcal{G}[\mu, L]$ ). Dans ce cas, le log-cumulant d'ordre 3 est lié au log-cumulant d'ordre 2 puisque l'on a :

$$\begin{aligned}\tilde{\kappa}_2 &= 2\Psi(1, L) \\ \tilde{\kappa}_3 &= 2\Psi(2, L)\end{aligned}$$

Il est alors possible de construire un RPM permettant d'approximer  $\tilde{\kappa}_3$  en fonction de  $\tilde{\kappa}_2$ , ce qui donne :

$$\hat{\kappa}_{3,K} \simeq \frac{1.000000 - 8.717075 \hat{\kappa}_2 + 230.993168 \hat{\kappa}_2^2 - 66.375044 \hat{\kappa}_2^3}{-422.510885 + 95.213370 \hat{\kappa}_2 + 8.342860 \hat{\kappa}_2^2 - 0.103926 \hat{\kappa}_2^3} \quad (32)$$

Cette expression a été obtenue sur l'intervalle  $L \in [0.3, 20]$  et l'erreur relative maximale est inférieure à 0.06%.

On trouve de manière équivalente la relation suivante :

$$\hat{\kappa}_{2,K} \simeq \frac{1.000000 - 122.852492 \hat{\kappa}_3 + 319.997616 \hat{\kappa}_3^2 - 34.750552 \hat{\kappa}_3^3}{14.029426 - 212.537862 \hat{\kappa}_3 + 105.369924 \hat{\kappa}_3^2 - 1.186557 \hat{\kappa}_3^3} \quad (33)$$

Les expressions 31 et 32 permettent de déterminer si une loi est potentiellement une loi  $\mathcal{K}$  grâce à un test sur  $\tilde{\kappa}_3$  connaissant  $\tilde{\kappa}_2$ . En effet, connaissant  $\tilde{\kappa}_2$ , on calcule  $\tilde{\kappa}_{3,G}$  (relation 31) et  $\tilde{\kappa}_{3,K}$  (relation 32). Si on a :

$$\tilde{\kappa}_{3,G} \leq \tilde{\kappa}_3 \leq \tilde{\kappa}_{3,K}$$

alors la loi dont les log-cumulants d'ordre 2 et 3 sont  $\tilde{\kappa}_2$  et  $\tilde{\kappa}_3$  peut être décrite par une loi  $\mathcal{K}$ .

## A.2 Loi de Rayleigh Nakagami et loi K en amplitude

### A.2.1 Inversion du coefficient de variation de la loi de Rayleigh Nakagami

Dans le cas de la loi de Nakagami  $\mathcal{RN}[\mu, L](x)$ , on trouve l'expression reliant le coefficient de variation  $\gamma$  et le facteur de forme  $L$  :

$$\gamma = \sqrt{\frac{\Gamma(L)\Gamma(L+1)}{\left(\Gamma(L+\frac{1}{2})\right)^2} - 1}$$

expression qui met en œuvre la fonction de Pochhammer et qui n'a pas d'inverse analytique. Connaissant  $\gamma$ , l'inversion numérique donne :

— une première expression (polynômes de degrés 3, figure 8) :

$$L \simeq \frac{1 + 0.06820 \gamma - 0.099691 \gamma^2 - 0.0487127 \gamma^3}{0.00123 - 0.03460 \gamma + 4.340579 \gamma^2 - 1.1729569 \gamma^3} \quad (34)$$

— une seconde expression (polynômes de degrés 2) :

$$L \simeq \frac{1.000000 + 0.482198 \gamma - 0.162729 \gamma^2}{0.003178 - 0.067854 \gamma + 4.539311 \gamma^2}$$



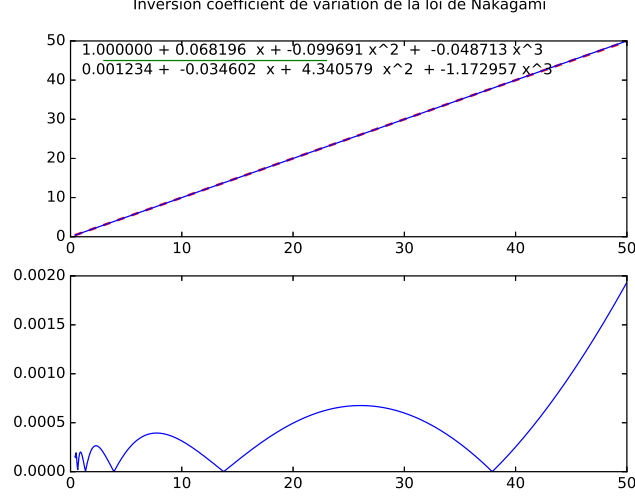


FIGURE 8 – Inversion du coefficient de variation de la loi de Nakagami (polynômes de degrés 3).

### A.2.2 Inversion du coefficient de variation de la loi de Rayleigh Nakagami Inverse

Dans le cas de la loi de Nakagami Inverse  $\mathcal{RN}[\mu, L](x)$ , on trouve l'expression reliant le coefficient de variation  $\gamma$  et le facteur de forme  $M$  :

$$\gamma = \sqrt{\frac{\Gamma(M)\Gamma(M-1)}{\Gamma(M-\frac{1}{2})^2} - 1} \quad M > 1$$

L'inversion numérique donne :

$$M \simeq \frac{1.000000 + 1.080088 \gamma + 4.764801 \gamma^2 + 1.963031 \gamma^3}{0.010254 - 0.172509 \gamma + 5.048353 \gamma^2 + 1.917664 \gamma^3}$$

### A.2.3 Utilitaire de la loi de Rayleigh-Nakagami : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$

Les deux log-cumulants de la loi de Rayleigh-Nakagami  $\mathcal{RN}[\mu, L]$  sont liés par une relation biunivoque ne dépendant que du paramètre  $L$  puisque l'on a :

$$\begin{aligned} {}_{3,G}\tilde{\kappa}_2 &= \frac{1}{4}\Psi(1, L) \\ \tilde{\kappa}_3 &= \frac{1}{8}\Psi(2, L) \end{aligned}$$

Il est tout à fait possible de trouver un RPC donnant directement  $\tilde{\kappa}_3$  à partir de  $\tilde{\kappa}_2$  sans passer par l'étape d'inversion du second log-cumulant. On obtient :

$$\tilde{\kappa}_{3,RN} = \frac{-0.000002 - 1.994173 \tilde{\kappa}_2^2 - 5.448133 \tilde{\kappa}_2^4 + 0.668981 \tilde{\kappa}_2^6 - 0.012186 \tilde{\kappa}_2^8}{1.000000 + 3.797440 \tilde{\kappa}_2^2 - 0.306149 \tilde{\kappa}_2^4 - 0.008805 \tilde{\kappa}_2^6 + 0.000241 \tilde{\kappa}_2^8} \quad (35)$$

Cette expression permet, dans le diagramme  $\tilde{\kappa}_2 - \tilde{\kappa}_3$ , de savoir si une loi est dans la zone des lois de Fisher en amplitude ou des lois Beta en amplitude. En effet, connaissant  $\tilde{\kappa}_2$  et en appliquant la relation 35, on obtient une valeur test  $\tilde{\kappa}_{3,RN}$  de sorte que :

- si  $\tilde{\kappa}_3 < \tilde{\kappa}_{3,RN}$ , alors on a une loi Beta en amplitude
- si  $\tilde{\kappa}_{3,RN} \leq \tilde{\kappa}_3 \leq -\tilde{\kappa}_{3,RN}$ , alors on a une loi de Fisher en amplitude
- si  $\tilde{\kappa}_3 > -\tilde{\kappa}_{3,RN}$ , alors on a une loi Beta en amplitude inverse

#### A.2.4 Utilitaire de la caustique des lois K en amplitude : expression de $\tilde{\kappa}_3$ en fonction de $\tilde{\kappa}_2$

Le cas des lois  $\mathcal{KA}$  s'exprimant sous la forme  $\mathcal{KA}[\mu, L, L]$  donne un cas limite des lois  $\mathcal{KA}$  (l'autre étant le cas  $\lim_{M \rightarrow \infty} \mathcal{KA}[\mu, L, M] = \mathcal{RN}[\mu, L]$ ). Dans ce cas, le log-cumulant d'ordre 3 est lié au log-cumulant d'ordre 2 puisque l'on a :

$$\begin{aligned}\tilde{\kappa}_2 &= 2 \frac{\Psi(1, L)}{4} \\ \tilde{\kappa}_3 &= 2 \frac{\Psi(2, L)}{8}\end{aligned}$$

Il est alors possible de construire un RPM permettant d'approximer  $\tilde{\kappa}_3$  en fonction de  $\tilde{\kappa}_2$ , ce qui donne :

$$\hat{\tilde{\kappa}}_{3,KA} \simeq \frac{1.000000 - 38.898309 \hat{\tilde{\kappa}}_2 + 5833.131819 \hat{\tilde{\kappa}}_2^2 - 1308.481140 \hat{\tilde{\kappa}}_2^3}{-5449.747994 - 95.776824 \hat{\tilde{\kappa}}_2 + 335.401372 \hat{\tilde{\kappa}}_2^2 - 6.785810 \hat{\tilde{\kappa}}_2^3} \quad (36)$$

Cette expression a été obtenue sur l'intervalle  $L \in [0.3, 20]$  et l'erreur relative maximale est inférieure à 0.06%.

On trouve de manière équivalente la relation suivante :

$$\hat{\tilde{\kappa}}_{2,KA} \simeq \frac{1.000000 - 422.266529 \hat{\tilde{\kappa}}_3 + 2108.362798 \hat{\tilde{\kappa}}_3^2 - 67.743218 \hat{\tilde{\kappa}}_3^3}{40.246459 - 1515.244224 \hat{\tilde{\kappa}}_3 + 870.720866 \hat{\tilde{\kappa}}_3^2 + 53.787895 \hat{\tilde{\kappa}}_3^3} \quad (37)$$

Les expressions 35 et 36 permettent de déterminer si une loi est potentiellement une loi  $\mathcal{KA}$  (loi K en amplitude) grâce à un test sur  $\tilde{\kappa}_3$  connaissant  $\tilde{\kappa}_2$ . En effet, connaissant  $\tilde{\kappa}_2$ , on calcule  $\tilde{\kappa}_{3,RN}$  (relation 35) et  $\tilde{\kappa}_{3,KA}$  (relation 36). Si on a :

$$\tilde{\kappa}_{3,RN} \leq \tilde{\kappa}_3 \leq \tilde{\kappa}_{3,KA}$$

alors la loi dont les log-cumulants d'ordre 2 et 3 sont  $\tilde{\kappa}_2$  et  $\tilde{\kappa}_3$  peut être décrite par une loi  $\mathcal{KA}$ .

#### A.2.5 De la loi de Nakagami à la loi Gamma

Soit une loi de Nakagami de facteur de forme  $L_{\mathcal{RN}}$ . On recherche la loi Gamma de facteur de forme  $L_{\mathcal{G}}$  tel que les premiers log-cumulants de ces deux lois soient égaux. On a alors :

$$\begin{aligned}\Psi(1, L_{\mathcal{G}}) &= \frac{1}{4} \Psi(1, L_{\mathcal{RN}}) \\ L_{\mathcal{G}} &\simeq \frac{1.000000 - 2.859582 L_{\mathcal{RN}}^{0.5} + 2.083235 L_{\mathcal{RN}} + 2.769941 L_{\mathcal{RN}}^{1.5}}{0.284726 + 0.756792 L_{\mathcal{RN}}^{0.5} - 0.008780 L_{\mathcal{RN}} + 0.000473 L_{\mathcal{RN}}^{1.5}}\end{aligned}$$

### A.2.6 De la loi Gamma à la loi de Nakagami

Soit une loi Gamma de facteur de forme  $L_G$ . On recherche la loi de Nakagami de facteur de forme  $L_{\mathcal{RN}}$  tel que les premiers log-cumulants de ces deux lois soient égaux. On a alors la relation suivante à vérifier :

$$\Psi(1, L_{\mathcal{RN}}) = 4\Psi(1, L_G)$$

ce qui donne l'expression :

$$L_{\mathcal{RN}} \simeq \frac{1.000000 - 3.286071 L_G^{0.5} - 0.152404 L_G - 0.167044 L_G^{1.5}}{-6.804131 + 0.837084 L_G^{0.5} - 0.181715 L_G + 0.008751 L_G^{1.5}}$$


---

### A.3 Loi de Rice (type 2) : inversion du coefficient de variation

La loi de Rice peut se modéliser sous la forme (voir [2]) :

$$\mathcal{RC}_2[\mu, \lambda](x) = \frac{2x}{\mu^2} e^{-\left(\frac{x^2}{\mu^2} + \lambda^2\right)} I_0\left(2\lambda \frac{x}{\mu}\right)$$

Avec cette paramétrisation, le coefficient de variation s'exprime uniquement en fonction de  $\lambda$  :

$$\gamma = \sqrt{\frac{e^{\lambda^2} {}_1F_1(2; 1; \lambda^2)}{\left(\Gamma\left(1 + \frac{1}{2}\right) {}_1F_1\left(1 + \frac{1}{2}; 1; \lambda^2\right)\right)^2} - 1}$$

L'inversion numérique donne :

$$\lambda \simeq \frac{1.000000 - 5.086703 \gamma^{0.5} + 9.639727 \gamma - 8.056629 \gamma^{1.5} + 2.501736 \gamma^2}{0.006952 + 0.328444 \gamma^{0.5} - 1.418807 \gamma + 1.966276 \gamma^{1.5} - 0.899872 \gamma^2}$$

### A.4 Loi de Weibull : inversion du coefficient de variation

Dans le cas de la loi de Weibull  $\mathcal{W}[\mu, \eta]$  on trouve l'expression reliant le coefficient de variation  $\gamma$  et le facteur de forme  $\eta$  :

$$\gamma = \sqrt{\frac{\Gamma(1 + \frac{2}{\eta})}{\Gamma(1 + \frac{1}{\eta})^2} - 1} \quad \eta > 0 \text{ ou } \eta < -2$$

L'inversion numérique donne :

$$\eta \simeq \frac{1.000000 + 0.804336 \gamma + 0.453554 \gamma^2 + 0.023463 \gamma^3}{0.013085 + 0.665238 \gamma + 1.416508 \gamma^2 + 0.185671 \gamma^3}$$

### A.5 Gamma Généralisée : inversion de la fonction $\frac{\Psi(2, L)}{\Psi(1, L)^{1.5}}$

Dans le cas de la loi Gamma Généralisée  $\mathcal{GG}[\mu, L, \eta]$ , on montre que les log-cumulants d'ordre 2 et 3 vérifient les expressions suivantes :

$$\begin{aligned} \tilde{\kappa}_2 &= \frac{\Psi(1, L)}{\eta^2} \\ \tilde{\kappa}_3 &= \frac{\Psi(2, L)}{\eta^3} \end{aligned}$$

On en déduit que le rapport :

$$x = \frac{\tilde{\kappa}_3}{\tilde{\kappa}_2^{1.5}} = \frac{\Psi(2, L)}{\Psi(1, L)^{1.5}}$$

ne dépend que du facteur de forme  $L$ . L'inversion numérique donne :

$$\hat{L} \simeq \frac{1.000000 + 0.653026 x^2 - 0.274110 x^4 + 0.014915 x^6}{-0.000058 + 1.001300 x^2 + 0.142117 x^4 - 0.055452 x^6}$$

## A.6 Lois à 3 variables

### A.6.1 Loi K

L'estimation des paramètres de forme de la loi K pose quelques problèmes d'optimisation et aucune relation fiable n'a pu être trouvée. Une origine plausible de ces problèmes pourrait venir de la relation de symétrie suivante :

$$\mathcal{K}[\mu, L, M] = \mathcal{K}[\mu, M, L]$$

### A.6.2 Loi de Fisher

Soit une loi de Fisher  $\mathcal{F}[\mu, L, M]$ . Ses log-cumulants d'ordre 2 et 3 s'expriment uniquement en fonction des facteurs de forme  $L$  et  $M$  :

$$\begin{aligned}\tilde{\kappa}_2 &= \Psi(1, L) + \Psi(1, M) \\ \tilde{\kappa}_3 &= \Psi(2, L) - \Psi(2, M)\end{aligned}$$

Si on connaît les valeurs de  $\tilde{\kappa}_2$  et  $\tilde{\kappa}_3$ , on peut inverser le système par le RPM suivant :

$$\begin{aligned}L &\simeq \frac{\left( 1. - 36.696 \tilde{\kappa}_2 + 9.462 \tilde{\kappa}_3 + 0.688 \tilde{\kappa}_2 \tilde{\kappa}_3 - 7.772 \tilde{\kappa}_2^2 + 0.050 \tilde{\kappa}_3^2 \right)}{\left( 0.024 + 0.231 \tilde{\kappa}_2 + 16.414 \tilde{\kappa}_3 + 2.222 \tilde{\kappa}_2 \tilde{\kappa}_3 - 17.592 \tilde{\kappa}_2^2 - 0.344 \tilde{\kappa}_3^2 \right.} \\ &\quad \left. - 0.340 \tilde{\kappa}_2^3 + 0.466 \tilde{\kappa}_2^2 \tilde{\kappa}_3 - 0.079 \tilde{\kappa}_2 \tilde{\kappa}_3^2 + 0.005 \tilde{\kappa}_3^3 \right) \\ M &\simeq \frac{\left( 1. - 36.696 \tilde{\kappa}_2 - 9.462 \tilde{\kappa}_3 - 0.688 \tilde{\kappa}_2 \tilde{\kappa}_3 - 7.772 \tilde{\kappa}_2^2 + 0.050 \tilde{\kappa}_3^2 \right)}{\left( 0.024 + 0.231 \tilde{\kappa}_2 - 16.414 \tilde{\kappa}_3 - 2.222 \tilde{\kappa}_2 \tilde{\kappa}_3 - 17.592 \tilde{\kappa}_2^2 - 0.344 \tilde{\kappa}_3^2 \right.} \\ &\quad \left. - 0.340 \tilde{\kappa}_2^3 - 0.466 \tilde{\kappa}_2^2 \tilde{\kappa}_3 - 0.079 \tilde{\kappa}_2 \tilde{\kappa}_3^2 - 0.005 \tilde{\kappa}_3^3 \right)\end{aligned}$$

La validité de ces deux relations peut s'analyser à l'aide du tableau suivant :

	$M_0 = 0.50$	$M_0 = 1.00$	$M_0 = 2.00$	$M_0 = 5.00$
$L_0 = 0.50$	L = 0.50 M = 0.50	L = 0.50 M = 1.00	L = 0.50 M = 1.99	L = 0.50 M = 5.01
$L_0 = 1.00$	L = 1.00 M = 0.50	L = 0.99 M = 0.99	L = 1.00 M = 1.98	L = 1.02 M = 5.01
$L_0 = 2.00$	L = 1.99 M = 0.50	L = 1.98 M = 1.00	L = 1.99 M = 1.99	L = 2.02 M = 4.98
$L_0 = 5.00$	L = 5.01 M = 0.50	L = 5.01 M = 1.02	L = 4.98 M = 2.02	L = 5.00 M = 5.00

ce qui est tout à fait raisonnable en traitement d'images cohérentes.

### A.6.3 Loi Beta

Soit une loi Beta  $\mathcal{B}[\mu, L, M]$ . Ses log-cumulants d'ordre 2 et 3 s'expriment uniquement en fonction des facteurs de forme  $L$  et  $M$  :

$$\begin{aligned}\tilde{\kappa}_2 &= \Psi(1, L) - \Psi(1, M) \\ \tilde{\kappa}_3 &= \Psi(2, L) - \Psi(2, M)\end{aligned}$$

Si on connaît les valeurs de  $\tilde{\kappa}_2$  et  $\tilde{\kappa}_3$ , on peut inverser le système par le RPM suivant :

$$\begin{aligned}L &\simeq \left( \frac{1. + 1928.4 \tilde{\kappa}_2 + 57.455 \tilde{\kappa}_3^{\frac{2}{3}} - 4664.9 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{2}{3}} + 7155.7 \tilde{\kappa}_2 \tilde{\kappa}_2}{+380.643 \tilde{\kappa}_3^{\frac{4}{3}} + 7310.7 \tilde{\kappa}_2^3 + 8386.3 \tilde{\kappa}_2^2 \tilde{\kappa}_3^{\frac{2}{3}} + 4240.5 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{4}{3}} + 507.41 \tilde{\kappa}_3^2} \right) \\ M &\simeq \left( \frac{1. + 490.94 \tilde{\kappa}_2 + 8.090 \tilde{\kappa}_3^{\frac{2}{3}} + 540.01 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{2}{3}} + 2121.3 \tilde{\kappa}_2 \tilde{\kappa}_2}{-15.803 \tilde{\kappa}_3^{\frac{4}{3}} - 704.08 \tilde{\kappa}_2^3 - 633.94 \tilde{\kappa}_2^2 \tilde{\kappa}_3^{\frac{2}{3}} - 48.92 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{4}{3}} + 53.93 \tilde{\kappa}_3^2} \right) \\ &\quad \left( \frac{0.0535 + 16.678 \tilde{\kappa}_2 - 74.221 \tilde{\kappa}_3^{\frac{2}{3}} - 3002.6 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{2}{3}} - 354.817 \tilde{\kappa}_2 \tilde{\kappa}_2}{+1593.6 \tilde{\kappa}_3^{\frac{4}{3}} + 8610.0 \tilde{\kappa}_2^3 + 3528.2 \tilde{\kappa}_2^2 \tilde{\kappa}_3^{\frac{2}{3}} + 1709.04 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{4}{3}} - 996.12 \tilde{\kappa}_3^2} \right) \\ &\quad \left( \frac{0.0691 - 15.32 \tilde{\kappa}_2 - 48.22 \tilde{\kappa}_3^{\frac{2}{3}} - 725.52 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{2}{3}} - 370.51 \tilde{\kappa}_2 \tilde{\kappa}_2}{+6.229 \tilde{\kappa}_3^{\frac{4}{3}} - 403.44 \tilde{\kappa}_2^3 + 845.4 \tilde{\kappa}_2^2 \tilde{\kappa}_3^{\frac{2}{3}} + 1322.1 \tilde{\kappa}_2 \tilde{\kappa}_3^{\frac{4}{3}} + 397.67 \tilde{\kappa}_3^2} \right)\end{aligned}$$

La validité de ces deux relations peut s'analyser à l'aide du tableau suivant :

	$M_0 = L_0 + 0.50$	$M_0 = L_0 + 1.00$	$M_0 = L_0 + 2.00$	$M_0 = L_0 + 5.00$
$L_0 = 0.50$	L = 0.49 M = 1.01	L = 0.50 M = 1.51	L = 0.50 M = 2.51	L = 0.50 M = 5.52
$L_0 = 1.00$	L = 1.01 M = 1.43	L = 1.01 M = 1.95	L = 1.00 M = 2.99	L = 1.00 M = 6.03
$L_0 = 2.00$	L = 1.98 M = 2.56	L = 2.00 M = 3.02	L = 2.01 M = 4.02	L = 2.00 M = 7.02
$L_0 = 5.00$	L = 5.15 M = 5.01	L = 5.07 M = 5.82	L = 5.00 M = 7.01	L = 5.00 M = 10.10

ce qui est tout à fait raisonnable en traitement d'images cohérentes.

## B Les codes de Sarlab dédiés aux RPM : rpclab

La plateforme Sarlab utilisée à Télécom ParisTech propose des outils pour établir des RPM de fonctions pour lesquelles on dispose d'un ensemble de données  $\{(x_n, U_n), n \in [1, N]\}$ . Ces procédures sont regroupées dans **rpclab**. Les modèles RPM peuvent alors s'obtenir pour la fonction  $x_n \rightarrow U_n$ , mais aussi pour son inverse  $U_n \rightarrow x_n$ .

Certains choix ont été effectués concernant la syntaxe des méthodes définies dans **Sarlab** :

— le modèle RPM se définit comme une liste de données caractérisant le RPM :

J, `degre_numerator`, `degre_denominateur`, `valeur_moyenne`. Cet ensemble est en effet le strict minimum pour définir et utiliser le RPM. Plutôt que de définir spécifiquement une liste Python, le choix de garder ces 4 grandeurs à travers les codes est une décision qui a semblé avoir plus d'avantages que d'inconvénients<sup>5</sup>.

5. une liste Python aurait eu des avantages, mais aussi des inconvénients.

- ces 4 grandeurs `J`, `degre_numerateur`, `degre_denominateur`, `valeur_moyenne` seront ainsi utilisées aussi bien en sortie de l'évaluation du RPM que dans l'appel des méthodes associées au RPM ainsi défini : évaluation de la fonction, méthodes donnant le RPM dans divers langages (Python C et latex).
- Une méthode d'évaluation de RPM est proposée dans un mode extrêmement simple, puisqu'elle n'exige aucun argument (hormis le tableau de valeurs en entrée et le tableau de valeurs en sortie).
- Une méthode plus générale permet de paramétrer les degrés des polynômes (numérateur, dénominateur). L'utilisateur peut enfin spécifier la valeur moyenne autour de laquelle le RPM est construit. Cet ordre est donc le même que celui utilisé pour décrire le modèle.

## B.1 Cas 1-D

Le cas monodimensionnel est le cas par défaut des calculs de RPM dans `rpclab` : c'est pour cela que, par simplification, les noms des procédures d'appel ne font pas référence à ce cas 1-D, bien qu'en interne les procédures soient souvent référencées avec le sigle 1D dans leur nom.

Les procédures 1D de `rpclab` se limitent à des RPM de degrés au plus égaux à 10 (numérateur et dénominateur). Ils requièrent tableau correspondant aux valeurs en entrée et un tableau correspondant aux valeurs attendues en sortie.

### B.1.1 Appels simplifiés

Les procédures de calcul de RPM peuvent être utilisées dans un mode simplifié :

- `calculRPC_solution` : sous sa forme d'appel la plus simple cette procédure requiert en entrée uniquement les tableaux de valeurs d'entrée et de sortie. La liste de sortie donne le vecteur  $J$ , le degré des polynômes et la valeur moyenne à utiliser dans le calcul de l'approximation.

```
J,degnum,degdenum,valmoy= rpclab.calculRPC_solution( tabENTREE, tabSORTIETHEO)
```

Les valeurs par défaut sont :

- degré du numérateur : 3
- degré du dénominateur : 3
- $\varepsilon = 0.0001$
- la valeur moyenne est égale à la valeur moyenne des entrées.

A noter que cette procédure possède des paramètres optionnels (voir le paragraphe B.1.3), ce qui permet d'éviter l'utilisation de la procédure "interne".

- `rpc_evaluation` : effectue le calcul correspondant à la relation 38 :  

```
sortie = rpclab.rpc_evaluation( entree, J, degnum, degdenum, valmoy)
```

On peut remarquer que les paramètres spécifiques à l'évaluation du RPM correspondent exactement aux paramètres de sorties de la procédure d'évaluation du RPM (`calculRPC_solution`).

- `rpc_erreur` : en utilisant les paramètres spécifiques au RPM, permet d'évaluer différentes erreurs à l'issue d'un calcul mettant en jeu des tableaux de valeurs d'entrées et des tableaux de valeurs de sorties :

```
erreurEQM, erreurmax, erreurReEQM, erreurRelmax =  
rpclab.rpc_erreur(J, degnum, degdenum, valmoy, tabENTREE, tabSORTIETHEO)
```

Les variables en sortie sont l'erreur quadratique moyenne, l'erreur maximale, l'erreur quadratique moyenne relative et l'erreur maximale relative. On peut remarquer que les paramètres spécifiques au calcul d'erreur du RPM correspondent exactement aux paramètres de sorties de la procédure d'évaluation du RPM (`calculRPC_solution`) : on y concatène simplement le jeu de données en entrée.

### B.1.2 Récupération du RPM

`rpclab` permet de récupérer le résultat RPM sous différentes modalités : expression Latex, code python, code C. Ces modalités peuvent être renvoyées par un `print` sur la console Python (on peut alors éventuellement les récupérer par un copier/coller) ou dans un fichier.

Toutes ces procédures ont en paramètres d'entrée :

- la liste de données caractérisant le RPM : `J`, `degre_numerator`, `degre_denominateur`, `valeur_moyenne`
- une chaîne de caractères, “commentaires”, qui sera placée en tête du résultat sous forme de commentaires.
- pour les écritures dans un fichier, le nom du fichier “`nom_fichier`”
- pour les codes, le nom de la procédure “`nom_procedure`”

On a donc :

- Pour récupérer le RPM en code latex, on a les procédures suivantes, dont les formulations minimales sont les suivantes :
  - `ecrire_latex_RPC` pour archiver l'expression Latex dans un fichier :  
`ecrire_latex_RPC( nom_fichier, commentaires, J, degnum, degdenum, valmoy)`  
l'extension “.tex” sera ajoutée automatiquement au nom du fichier.
  - `print_latex_RPC` pour l'afficher sur la console Python :  
`print_latex_RPC(commentaires, J, degnum, degdenum, valmoy)`
- Pour récupérer le RPM en code Python, on a les procédures suivantes, dont les formulations minimales sont les suivantes :
  - `ecrire_procedurepython_RPC` pour archiver le code Python dans un fichier :  
`ecrire_procedurepython_RPC( nom_fichier, nom_procedure, commentaires, J, degnum, degdenum, valmoy)`  
l'extension “.py” sera ajoutée automatiquement au nom du fichier.
  - `print_procedurepython_RPC` pour l'afficher sur la console Python :  
`print_procedurepython_RPC(commentaires, J, degnum, degdenum, valmoy)`
- Pour récupérer le RPM en code C, on a les procédures suivantes, dont les formulations minimales sont les suivantes :
  - `ecrire_procedureC_RPC` pour archiver le code C dans un fichier :  
`ecrire_procedureC_RPC( nom_fichier, nom_procedure, commentaires, J, degnum, degdenum, valmoy)`  
l'extension “.c” sera ajoutée automatiquement au nom du fichier.
  - `print_procedureC_RPC` pour l'afficher sur la console Python :  
`print_procedureC_RPC(commentaires, J, degnum, degdenum, valmoy)`

### B.1.3 Appels simplifiés avec paramètres optionnels

On peut utiliser des paramètres optionnels à l'appel de la procédure simplifiée `calculRPC_solution` :

- En rajoutant le degré du numérateur et celui du dénominateur :

```
J,degnum,degdenum,valmoy=
    rpclab.calculRPC_solution( tabENTREE, tabSORTIETHEO,
        degnum_entree, degdenum_entree)
```

avec  $\text{idegnum} \leq 10$  et  $\text{idegdenum} \leq 10$ .

Bien évidemment, les valeurs en sortie `idegnum` et `idegdenum` ont les mêmes valeurs qu'en entrée.

- En rajoutant le degré du numérateur et celui du dénominateur, ainsi que la valeur  $\varepsilon$  :  
`J,degnum,degdenum,valmoy=`

```
rpclab.calculRPC_solution( tabENTREE, tabSORTIETHEO,
    degnum_entree, degdenum_entree, epsilon)
```

Bien évidemment, les valeurs en sortie `idegnum` et `idegdenum` ont les mêmes valeurs qu'en entrée.

- En rajoutant le degré du numérateur et celui du dénominateur, la valeur  $\varepsilon$  ainsi que la valeur moyenne :

```
J,degnum,degdenum,valmoy=
    rpclab.calculRPC_solution( tabENTREE, tabSORTIETHEO,
        degnum_entree, degdenum_entree, epsilon, valmoy_entree)
```

Bien évidemment, les valeurs en sortie `idegnum`, `idegdenum` et `valmoy` ont les mêmes valeurs qu'en entrée.

#### B.1.4 Procédure “interne” générale

En interne, l'appel du calcul de RPM fait intervenir tous les paramètres abordés dans le descriptif du paragraphe 2, c'est à dire :

- le tableau des entrées
- le tableau des sorties théoriques correspondant aux entrées
- le degré du numérateur (au plus égal à 10)
- le degré du dénominateur (au plus égal à 10)
- une valeur moyenne correspondant au vecteur des entrées (qui n'est pas obligatoirement la moyenne des entrées)
- un paramètre permettant de choisir le type d'initialisation souhaitée :
  - $=0$  : c'est le cas classique (schéma “à la Tikhonov”, paragraphe 2.2.4) dans lequel le calcul de  $\mathbf{J}$  est effectué en schéma itératif (relation 19)
  - $=1$  : c'est le cas classique (schéma “à la Tikhonov”, paragraphe 2.2.4) dans lequel le calcul de  $\mathbf{J}$  est effectué de manière directe (relation 18).
- un paramètre permettant de choisir le type de méthode de résolution :
  - $=0$  : Cas classique “à la Tikhonov” (voir paragraphe 2.2.4), le vecteur  $\mathbf{J}$  étant initialisé à zéro.
  - $=1$  : Cas classique “à la Tikhonov” (voir paragraphe 2.2.4), le vecteur  $\mathbf{J}$  étant initialisé avec la valeur obtenue par la méthode de résolution directe.
  - $=2$  : méthode de résolution directe (voir paragraphe 2.2.2) .
- une variable  $\varepsilon$  telle que  $h^2 = \varepsilon$  (voir expression 16)

ce qui donne comme exemple de cas très général :

```
J = calculRPC_solution_tot_V1( tabENTREE, degnum, degdenum, tabSORTIETHEO,
    valmoy, taghybride, idirect, epsh)
```

Cette procédure ne retourne que le vecteur  $\mathbf{J}$ . En notant  $L$  le degré du numérateur du RPM,  $M$  le degré du dénominateur du RPM et  $\mu$  la valeur moyenne (trois grandeurs en entrée de la procédure), la sortie s'écrit (relation 3) :

$$y = \frac{J[0] + \sum_{i=1}^L J[i] (x - \mu)^i}{1. + \sum_{i=1}^M J[L+i] (x - \mu)^i} \quad (38)$$

Pour permettre une utilisation plus conviviale, des procédures permettent des appels plus simples à mettre en œuvre sont disponibles dans `rpclab`.



## B.2 Cas 2-D

Les procédures 2D de `rpclab` se limitent à des RPM de degré au plus égal à 7 (numérateur et dénominateur). Ils requièrent deux tableaux correspondant aux valeurs en entrée et deux tableaux correspondant aux valeurs attendues en sortie.

Les noms des procédures et leurs utilisations sont très similaires à celles du modèle 1-D.

### B.2.1 Appels simplifiés

- `calculRPC_solution2D` qui sous sa forme la plus simple requiert en entrée les tableaux de valeurs d'entrée et de sortie. La liste de sortie donne le vecteur  $J$ , le degré des polynômes et les valeurs moyennes.

```
J, idegnum, idegdenum, valmoy1, valmoy2 =  
    rpclab.calculRPC_solution2D( tabENTREE_1, tabENTREE_2,  
                                tabSORTIETHEO_1, tabSORTIETHEO_2)
```

Les polynômes sont de degré 3. La valeur de  $\varepsilon$  par défaut est de 0.0001

- `rpc2D_evaluation`  

```
sortie_1, sortie_2 = rpclab.rpc2D_evaluation( entree_1, entree_2,  
                                             J, degnum, degdenum, valmoy1, valmoy2)
```

### B.2.2 Récupération du RPM 2D

`rpclab` permet de récupérer le résultat RPM-2D sous différentes modalités : expression Latex, code python, code C. Ces modalités peuvent être renvoyées sur la console Python (on peut alors éventuellement les récupérer par un copier/coller) ou dans un fichier.

Toutes ces procédures ont en paramètres d'entrée :

- la liste de données caractérisant le RPM : `J`, `degre_numerateur`, `degre_denominateur`, `valeur_moyenne_1`, `valeur_moyenne_2`
- une chaîne de caractères, "commentaires", qui sera placé en tête du résultat sous forme de commentaires.
- pour les écritures dans un fichier, le nom du fichier "`nom_fichier`"
- pour les codes, le nom de la procédure

On a donc :

- Pour récupérer le RPM en code latex, on a les procédures suivantes, dont les formulations minimales sont les suivantes :
  - `ecrire_latex_RPC2D` pour archiver l'expression Latex dans un fichier :  

```
ecrire_latex_RPC2D( nom_fichier, commentaires,  
                    J, degnum, degdenum, valmoy1, valmoy2)
```

  
l'extension ".tex" sera ajoutée automatiquement au nom du fichier.
  - `print_latex_RPC` pour l'afficher sur la console Python :  

```
print_latex_RPC2D(commentaires, J, degnum, degdenum, valmoy1, valmoy2)
```
- Pour récupérer le RPM en code Python, on a les procédures suivantes, dont les formulations minimales sont les suivantes :
  - `ecrire_procedurepython_RPC2D` pour archiver le code Python dans un fichier :  

```
ecrire_procedurepython_RPC2D( nom_fichier, nom_procedure, commentaires,  
                               J, degnum, degdenum, valmoy1, valmoy2)
```

  
l'extension ".py" sera ajoutée automatiquement au nom du fichier.
  - `print_procedurepython_RPC2D` pour l'afficher sur la console Python :  

```
print_procedurepython_RPC2D(commentaires, J, degnum, degdenum, valmoy1, valmoy2)
```

- Pour récupérer le RPM en code C, on a les procédures suivantes, dont les formulations minimales sont les suivantes :
  - `ecrire_procedureC_RPC2D` pour archiver le code C dans un fichier :
 

```
ecrire_procedureC_RPC2D( nom_fichier, nom_procedure, commentaires,
                        J, degnum, degdenum, valmoy1, valmoy2)
```

 l'extension ".c" sera ajoutée automatiquement au nom du fichier.
  - `print_procedureC_RPC` pour l'afficher sur la console Python :
 

```
print_procedureC_RPC2D(commentaires, J, degnum, degdenum, valmoy1, valmoy2)
```

### B.2.3 Appels simplifiés avec paramètres optionnels

On peut utiliser des paramètres optionnels à l'appel de la procédure simplifiée `calculRPC_solution` :

- En rajoutant le degré du numérateur et celui du dénominateur :
 

```
J, idegnum, idegdenum, valmoy1, valmoy2 =
    rpclab.calculRPC_solution2D( tabENTREE_1, tabENTREE_2,
                              tabSORTIETHEO_1, tabSORTIETHEO_2, idegnum_in, idegdenum_in)
```

 Bien évidemment, les valeurs en sortie `idegnum` et `idegdenum` ont les mêmes valeurs qu'en entrée.
- En rajoutant le degré du numérateur et celui du dénominateur, ainsi que la valeur  $\varepsilon$  :
 

```
J, idegnum, idegdenum, valmoy1, valmoy2 =
    rpclab.calculRPC_solution2D( tabENTREE_1, tabENTREE_2,
                              tabSORTIETHEO_1, tabSORTIETHEO_2, idegnum_in, idegdenum_in, epsilon)
```
- En rajoutant le degré du numérateur et celui du dénominateur, la valeur  $\varepsilon$  ainsi que les valeurs moyennes :
 

```
J, idegnum, idegdenum, valmoy1, valmoy2 =
    rpclab.calculRPC_solution2D( tabENTREE_1, tabENTREE_2,
                              tabSORTIETHEO_1, tabSORTIETHEO_2, idegnum_in, idegdenum_in, epsilon,
                              valmoy1_in, valmoy2_in)
```

 Bien évidemment, les valeurs en sortie `idegnum`, `idegdenum`, `valmoy_1` et `valmoy_2` ont les mêmes valeurs qu'en entrée.

## B.3 Exemple de l'inversion et de l'approximation d'une fonction

### B.3.1 Approximation d'une fonction

Le code proposé ici est une illustration des appels en `rpclab`. La fonction à approximer est un RPM :

$$\frac{1. + 5. x - 0.222 x^3}{1.000000 + 2.70 x^2} \quad (39)$$

définie en python comme suit :

```
def fonction_base(entreep) :
    sortie= (1.+5.*entreep-0.222*entreep*entreep*entreep)/(1.+2.7*entreep*entreep)
    return (sortie)
```

La construction de la "boîte à chaussure" se fait sur 200 points entre les valeurs 0.1 et 10. :

```
NNN=200
tabENTREE=np.linspace(0.1,10.,NNN)
tabSORTIETHEO=np.zeros(NNN)
```

```
for iut in range(NNN) :
    tabSORTIETHEO[iut]= fonction_base(tabENTREE[iut])
```

L'appel pour le calcul des RPC est alors (forme simplifiée) :

```
J,degnum,degdenum,valmoy= rpclab.calculRPC_solution( tabENTREE, tabSORTIETHEO)
```

On peut alors construire un vecteur des sorties approchées (à noter que les arguments de `rpc_evaluation` sont exactement les sorties de `calculRPC_solution`) :

```
tabSORTIEAPPROX=np.zeros(NNN)
for iut in range(NNN)
    tabSORTIEAPPROX[iut]= rpclab.rpc_evaluation( tabENTREE[iut],
        J, degnum, degdenum, valmoy)
```

et le résultat est tracé figure 9.

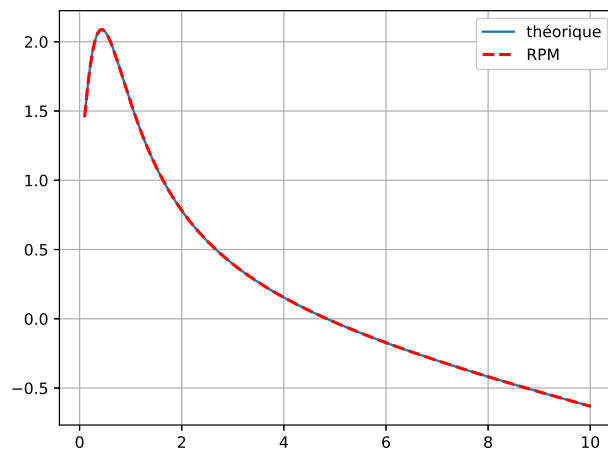


FIGURE 9 – Exemple de RPM pour approximer une fonction.

Si on veut sortir sur la console le RPM sous forme latex, on appelle la procédure `print_latex_RPC`

```
rpclab.print_latex_RPC('Un commentaire : note technique',
    Jbis,degnumA,degdenumA,valmoyA)
```

ce qui donne sur la console Python une sortie assez bavarde

```
----- LATEX (appel print_latex_RPC)-----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\iffalse
date de création : 2019-01-03
@auteur: appel ecrire_latex_RPC de la librairie sartools/rpclab Version V3.1 C2019

Un commentaire : note technique

Code Latex en notation scientifique pour avoir tous les chiffres après la virgule
```

```

Dans ecrire_label_RPC >>>> 0 0 2
1.000000 + 5.000000e+00\ : x -1.282062e-07\ : x^2 -2.220000e-01\ : x^3

1.000000 + 6.522912e-08\ : x + 2.700000e+00\ : x^2 -1.230738e-08\ : x^3

\fi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Dans ecrire_label_RPC >>>> 0 0 1
%\begin{equation}
\[
\:\: \simeq \:\: \frac{
1.000000 + 5.000000\ : x -0.000000\ : x^2 -0.222000\ : x^3
}{
1.000000 + 0.000000\ : x + 2.700000\ : x^2 -0.000000\ : x^3
}
\]
%\end{equation}
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
----- FIN LATEX -----

```

que l'on peut copier-coller dans du code latex, ce qui donne :

$$\simeq \frac{1.000000 + 5.000000 x - 0.000000 x^2 - 0.222000 x^3}{1.000000 + 0.000000 x + 2.700000 x^2 - 0.000000 x^3}$$

et on peut alors remarquer la parfaite adéquation de ce résultat avec la formule de départ (relation 39).

L'appel :

```

rpclab.ecrire_latex_RPC( 'monbeaulatex', 'Un commentaire : note technique',
    Jbis,degnumA,degdenumA,valmoyA)

```

crée un fichier monbeaulatex.tex contenant ces mêmes informations imprimées à la console.

De même :

— L'appel :

```

rpclab.ecrire_procedurepython_RPC( 'moncodepython', 'maprocedure',
    'Un commentaire : note technique',
    Jbis,degnumA,degdenumA,valmoyA)

```

crée un fichier moncodepython.py contenant le code python (appel maprocedure) concernant cette formule. Il contient aussi en commentaires la formule en Latex.

— L'appel :

```

rpclab.ecrire_procedurepython_C( 'moncodeC', 'maprocedure',
    'Un commentaire : note technique',
    Jbis,degnumA,degdenumA,valmoyA)

```

crée un fichier moncodeC.c contenant le code C (appel maprocedure) concernant cette formule. Ceci donne :

```

/* *****
# -*- coding: utf-8 -*-
"""

```

date de création : 2019-01-03  
 @auteur: appel ecrire\_procedureC\_RPC de la librairie sartools/rpclab  
 Version V3.1 C2019

Un commentaire : note technique

"""

```
***** */
float maproc(xp)
float xp;
{tabENTREE,
float x, num, denum, val;
x=xp-5.050000000e+00 ;
num=-3.350932805e-02 + -1.715605857e-01 * x
+ -4.814566839e-02 * x * x + -3.177931786e-03 * x * x * x ;
denum=1. + 3.903702880e-01 * x + 3.865052214e-02 * x * x
+ -1.761803092e-10 * x * x * x ;
val = num/denum;
return(val);
}
```

A noter que, comme pour le code python, le calcul commence par l'évaluation de l'écart à la valeur moyenne, ce qui rend difficile la comparaison avec la formule Latex.

### B.3.2 Inversion d'une fonction

La fonction à inverser est le RPM précédent (relation 39) :

$$\frac{1. + 5. x - 0.222 x^3}{1.000000 + 2.70 x^2}$$

définie en python comme précédemment :

```
def fonction_base(entreep) :
    sortie= (1.+5.*entreep-0.222*entreep*entreep*entreep)/(1.+2.7*entreep*entreep)
    return (sortie)
```

La construction de la "boîte à chaussure" se fait sur 200 points entre les valeurs 0.6 et 10. (pour avoir une fonction injective) :

```
NNN=200
tabENTREE=np.linspace(0.6,10.,NNN)
tabSORTIETHEO=np.zeros(NNN)
for iut in range(NNN) :
    tabSORTIETHEO[iut]= fonction_base(tabENTREE[iut])
```

et l'appel à la procédure se fait simplement en permutant les vecteurs :

```
JI,degnum,degdenum,valmoyI= rpclab.calculRPC_solution( tabSORTIETHEO, tabENTREE)
```

## Références

- [1] J.M. Nicolas *Application de la transformée de Mellin : étude des lois statistiques de l'imagerie cohérente* Rapport TélécomParisTech 2006D010
- [2] J.M. Nicolas *Un nouveau formalisme pour la loi de Rice* Rapport Télécom ParisTech 2018D003
- [3] C. Vincent Tao et Yong Hu *A comprehensive study of the Rational Function Model for Photogrammetric Processing*, Photogrammetric Engineering & Remote Sensing, Vol. 67, No 12, Dec 2001, pp. 1347-1357



---

**Télécom ParisTech**

Institut Mines-Télécom - membre de l'Université Paris Saclay

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - [www.telecom-paristech.fr](http://www.telecom-paristech.fr)

**Département IDS**