ROYAUME DU MAROC
Université Mohammed V - RABAT -
جامعة محمد الخامس – الرباط –
Faculté des sciences
كلية العلوم
CENTRE D'ETUDES DOCTORALES - SCIENCES ET TECHNOLOGIES

THESE

N° d'ordre : 3696

En vue de l'obtention du : **DOCTORAT**

Centre de Recherche : CeReMAR
Structure de Recherche : Laboratoire Mathématiques Informatique et Applications - Sécurité de l'information (LabMiA-SI)
Discipline : Mathématiques
Spécialité : Analyse Numérique

Présentée et soutenue le : 28/10/2022 par :

Abdelhak ESSANHAJI

Contribution à l'interpolation polynomiale de Lagrange à plusieurs variables

JURY

Souad EL BERNOUSSI	PES, Université Mohammed V, Faculté des sciences, Rabat	Présidente
Rachid SADAKA	PES, Université Mohammed V, Ecole Normale Supérieure, Rabat	Rapporteur / Examinateur
Brahim BENOUAHMANE	PES, Université Hassan 2, Faculté des Sciences et Techniques de Mohammedia, Casablanca	Rapporteur / Examinateur
Abdellah EL KINANI	PES, Université Mohammed V, Ecole Normale Supérieure, Rabat	Rapporteur / Examinateur
Jilali ABOUIR	PES, Université Hassan 2, Faculté des Sciences et Techniques de Mohammedia, Casablanca	Examinateur
Mohammed ERRACHID	PH, Centre Régional des Métiers de l'Education et de la Formation, Rabat	Co-directeur de thèse
Abderrahim MESSAOUDI	PES, Université Mohammed V, Ecole Normale Supérieure, Rabat	Directeur de thèse

Année Universitaire : 2022/2023

Dédicace

À la mémoire de mon frère Mohammed

qui nous a quitté le 27 Mars 2014

À mes chers parents

À ma chère femme

À mes chères filles Marwa et Israe

À toute ma famille

Remerciements

Ce travail a été effectué au laboratoire Mathématiques, Informatique et Applications - Sécurité de l'information (LabMiA-SI), de la Faculté des Sciences de Rabat, sous la direction de madame **Souad EL BERNOUSSI**, professeur d'enseignement supérieur à la Faculté des Sciences de Rabat.

Je souhaite remercier mon directeur de thèse, monsieur **Abderrahim MESSAOUDI** et mon co-directeur, monsieur **Mohammed ERRACHID** pour la confiance qu'ils m'ont accordée et pour leur investissement dans la réalisation de ce travail.

J'adresse tout d'abord toute ma gratitude à monsieur **Abderrahim MESSAOUDI**, professeur d'enseignement supérieur à l'Ecole Normale Supérieure de Rabat, pour toutes les idées sans lesquelles mes travaux n'auraient pu aboutir, pour les relectures méticuleuses de ce manuscrit et les corrections proposées. Il m'a fait profiter de sa grande culture mathématique et j'ai beaucoup appris à ses côtés. Pour tout cela, je lui exprime mes plus vifs remerciements.

Je témoigne ma profonde reconnaissance à monsieur **Mohammed ERRACHID**, professeur habilité au Centre Régional des Métiers de l'Education et de la Formation, CRMEF-Rabat, pour avoir guidé mes premiers pas dans le monde de la recherche et prodigué de précieux conseils qui m'ont fait progresser. Je le remercie sincèrement pour sa disponibilité, sa patience et pour m'avoir constamment rassuré et encouragé.

J'aimerais aussi adresser ma gratitude à madame **Souad EL BERNOUSSI**, professeur d'enseignement supérieur à la Faculté des Sciences de Rabat, pour avoir honoré cette soutenance de thèse par la présidence de son jury, ainsi que pour son assistance administrative en tant que responsable du laboratoire de Mathématiques, Informatique et Applications - Sécurité de l'Information.

Je tiens à remercier monsieur **Rachid SADAKA**, professeur d'enseignement supérieur à l'Ecole Normale Supérieure de Rabat, pour l'intérêt qu'il a manifesté à l'égard de cette recherche en s'engageant à être rapporteur et examinateur.

Mes remerciements vont aussi à monsieur **Brahim BENOUAHMANE**, professeur d'enseignement supérieur à la Faculté des Sciences et Techniques de Mohammedia, pour avoir accepté d'être rapporteur et examinateur, et consacré son temps et effort pour juger ce travail.

Mes remerciements vont aussi à monsieur **Abdellah KINANI**, professeur d'enseignement supérieur à l'Ecole Normale Supérieure de Rabat, pour avoir accepté d'être rapporteur et examinateur, et consacré son temps et effort pour juger ce travail.

Je voudrais remercier monsieur **Jilali ABOUIR**, à la Faculté des Sciences et Techniques de Mohammedia, d'avoir consacré son temps à examiner ce mémoire de thèse et d'avoir accepté de faire partie du jury.

Mes remerciement vont également à mes collègues du centre de préparation de l'agrégation de Mathématiques du Centre Régional des Métiers de l'Education et de la Formation Rabat-Salé-Kénitra.

Enfin, j'adresse un grand merci à mes parents, pour leur soutien et leur amour, à mon épouse, pour son aide, son soutien et ses encouragements, mes chères filles, mes frères, mes amis et tous ceux qui m'ont soutenu durant l'élaboration de ce travail.

Qu'ils en soient tous remerciés, ici.

Résumé

Les problèmes de l'interpolation polynomiale uni-variée de Lagrange ou d'Hermite ont été traités par plusieurs recherches récentes. L'étude de l'interpolation polynomiale à plusieurs variables est plus difficile et les approches sont moins évidentes. Dans ce travail, nous proposons des approches simples et efficaces pour étudier ce problème sur une configuration finie $Z \subset \mathbb{K}^p$ quelconque. En effet, nous proposons un algorithme appelé RMVPIA, qui détermine un polynôme d'interpolation dans le cas particulier où l'ensemble d'interpolation Z est une grille, puis nous donnons une généralisation de cet algorithme, appelé RMVPIA avec ordre. Ensuite, nous proposons un algorithme randomisé RLMVPIA qui traite le cas général et permet de construire plusieurs espaces d'interpolations basées sur un concept aléatoire. OLMVPIA est un autre algorithme, qui nous permet de calculer un espace d'interpolation optimal associé à l'ensemble d'interpolation Z.

Mots-clés : Complément de Schur, Interpolation polynomiale de Lagrange à plusieurs variables, algorithmes récursifs, approche aléatoire d'interpolation de Lagrange, espace optimal d'interpolation.

Abstract

The problems of one-variate Lagrange or Hermite polynomial interpolation have been addressed by several recent researches. The study of multivariate polynomial interpolation is more difficult and the approaches are less obvious. In this work, we propose simple and efficient approaches to study this problem on any finite $Z \subset \mathbb{K}^p$ configuration. Indeed, we propose an algorithm called RMVPIA, which determines an interpolation polynomial in the particular case where the interpolation set Z is a grid, and then we give a generalization of this algorithm, called RMVPIA with order. Then, we propose a randomized algorithm RLMVPIA that deals with the general case and allows to construct several interpolation spaces based on a random concept. OLMVPIA is another algorithm, which allows us to compute an optimal interpolation space associated with the interpolation set Z.

Keywords : Schur complement, Lagrange Multivariate Polynomial Interpolation Problem, Recursive Algorithms, Random approach of Lagrange's interpolation, Optimal space of interpolation.

Publications

— M. Errachid, **A. Essanhaji**, and A. Messaoudi, *RMVPIA : a new algorithm for computing the Lagrange multivariate polynomial interpolation*, Numerical Algorithms Journal, vol. 84, no. 4, pp. 1507-1534, 2020.

— **A. Essanhaji**, M. Errachid, *Lagrange Multivariate Polynomial Interpolation : A Random Algorithmic Approach*, Journal of Applied Mathematics, vol. 2022, Article ID 8227086, 8 pages, 2022. https ://doi.org/10.1155/2022/8227086

— **A. Essanhaji**, M. Errachid, OLMVPIA, *Optimal recursive algorithm for computing the Lagrange multivariate polynomial interpolation*, (submitted)

Communications

Ci-après, les communications orales que j'ai prononcées, dans différentes manifestations scientifiques organisées par le labMia-SI de la Faculté des sciences de Rabat, l'école normale supérieure de Rabat et la Faculté des sciences et techniques de Fés.

02-04 Avril 2019 : M. Errachid, A. Essanhaji, "A New Algorithm for Computing Multi-Variate Polynomial Interpolation", The International Conference On Mathematic Modeling with Applications. Université Mohammed V, FSR.

30 Mai 2019 : A. Essanhaji, M. Errachid, "Nouvel Algorithme pour calculer le polynôme d'interpolation à plusieurs variables", Séminaire du LabMia-Si de la Faculté des sciences de Rabat.

21-22 Juin 2019 : A. Essanhaji, M. Errachid, "New Algorithms for Computing Interpolation polynomials", Rencontre National du Laboratoire AAFA de la FST de Fès.

24-25 Décembre 2019 : A. Essanhaji, M. Errachid, " L'interpolation de Lagrange à plusieurs variables", Journées Doctorales de l'Ecole Normale Supérieure de Rabat.

03-04 Décembre 2021 : A. Essanhaji, M. Errachid, " L'interpolation de Lagrange à plusieurs variables, une nouvelle approche", Première rencontre Nationale du Laboratoire : "Modélisation et Structures Mathématiques" Faculté des sciences et Techniques Fès.

13 Janvier 2022 : A. Essanhaji, M. Errachid, Optimal Lagrange Multivariate Polynomial Interpolation Algorithm, l'Ecole Normale Supérieure de Rabat.

02 Juin 2022 : A. Essanhaji, M. Errachid, "Multivariate polynomial interpolation : new approaches", Séminaire du LabMia-Si de la Faculté des sciences de Rabat.

Activités Pédagogiques

Depuis Septembre 2015 : Formateur au cycle de préparation à l'agrégation au CRMEF de Rabat.

15-20 Février 2020 : Participation à la préparation des élèves MP* aux concours des grandes écoles d'ingénieurs en France.

18-28 Septembre 2020 : Membre du Jury de l'équipe nationale qui a participé à la compétition internationale de l'olympiade des mathématiques, IMO 2020, en Saint Petersburg - Russie.

8-10 Juillet 2021 : Formation à l'utilisation du Python scientifique, aux doctorants de la faculté des sciences de Rabat.

Avril-Juin, 2022 : Responsable du cours : initiation au Python scientifique, Master Fimath à la faculté des sciences de Rabat.

22-28 Juin 2022 : Membre du Jury de la compétition : Olympiades Pan Africaines de Mathématiques OPAM 2022, UM6P Benguerir - Maroc.

Table des figures

Table des matières

Introduction Générale

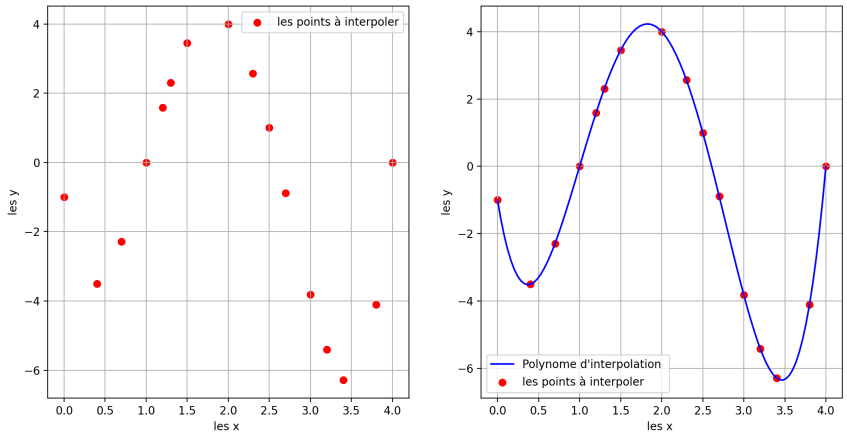

FIGURE 1 – Problème d'interpolation.

Étant donnés un corps $\mathbb{K}$ commutatif et n couples $(z_1, r_1), \ldots, (z_n, r_n)$ où $z_1, ..., z_n$ sont des points de $\mathbb{K}^p$ $(p \in \mathbb{N}^*)$ et $r_1, ..., r_n$ sont des scalaires dans $\mathbb{K}$, l'interpolation est le problème de la construction d'une fonction P appartenant à un espace de fonctions simples de dimension finie dont le graphe passe par les points (z_i, r_i), figure 1.

$r_1, ..., r_n$ sont dits les valeurs de l'interpolation, les points $z_1, ..., z_n$ sont appelés les noeuds de l'interpolation.

Les quantités r_i peuvent, par exemple, représenter les valeurs aux noeuds z_i d'une fonction f connue analytiquement(plus difficile) ou des valeurs relevées expérimentalement en fonction des variables $z_1, ..., z_n$. Dans le premier cas, on dit que P interpole f, dans le sens où les deux fonctions coïncident sur cet ensemble de données et l'approximation a pour but de remplacer f par la fonction plus simple P en vue par exemple, d'un calcul numérique d'intégrale ou de dérivée... Dans le second cas, le but est d'avoir une représentation synthétique de données expérimentales, afin de faire des prévisions par exemple.

Le contexte le plus simple à étudier ici est l'interpolation par des polynômes à une variables $(p = 1)$. Il n'est donc pas surprenant que l'interpolation par des polynômes à une variable soit un sujet très classique.

On parle d'interpolation polynomiale (de Lagrange) quand P est un polynôme, d'approximation trigonométrique quand P est un polynôme trigonométrique et d'interpolation polynomiale par morceaux (ou d'interpolation par fonctions splines) si P est polynomiale par morceaux.

La théorie de l'interpolation polynomiale comprend alors l'étude de la recherche des conditions d'existence et d'unicité des solutions du problème posé mais aussi celle de la recherche d'algorithmes qui permettent de déterminer des solutions éventuelles et d'en étudier les propriétés et les performances.

Pourquoi s'intéresse t-on à l'interpolation polynomiale ? Et bien, c'est tout simplement parce que les polynômes sont les fonctions les plus faciles à manipuler que ça soit au niveau des opérations arithmétiques comme la somme, le produit, les puissances ou au niveau des opérateurs analytiques comme l'intégration et la dérivation...

De plus, l'interpolation polynomiale s'applique dans plusieurs domaines des mathématques appliquées notamment dans les domaines de l'approximation, du calcul intégral, des équations différentielles, de la géométrie, des éléments finis, des statistiques, de la cryptographie etc.

Dans une étude historique[16] publiée en **1994** dans son manuel intitulé "Historical Perspective on Interpolation, Approximation and Quadrature", Claude Brezinski parcourt la grande histoire de la théorie de l'interpolation qui a débuté grâce au mathématicien perse Afzal Muhammed Al-Biruni (**973-1048**) et reprise au $17^{ème}$ siécle par les mathématiciens occidentaux Thomas-Harriot (**1560-1621**), John Wallis (**1616-1703**) et Henry Briggs (**1561-1630**) qui a construit la table du logarithme. Mais l'interpolation polynomiale a connu son grand essor grâce notamment à cette liste non exhaustive de grands mathématiciens

— Issac Newton [68, 69] : Mathématicien, physicien et philosophe anglais puis britannique né le **25-12-1642** et décédé le **20-03-1727**. Figure emblématique des

sciences, il est surtout reconnu pour avoir fondé la mécanique classique, pour sa théorie de la gravitation universelle et la création, en concurrence avec Gottfried Wilhelm Leibniz, du calcul infinitésimal. En mathématiques il est connu pour la généralisation du théorème du binôme et l'invention dite de la méthode de Newton permettant de trouver des approximations d'un zéro (ou d'une racine) d'une fonction d'une variable réelle à valeurs réelles.

— Roger Cotes[21] : Mathématicien anglais né le **10-07-1682** et décédé le **05-06-1716**. C'est un proche de Newton avec qui il partage la découverte de la méthode de Newton-Cotes en analyse numérique, qui étend de manière générale la méthode des trapèzes et la méthode de Simpson pour le calcul des intégrales.

— Joseph-Louis Lagrange [50] : Mathématicien Français né à Tourain le **25-01-1736** et décédé le **10-04-1813** à Paris. Il est l'un des 72 savants dont le nom est inscrit sur le premier étage de la tour Eiffel. Bien que le concept de l'interpolation soit connu avant lui, mais c'est Lagrange qui montra l'intérêt de cette théorie qui portera d'ailleurs son nom et c'est lui qui inventa la base d'interpolation de Lagrange, dans laquelle s'exprime très simplement un polynôme d'interpolation de Lagrange.

— Pafnuty Chebyshev[86] : Mathématicien russe né à Okatovo le **16-05-1821** et décédé le **08-12-1894**. Il a enseigné à l'université de St Petersburg. Il a contribué en particulier dans les théories des nombres, de la géomètrie, des formes quadratiques, des probabilités, des approximations et celle des intégrales.

— Charles Hermite[43, 44] : Mathématicien Français né à Dieuze (en Lorraine) le **24-12-1822** avec une déformation au pied droit et décédé le **14-01-1901**. On lui doit notamment de grands progrés en théorie des nombres, des fonctions doublement périodiques, la démonstration de la transcendance du nombre e en 1872 et l'interpolation polynomiale qui porte son nom et qui généralise celle de Lagrange.

— Carl David Tolmé Runge [77] : Mathématicien et physicien allemand né le **30-08-1856** et décédé le **03-01-1927**, est le premier professeur de maths appliquées

de l'histoire. Élève de Weierstrass ayant donc une solide formation en mathématiques pures, il fût le premier à montrer en 1901 que l'interpolation de Lagrange peut diverger même avec des fonctions très régulières. C'est ce qu'on appelle le phénomène de Runge.

— Georg Faber [33] : Mathématicien allemand né le **05-04-1877** à Kaiserslautern et décédé le **07-03-1966** à Munich, spécialiste de l'analyse complexe. Il démontra en 1914 que les polynômes d'interpolation de Lagrange ne convergent pas uniformément.

— Lipót Fejér [34, 35] : Mathématicien hongrois né le **09-02-1880** à Pécs et mort le **15-10-1959** à Budapest. C'est l'un des mathématiciens qui ont démontré le célébre théorème de Weierstrass et il a publié un théorème de convergence remarquable pour les séries de Fourier.

— George David Birkhoff[2, 3] : Mathématicien de nationalité Américaine, né le **21-03-1884** et décédé le **12-11-1944**. Diplômé de Harvard en 1905, il s'est intéressé en particulier aux équations différentielles et les systèmes dynamiques. En 1906, il publia un article sur l'interpolation polynomiale généralisant les travaux de Lagrange et d'Hermite.

— Eric Harold Neville[67] : Mathématicien de nationalité britannique né le **01-01-1889** et décédé le **22-08-1961**. Les principaux domaines d'expertise de Neville étaient la géométrie différentielle, la géométrie analytique et complexe. Mais l'algorithme de l'interpolation polynomiale qui porte son nom et qui est très proche de celui d'Aitken publié en 1932 reste le travail qui fait jusqu'à aujourd'hui sa célébrité.

— Alexander Craig Aitken[1] : Un des mathématiciens les plus éminents de la Nouvelle-Zélande, né le **01-04-1895** et décédé le **03-04-1967**. Dans un article de 1935, il a introduit le concept des moindres carrés généralisés. Il conçoit un algorithme qui permet de déterminer le polynôme d'interpolation de Lagrange exprimé dans une base de Newton en utilisant la méthode de la différence divi-

sée.

— Pàl. Turàn[28, 88, 89, 53] : Mathématicien hongrois né le 18-08-1910 à Budapest et décédé le 26-09-1976, connu pour ses travaux en théorie des graphes. Il est peut être l'un des plus grands mathématiciens contemporains qui ont oeuvré au développement de l'interpolation, en posant en **1974** , 15 questions ouvertes relatives à l'interpolation d'Hermite-Birkhoff à une variable.

— Paul Erdõs[28, 29] : Mathématicien hongrois né le **26-03-1913** à Budapest et mort le **20-09-1996** à Varsovie. C'est un spécialiste des probabilités et de l'analyse combinatoire. Mais il a aussi contribué à la publication de plusieurs résultats concernant l'interpolation, en particulier le théorème de la convergence d'Erdõs et Turàn en **1937**.

Si l'interpolation, par polynômes ou autres fonctions, est une méthode assez ancienne en mathématiques appliquées, apparemment, le mot "interpolation" lui-même a été introduit par **J. Wallis** dès **1655** comme il est affirmé dans [27], par rapport à cela, l'interpolation polynomiale à plusieurs variables est un sujet relativement nouveau et n'a probablement commencé que dans la seconde moitié du 19ème siècle [4, 49]. Et son étude est beaucoup plus complexe et il s'agit d'un sujet de recherche actif depuis plus de 50 ans.

L'interpolation bi-variée par le produit tensoriel de fonctions d'interpolation univariées, c'est-à-dire lorsque les variables sont traitées séparément, est l'approche classique de l'interpolation multivariée. Cependant, lorsque l'ensemble des points d'interpolation n'est pas une grille, il est impossible d'utiliser cette idée. Aujourd'hui, étant donné n'importe quel ensemble de points d'interpolation, il existe de nombreuses méthodes [16] pour construire un espace polynomial adéquat qui garantit l'unisolvabilité du problème d'interpolation. Étonnamment, cette idée de construire un espace d'interpolation approprié a déjà été poursuivie par **Kronecker** [49] dans un article largement inconnu de **1865**, qui semble être le premier traitement de l'interpolation polynomiale

multivariée par rapport à un ensemble de points assez arbitraires.

On peut observer que jusqu'au début du siècle dernier il y a peu d'articles de recherche qui traitent l'interpolation polynomiale multivariée. Dans le livre classique *Interpolation* [85], où une section (Section 19) est consacrée à ce sujet, l'auteur ne fait référence qu'à deux articles connexes, récents à cette époque (1927), à savoir [64, 65]. Le dernier [65] s'est avéré malheureusement inaccessible pour nous, mais il n'est pas difficile de deviner qu'il aurait pu poursuivre une approche de produit tensoriel, car c'est le point de vue unique de [85] (voir aussi [74]).

Les formules données dans [64] sont des formules de Newton pour l'interpolation par produit tensoriel en deux variables, et l'auteur, Narumi, affirme qu'elles peuvent être étendues à plusieurs variables. Puisqu'il s'agit d'une approche par produit tensoriel, les points d'interpolation sont de la forme $(x_i, y_j), 0 \leqslant i \leqslant m, \quad 0 \leqslant j \leqslant n$, répartis arbitrairement sur les axes OX et OY, respectivement. Les différences divisées bivariées pour ces ensembles de points sont obtenues dans [64], par récurrence, séparément pour chaque variable. Avec les notations habituelles, la formule d'interpolation de [64] se lit comme suit

$$p(x,y) = \sum_{i=0}^{m} \sum_{j=0}^{n} f\left[x_0, \ldots, x_i; y_0, \ldots, y_j\right] \prod_{h=0}^{i-1} (x - x_h) \prod_{k=0}^{j-1} (y - y_k),$$

où les produits vides ont la valeur 1.

Dans les années 1950, un changement important de paradigme s'est produit dans le domaine de l'interpolation polynomiale multivariée, car plusieurs personnes ont commencé à étudier des distributions plus générales de points, et pas seulement des sous-ensembles (spéciaux) de produits cartésiens. Et grâce à l'évolution du calcul scientifique, l'intérêt de l'interpolation polynomiale n'a cessé d'augmenter sucitant l'implication de plusieurs chercheurs en mathématiques appliquées contemporaines à travers la publication de plusieurs recherches dans le domaine tant pour élargir le spectre des

applications de l'interpolation que pour étudier les effets et les qualités numériques de ces interpolations. À ce niveau on pourra citer les travaux de George.G. Lorentz (depuis **1966**) [52, 53, 25], Samuel. Karlin (**1924-2007**)[47], Gunter. Mühlbach (depuis **1973**)[61, 62, 63], J.I Maeztu (depuis **1979**)[36], A. Lopez-Carmona (depuis **1980**)[37], Mariano Gasca (depuis **1980**)[19, 36, 38, 39], Rudolph A. Lorentz (depuis **1980**)[54, 55], Ronald.A. DeVore (depuis**1992**)[25], Thomas. Sauer (depuis **1980**)[78, 79, 39], le mathématicien chinois Ying Guang Shi (depuis **1993**)[83] qui a répondu en **2003** aux questions posées par Turàn, F. Palacios (depuis **2003**)[72] et P. Rubiò (depuis **2003**)[76].

Dans un article[15] publié en **1983**, Claude. Brezinski inventera des méthodes récursives pour résoudre des problèmes de projections, d'interpolation et d'extrapolation et en **1988** il introduit [13] la notion du complément de Schur pour obtenir des méthodes récursives pour résoudre des problèmes d'accélération de la convergence, des approximants de Padé et des fractions continues. Depuis **1995**, des chercheurs Marocains comme Abderrahim. Messaoudi, Khalide. Jbilou, Rachid Sadaka et Hassane. Sadok et bien d'autres ont utilisé cette approche pour découvrir de nouveaux algorithmes performants permettant de résoudre des systèmes d'équations linéaires [46, 56, 57] et de les appliquer pour résoudre des problèmes d'interpolation polynomiale de Lagrange et d'Hermite à une variable[60, 58, 59]. Dans le chapitre 2 on propose une extension de ces méthodes à la résolution des problèmes de l'interpolation de Lagrange à plusieurs variables dans le cas d'une grille.

Étant donné les différents avantages de la base de Newton dans l'interpolation polynomiale à une variable et l'efficacité des différences divisées, plusieurs chercheurs se sont efforcés d'adapter l'algorithme au cas de plusieurs variables, et de trouver des conditions sur l'ensemble des noeuds d'interpolation pour utiliser la base de Newton [45, 66, 70, 91]. Dans le récent article [66], l'auteur a proposé une généralisation de l'algorithme uni-varié de la forme de base de Newton et de l'algorithme de la différence divisée dans $\mathbb{K}^p$. Les ensembles d'interpolation requis sont ceux qui admettent

une indexation ayant une structure régulière comme triangulaire, rectangulaire, ou plus généralement un ensemble inférieur (lower set) [7, 70, 40, 66, 81]. Il montre comment la structure de l'ensemble d'indices détermine de manière appropriée l'espace d'interpolation. Lorsque la somme des indices est bornée par d, il existe une interpolation unique avec un polynôme de degré $\leq d$.

Dans la présente thèse, nous nous intéressons dans un premier temps au cas où l'ensemble de l'interpolation est une grille. Nous présentons un nouvel algorithme nommé RMVPIA Recursive Multi-Varite Polynomial Interpolation Algorithm, qui a fait l'objet d'une publication en 2020.

Pour résoudre le problème de l'interpolation de Lagrange associé à un ensemble d'interpolation fini quelconque, nous avons construit, en ordonnant les noeuds, un algorithme nommé RLMVPIA Recursive Lagrange Multi-Varite Polynomial Interpolation Algorithm, qui calcule un polynôme interpolant répondant au problème.

Une analyse de cet algorithme permet de se rendre compte que le polynôme interpolant obtenu peut admettre, dans certaines situations, un degré total très élevé.

Pour remédier à ce problème, nous avons conçu un autre algorithme nommé RLMVPIA aléatoire publié en 2022. Cet algorithme remplace l'ordre des noeuds par un nouveau concept dit (Z, z)-partition, qui donne plus de liberté dans le choix des nouveaux noeuds à interpoler. RLMVPIA aléatoire nous permet, en fait, de construire plusieurs solutions dont les expressions peuvent différer par le degré total ou par le nombre de termes.

Afin de construire un espace d'interpolation, dans lequel le problème considéré admet une et une seule solution de degré optimal, nous avons conçu un algorithme nommé OLMVPIA Optimal Lagrange Multi-Varite Polynomial Interpolation Algorithm, qui repose sur l'idée de classer les noeuds selon un concept de poids des cordonnées comme c'est expliqué dans le chapitre 4.

Chapitre -1

Préliminaires

L'objectif de ce chapitre est de rappeler l'essentiel des notions et résultats utilisés tout au long de ce travail. On commence dans la section 1 par décrire le problème d'interpolation de Lagrange à plusieurs variables, à définir la notion du complément de Schur et d'en étudier les propriétés, ensuite on donne les ingrédients théoriques de l'analyse des algorithmes à savoir, la notion de complexité, terminaison et correction. La section 2, est consacrée à rappeler les différents algorithmes utilisés dans l'interpolation de Lagrange à une seule variable, qui seront utilisés dans l'algorithme OLMVPIA traité au chapitre 4.

1.1 Généralités

1.1.1 Le Problème d'interpolation de Lagrange

Dans ce paragraphe on présente les différentes notations et définitions liées au problème d'interpolation de Lagrange qui seront utilisées tout au long de cette thèse.

Soient p et d deux entiers naturels, $p \neq 0$ et $\mathbb{K}$ un corps commutatif. L'espace des polynômes à p variables est noté $\Pi^p = \mathbb{K}[x_1, \ldots, x_p]$, et on note Π_d^p le sous-espace des polynômes de degré total inférieur ou égal à d.

On se donne un ensemble d'interpolation fini $Z = \{z_1, \ldots, z_n\}$ à n noeuds distincts de $\mathbb{K}^p$, le problème de l'interpolation polynomiale de Lagrange consiste à chercher, pour un ensemble de valeurs $R = (r_z : z \in Z) \in \mathbb{K}^Z$, une fonction polynomiale $P \in \Pi^p$ vérifiant

$$P(Z) = R, \quad \text{i.e} \quad P(z) = r_z, \quad z \in Z, \tag{1.1.1}$$

On dit alors que P est un polynôme interpolant R en Z. Plus précisément, on dit que Z est bien-posé, correct ou unisolvant [20, 40, 80] dans le sous espace $\mathcal{P}$ de Π^p, si pour tout ensemble de valeurs $R = (r_z : z \in Z) \in \mathbb{K}^Z$, le problème d'interpolation (1.1.1) admet une unique solution dans $\mathcal{P}$, autrement dit l'application

$$P \in \mathcal{P} \mapsto (P(z) : z \in Z) \in \mathbb{K}^Z,$$

est un isomorphisme. Il s'ensuit que $\dim \mathcal{P} = n$. Ce problème sera noté par la suite par $\mathcal{P}(Z)$.

Il est bien connu que dans le cas univarié ($p = 1$) (voir la sous-section 3) le problème d'interpolation de Lagrange par rapport à n points distincts est correct, si on prend $\mathcal{P}$ l'espace des polynômes de degré inférieur ou égal à $n - 1$. Cependant la situation est beaucoup plus difficile dans le cas d'interpolation à plusieurs variables. Afin d'interpoler

avec succès Z dans Π_d^p, nous devons avoir

$$n = \begin{pmatrix} p+d \\ p \end{pmatrix},$$

puisque

$$\dim \Pi_d^p = \begin{pmatrix} p+d \\ p \end{pmatrix}.$$

Et même si c'est le cas, il peut y avoir le problème que les points se trouvent sur une surface algébrique de degré d, c'est-à-dire qu'il existe un certain polynôme Q de degré total au plus d qui est nul sur Z. Par exemple, si on prend $p = 2$, $d = 1$ et $Z = \{(-1, 0), (0, 0), (1, 0)\}$, il est facile de voir que l'ensemble Z n'est pas correct dans l'espace $\Pi_1^2 = vect\{1, x, y\}$ (puisque les éléments de Z sont des zéros de $P = y$). Ainsi, l'unisolvance du problème d'interpolation polynomiale à plusieurs variables dépend de la structure géométrique de l'ensemble d'interpolation Z.

1.1.2 Le complément de Schur

Dans cette section, on rappelle les définitions et propriétés du complément de Schur [71, 13].

Définition 1.1.1 *Soit M une matrice définie par blocs $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$. Si A une matrice inversible, on définit le* ***complément de Schur*** *de A dans M noté (M/A) par*

$$(M/A) = D - CA^{-1}B$$

De manière plus générale, si $M = [M_{i,j}]$ est une matrice définie par blocs telle que l'un de ses blocs $M_{i,j}$ est inversible, alors le complément de Schur du bloc $M_{i,j}$ dans M noté $(M/M_{i,j})$ est défini par

$$(M/M_{i,j}) := \widetilde{M}_{i,j} - M_{i^*,j} M_{i,j}^{-1} M_{i,j^*},$$

où, $\widetilde{M}_{i,j}$ est la matrice obtenue à partir de M en y supprimant tous les blocs de la ligne i et ceux de la colonne j, $M_{i^*,j}$ est la matrice formée des blocs de la colonne j de M sans l'élément $M_{i,j}$ et M_{i,j^*} est la matrice formée des blocs de la ligne i de M sans l'élément $M_{i,j}$.

On obtient aisément les premières propriétés générales suivantes.

Propriétés 1.1.1 *Soit $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ une matrice définie par blocs.*

1. *Supposons D inversible alors*

$$\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}/D\right) = \left(\begin{bmatrix} D & C \\ B & A \end{bmatrix}/D\right) \tag{1.1.2}$$
$$= \left(\begin{bmatrix} B & A \\ D & C \end{bmatrix}/D\right) = \left(\begin{bmatrix} C & D \\ A & B \end{bmatrix}/D\right).$$

2. *Si D inversible et E une matrice telle que EA est bien définie alors*

$$\left(\begin{bmatrix} EA & EB \\ C & D \end{bmatrix}/D\right) = E\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}/D\right). \tag{1.1.3}$$

3. *De même si AE est bien définie alors*

$$\left(\begin{bmatrix} AE & B \\ CE & D \end{bmatrix}/D\right) = \left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}/D\right)E\ . \tag{1.1.4}$$

et lorsque M est de plus carrée on obtient les propriétés supplémentaires intéressantes suivantes faciles aussi à vérifier.

Théorème 1.1.1 *Soit $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ supposée carrée et telle que A est inversible. Alors*

1. *(M/A) est carrée et est de même taille que D et on a*

$$\det(M/A) = \frac{\det M}{\det A}. \tag{1.1.5}$$

2. *Si D est un scalaire alors (M/A) est un scalair égal à* $\dfrac{\det M}{\det A}$.

3. *M est inversible si, et seulement si, (M/A) est inversible et dans ce cas on a*

$$M^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B\,(M/A)^{-1}\,CA^{-1} & -A^{-1}B\,(M/A)^{-1} \\ -\,(M/A)^{-1}\,CA^{-1} & (M/A)^{-1} \end{bmatrix}. \tag{1.1.6}$$

Enfin, dans le théorème suivant on rappelle la fameuse identité matricielle de Sylvester [71] qui permet de ramener le calcul du complément de Schur d'une matrice découpée en plusieurs blocs à celui de compléments de Schur de matrices à quatre blocs.

Théorème 1.1.2 *Si K est une matrice définie par blocs de la forme*

$$K = \begin{bmatrix} E & F & G \\ H & A & B \\ L & C & D \end{bmatrix}.$$

et si on pose $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ *et qu'on suppose A et M inversibles alors on a l'**Identité matricielle de Sylvester** suivante*

$$\begin{aligned} (K/M) &= ((K/A)\,/\,(M/A)) \\ &= \left(\begin{bmatrix} E & F \\ H & A \end{bmatrix}/A\right) - \left(\begin{bmatrix} F & G \\ A & B \end{bmatrix}/A\right)(M/A)^{-1}\left(\begin{bmatrix} H & A \\ L & C \end{bmatrix}/A\right). \end{aligned} \tag{1.1.7}$$

1.1.3 Analyse d'un algorithme

Le but de cette sous-section est de définir quelques notions utiles pour analyser un algorithme. On va donner les outils permettant de répondre aux trois questions suivantes :

— l'algorithme s'arrête-t-il un jour ?

— est-ce qu'il fait bien ce qu'il est sensé faire ? Autrement dit, est-il correct ?

— combien de temps met-il à s'exécuter ?

Le premier point s'appelle la terminaison de l'algorithme, le deuxième sa correction et le dernier sa complexité.

Terminaison

Pour montrer qu'un algorithme termine quel que soit le jeu de paramètres passé en entrée respectant la spécification, il faut montrer que chaque bloc élémentaire constituant le corps de l'algorithme termine ! Or, les boucles for et les instructions conditionnelles terminent forcément. Le seul souci pourrait venir d'une boucle *while*. En général, pour montrer la terminaison d'une boucle on procède ainsi : on exhibe une quantité, dépendant des paramètres, à valeurs dans $\mathbb{N}$, qui décroît strictement à chaque passage dans la boucle. Puisqu'il n'existe pas de suite infinie strictement décroissante dans N, cela prouve que la boucle se termine ! Cette quantité porte un nom en informatique :

Définition 1.1.2 *Un variant de boucle est une quantité positive, à valeurs dans* $\mathbb{N}$*, dépendant des variables de la boucle, qui décroît strictement à chaque passage dans la boucle.*

Exemple Soient a et b des entiers tels que $a \geq 0$ et $b > 0$. On considère le programme suivant, dont l'objectif et de renvoyer le reste et le quotient de la division euclidienne de a par b.

```
14 def f(a, b):
15     q, r = 0, a
16     while r >= b:
17         r = r - b
18         q = q + 1
19     return [q, r]
```

On peut montrer que la fonction f(a, b) termine, en effet : la variable informatique r est un variant de boucle, il est à valeur dans $\mathbb{N}$ et qui décroit strictement à chaque

passage dans la boucle *while*. Tandis que si $b > 0$ et $a \leq 0$ le programme présente une boucle infinie !

Correction

Pour montrer qu'un algorithme est correct, il s'agit de montrer que quels que soient les paramètres vérifiant sa spécification, l'action de l'algorithme correspond à ce qui est attendu. Pour montrer la correction de l'algorithme, il s'agit de montrer que chacun des blocs effectue une action bien précise. Pour les blocs conditionnels (if, elif,...,else), il n'y a en général pas grand chose à dire de plus que le bloc lui-même. En revanche, analyser les boucles for et while est essentiel, car l'action de ces boucles n'est pas forcément évidente en première lecture. La notion essentielle pour montrer la correction des boucles est celle d'invariant de boucle.

Définition 1.1.3 *Un invariant de boucle est une propriété dépendant des variables de l'algorithme, qui est vérifiée à chaque passage dans la boucle.*

Les invariants de boucle nous aideront à comprendre pourquoi un algorithme est correct. Nous devons montrer trois choses, concernant un invariant de boucle :

— Initialisation : Il est vrai avant la première itération de la boucle.
— Conservation : S'il est vrai avant une itération de la boucle, il le reste avant l'itération suivante.
— Terminaison : Une fois terminée la boucle, l'invariant fournit une propriété utile qui aide à montrer la validité de l'algorithme.

Si les deux premières propriétés sont vérifiées, alors l'invariant est vrai avant chaque itération de la boucle. Notez la ressemblance avec la récurrence mathématique, la troisième propriété est peut-être la plus importante, vu que nous utilisons l'invariant de boucle pour prouver la validité de l'algorithme. Elle diffère aussi de l'usage habituel de

la récurrence mathématique, dans laquelle la phase inductive se répète indéfiniment ; ici, on arrête « l'induction » quand la boucle se termine.

Exemple Revenons à l'exemple ci-dessus

```
14 def f(a, b):
15     q, r = 0, a
16     while r >= b:
17         r = r - b
18         q = q + 1
19     return [q, r]
```

avec $a \geq 0$ et $b > 0$, nous avons vu que la fonction f termine toujours, avec la notion d'invariant de la boucle, on arrive à montrer qu'elle est correcte, dans le sens qu'elle renvoie toujours le quotient et le reste de la division euclidien de a par b. En effet, il est facile de voir que

$$P : \text{ " } a = bq + r \text{ "},$$

est invariant de la boucle while.

1. P est vraie initialement car $r = a$ et $q = 0$.
2. Lors d'une itération, on augmente q de 1 et on diminue r de b. Donc $bq + r$ reste le même, ce qui montre que P reste vraie.

Les entiers q et r renvoyés par f sont donc tels que $a = bq + r$ et $0 <= r < b$ (la boucle while termine !). $f(a, b)$ renvoie donc le quotient et le reste de la division euclidienne de a par b.

Complexité

Déterminer la complexité d'un algorithme, c'est évaluer les ressources nécessaires à son exécution (essentiellement la quantité de mémoire requise) et le temps de calcul

à prévoir. Ces deux notions dépendent de nombreux paramètres matériels qui sortent du domaine de l'algorithmique : nous ne pouvons attribuer une valeur absolue ni à la quantité de mémoire requise ni au temps d'exécution d'un algorithme donné. En revanche, il est souvent possible d'évaluer l'ordre de grandeur de ces deux quantités de manière à identifier l'algorithme le plus efficace au sein d'un ensemble d'algorithmes résolvant le même problème

La première étape consiste donc à préciser quelles sont les instructions élémentaires, c'est-à-dire celles qui seront considérées comme ayant un coût constant, indépendant de leurs paramètres. Parmi celles-ci figurent en général :

1. Les opérations arithmétiques (addition, soustraction, multiplication, division, modulo, partie entière,...).
2. La comparaisons de données (relation d'égalité, d'infériorité, . . .).
3. Le transfert de données (lecture et écriture dans un emplacement mémoire).
4. Les instructions de contrôle (branchement conditionnel et inconditionnel, appel à une fonction auxiliaire,...).

Une fois précisée la notion d'opération élémentaire, il convient de définir ce qu'on appelle la taille de l'entrée. Cette notion dépend du problème étudié : pour de nombreux problèmes, il peut s'agir du nombre d'éléments constituant les paramètres de l'algorithme (par exemple le nombre d'éléments du tableau dans le cas d'un algorithme de tri) ; dans le cas d'algorithmes de nature arithmétique (le calcul de n^k par exemple) il peut s'agir d'un entier passé en paramètre, voire du nombre de bits nécessaire à la représentation de ce dernier. Dans le cas de nos algorithmes présentés ici, la taille d'entrée sera le cardinal de l'ensemble des noeuds d'interpolation Z.

Une fois la taille n de l'entrée définie, il reste à évaluer en fonction de celle-ci le nombre $f(n)$ d'opérations élémentaires requises par l'algorithme. Mais même s'il est parfois possible d'en déterminer le nombre exact, on se contentera le plus souvent d'en donner l'ordre de grandeur à l'aide des notations de *Landau* : $O(n)$.

1.2 Interpolation de Lagrange à une variable $(p = 1)$

1.2.1 Définition

On considère un ensemble de noeuds [1] $Z = \{(z_0, ..., z_n)\}$, constitué de $(n+1)$ noeuds distincts appartenant à un corps commutatif $\mathbb{K}$, et on considère un vecteur $R = (r_i,\ i \in [\![0, n]\!])$ à valeurs dans $\mathbb{K}^{n+1}$. D'autre part, on note $\mathbb{K}_n[x]$ le sous-espace des polynômes de degré inférieur ou égal à n et à coefficients dans le corps $\mathbb{K}$.

Le problème de l'interpolation de Lagrange (1.1.1) associé à (Z, R) consiste à chercher un polynôme $P \in \mathbb{K}_n[x]$ vérifiant

$$P(z_i) = r_i := r_{z_i},\ \forall 0 \leq i \leq n.$$

L'objet de la suite de cette section est de rappeler les différentes méthodes historiques pour résoudre le principal résultat suivant.

Théorème 1.2.1 *Il existe un et un seul polynôme π dans $\mathbb{K}_n[x]$ interpolant f sur Z. En d'autres termes π est l'unique polynôme de degré $\leq n$ vérifiant*

$$\pi(z_i) = r_i,\ \forall 0 \leq i \leq n.$$

Remarques 1.2.1 *Notons ici; comme il est indiqué dans la sous-section 1.1.1, le problème d'interpolation à une variable est bien posé, et le sous espace $\mathbb{K}_n[x]$ est l'espace d'interpolation associé à Z de degré optimal.*

1. Dans cette section afin de simplifier et de garder la notation habituelle de l'interpolation à une variable, je prends, ici Z à $n+1$ points.

1.2.2 Écriture matricielle du problème et forme matricielle de l'interpolant

Chercher un polynôme $P(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{K}_n[x]$ solution du problème (1.1.1) revient à résoudre le système d'équations linéaires suivant

$$\begin{cases} a_0 + a_1 z_0 + ... + a_n z_0^n = r_0 \\ a_0 + a_1 z_1 + ... + a_n z_1^n = r_1 \\ \vdots \\ a_0 + a_1 z_n + ... + a_n z_n^n = r_n \end{cases},$$

d'inconnues les coefficients du polynôme interpolant $a_0, ..., a_n$ de matrice la matrice de Vandermonde

$$\mathbb{V}(Z) = \mathbb{V}(z_0, ..., z_n) = \left(z_i^j\right)_{0 \leq i,j \leq n},$$

et de second membre les valeurs de l'interpolation correspondantes $r_0, ..., r_n$. La résolution du problème (1.1.1) revient donc à résoudre l'équation matricielle suivante

$$\mathbb{V}(z_0, ..., z_n) \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_n \end{pmatrix}. \tag{1.2.1}$$

Comme on sait que

$$\det \mathbb{V}(z_0, ..., z_n) = \prod_{0 \leq i < j \leq n} (z_j - z_i) \neq 0,$$

alors on déduit que

Théorème 1.2.2 *Le problème (1.1.1) admet bien une et une seule solution π qu'on peut exprimer sous la forme matricielle suivante*

$$\pi(x) = [1 \ x \ \cdots \ x^n] \left(\mathbb{V}(z_0, ..., z_n)\right)^{-1} \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_n \end{pmatrix}, \tag{1.2.2}$$

Remarque 1.2.1 *L'expression matricielle (1.2.2) permet d'exprimer directement* π *dans la base canonique* $(1, x, ..., x^n)$ *de* $\mathbb{K}_n[x]$. *Cependant, cette méthode n'est pas très pratique car elle exige la résolution d'un système d'équations linéaires qui requiert un nombre* $O(n^3)$ *d'opérations si on utilise par exemple une méthode d'élimination de Gauss et de plus la matrice associée est une matrice de Vandermonde et on sait que ce genre de matrices est très mal conditionnée. La résolution numérique du système associé est entâchée de grosses erreurs numériques quand la taille devient très grande et les noeuds très proches. D'où, la nécessité de chercher d'autre méthodes plus efficaces.*

1.2.3 Formule de Lagrange et forme barycentrique du polynôme interpolant

Ne maîtrisant pas à son époque les grands outils de la résolution des équations linéaires, Lagrange (**1736-1813**) donna une forme plus pratique de π en l'exprimant dans une base autre que la base canonique. C'est ce que nous allons détailler dans cette section.

Aux noeuds $z_0, ..., z_n$ on associe les formes linéaires u_i de $\mathbb{K}_n[x],\ 0 \leq i \leq n$, définies par

$$< u_i, P >= P(z_i)\ ,\ 0 \leq i \leq n \text{ et } P \in \mathbb{K}_n[x],$$

alors on a

Théorème 1.2.3 $(u_0, ..., u_n)$ *est une base de l'espace dual* $\mathbb{K}_n[x]^*$ *et la base préduale associée est la famille formée des polynômes* $l_0, ..., l_n$ *donnés par*

$$l_j(x) = \prod_{k \neq j} \frac{x - z_k}{z_j - z_k}, \tag{1.2.3}$$

$l_0, ..., l_n$ *forment une base dit de type Lagrange de* $\mathbb{K}_n[x]$. *On les appelle aussi les polynômes caractéristiques de Lagrange associés à* $z_0, ..., z_n$. *De plus, pour tout polynôme*

$P \in \mathbb{K}_n[x]$ on a

$$P(x) = \sum_{i=0}^{n} < u_i, P > l_i(x) = \sum_{i=0}^{n} P(z_i) \prod_{k \neq i} \frac{x - z_k}{z_i - z_k}. \tag{1.2.4}$$

Corollaire 1.2.4 *Il existe bien un et un seul polynôme π solution du problème (1.1.1). De plus, π s'exprime sous la forme*

$$\pi(x) = \sum_{i=0}^{n} r_i l_i(x) = \sum_{i=0}^{n} P(z_i) \prod_{k \neq i} \frac{x - z_k}{z_i - z_k}. \tag{1.2.5}$$

L'expression (1.2.5) s'appelle forme de Lagrange de l'interpolant π.

Remarque 1.2.2 *La forme de Lagrange (1.2.5) de π, attribuée à Lagrange, nécessite moins d'opérations de l'ordre de $O(n^2)$ si on la compare avec la forme matricielle précédente. Mais si on désire évaluer π en plusieurs valeurs de x, alors il faudrait refaire tous les calculs pour chacune de ces valeurs. Ce qui peut devenir très coûteux si le nombre d'évaluations est très grand. D'où, l'idée de chercher une forme améliorée dite formule barycentrique du polynôme interpolant et qui repose sur la proposition suivante*

Proposition 1.2.5 *Les polynômes caractéristiques de Lagrange $l_0, ..., l_n$ vérifient*

$$\sum_{k=0}^{n} l_k = 1,$$

et en posant

$$\omega(x) = \prod_{k=0}^{n} (x - z_k), \tag{1.2.6}$$

on a, pour tout $0 \leq k \leq n$

$$l_k(x) = \frac{\omega(x)}{(x - z_k)\omega'(z_k)},$$

ce qui permet d'obtenir, en sommant ces inégalités

$$\sum_{k=0}^{n} \frac{\omega(x)}{(x - z_k)\omega'(z_k)} = \sum_{k=0}^{n} l_k(x) = 1,$$

soit que pour $x \notin Z$

$$\frac{1}{\omega(x)} = \sum_{k=0}^{n} \frac{1}{(x - z_k)\omega'(z_k)},$$

qui coïncide avec la décomposition en éléments simples de la fraction rationnelle $\frac{1}{\omega}$. *En injectant dans (1.2.5), on obtient la* ***formule barycentrique*** *du polynôme interpolant* π

$$\pi(x) = \frac{\sum_{k=0}^{n} \frac{r_k}{(x - z_k)\omega'(z_k)}}{\sum_{k=0}^{n} \frac{1}{(x - z_k)\omega'(z_k)}}.$$

En posant pour tout $0 \leq k \leq n$

$$\alpha_k = \frac{1}{\omega'(z_k)} = \frac{1}{\prod_{i \neq k} (z_i - z_k)},$$

l'expression précédente devient

$$\pi(x) = \frac{\sum_{k=0}^{n} \frac{r_k \alpha_k}{(x - z_k)}}{\sum_{k=0}^{n} \frac{\alpha_k}{(x - z_k)}}. \tag{1.2.7}$$

Remarque 1.2.3 *Le calcul de chaque* α_k *nécessite*

n *additions*,

$n - 1$ *multiplications*,

1 *division*,

soit pour obtenir l'expression de π *un total d'opérations de l'ordre* $O(n^2)$. *Mais chaque nouvelle évaluation de* $\pi(x)$, *en gardant en mémoire les valeurs des* α_k *et de* $r_k\alpha_k$, *ne va nécessiter que*

$3n + 1$ *additions*,

$2n + 3$ *divisions*,

soit un nombre d'opérations de l'ordre de $O(n)$.

Cependant, pour les trois méthodes citées ci-dessus, si on rajoute un autre noeud aux noeuds précédents et qu'on désire déterminer le nouveau interpolant, il faudrait refaire tous les calculs. Ce qui est un grand inconvénient numérique. On doit donc chercher d'autres méthodes encore plus efficaces.

Méthode d'Aitken-Neville

La méthode d'interpolation proposée séparémment par Aitken (**1895-1967**) et par Neville (**1889-1961**) permet de déterminer le polynôme interpolant π de manière récursive et donc elle est moins coûteuse que les méthodes précédentes. La méthode repose sur le théorème suivant

Théorème 1.2.6 *(d'Aitken) Considérons $n+1$ noeuds $z_0, ..., z_{n-1}, a, b$ de $\mathbb{K}$ distincts deux à deux. Et on se donne des valeurs d'interpolation arbitraires correspondantes $r_0, ..., r_{n-1}, \alpha, \beta$ de $\mathbb{K}$ aussi. On suppose connaître les polynômes P, Q interpolant $r_0, ..., r_{n-1}, \alpha$ en $z_0, ..., z_{n-1}, a$ et $r_0, ..., r_{n-1}, \beta$ en $z_0, ..., z_{n-1}, b$ respectivement. Alors le polynôme*

$$R(x) = \frac{(b-x)P(x) - (a-x)Q(x)}{b-a},$$

est le polynôme interpolant $r_0, ..., r_{n-1}, \alpha, \beta$ en $z_0, ..., z_{n-1}, a, b$.

On applique ce théorème à la détermination récursive du polynôme interpolant P_n de $r_0, ..., r_n$ aux noeuds $z_0, ..., z_n$ comme suit.

Pour tout $0 \leq j \leq k \leq n$ on construit un polynôme $P_{k,j}$ selon la règle de récurrence suivante

$$P_{k,0}(x) = r_k \text{ le polynôme constant interpolant } r_k \text{ en } z_k,$$
$$P_{k,j+1}(x) = \frac{(z_k - x)P_{j,j}(x) - (z_j - x)P_{k,j}(x)}{z_k - z_j} \ ; \ \text{pour } j < k.$$

Les $P_{k,j}$ peuvent être rangés dans le tableau triangulaire inférieur suivant

	0	1	$\cdots$	j	$\cdots$	k	$\cdots$	n	
0	$P_{0,0}$								$(z_0 - x)$
1	$P_{1,0}$	$P_{1,1}$							$(z_1 - x)$
$\vdots$			$\ddots$						$\vdots$
j	$P_{j,0}$	$P_{j,1}$	$\cdots$	$\boxed{P_{j,j}}$					$\boxed{(z_j - x)}$
$\vdots$					$\ddots$				$\vdots$
k	$P_{k,0}$	$P_{k,1}$	$\cdots$	$\boxed{P_{k,j}}$	$\cdots$	$P_{k,k}$			$\boxed{(z_k - x)}$
$\vdots$							$\ddots$		$\vdots$
n	$P_{n,0}$	$P_{n,1}$	$\cdots$	$P_{n,j}$	$\cdots$	$P_{n,k}$	$\cdots$	$P_{n,n}$	$(z_n - x)$

Schéma d'interpolation d'Aitken

où le terme $P_{k,j+1}$ est déterminé en divisant le déterminant $\begin{vmatrix} P_{j,j} & (z_j - x) \\ P_{k,j} & (z_k - x) \end{vmatrix}$ par $z_k - z_j$. On a alors le résultat suivant

Théorème 1.2.7 *Pour tout $0 \leq j \leq k \leq n$, le polynôme $P_{k,j}$ est l'interpolant de degré $\leq j$ de $r_0, ..., r_{j-1}, r_k$ aux noeuds $z_0, ..., z_{j-1}, z_k$. En particulier, $P_{n,n}$ est le polynôme d'interpolation π interpolant $r_0, ..., r_n$ en $z_0, ..., z_n$ solution du problème de départ.*

Remarque 1.2.4 *Contrairement aux méthodes précédentes la méthode d'interpolation d'Aitken est plus optimale puisque si on rajoute un $n+2$ ème noeud on n'est pas obligé de refaire tous les calculs mais il suffit de rajouter une ligne et une colonne au schéma précédent. Ce qui exige d'effectuer en conservant les autres polynômes $P_{j,j}$, $0 \leq j \leq n$, pour un calcul d'évaluation, $2n+3$ additions, $2(n+1)$ multiplications et $n+1$ divisions supplémentaires seulement.*

Différences divisées et formule de Newton

Dans une lettre adressée le **08-11-1676** à John Collins (**1624-1683**) publiée plus tard par Roger Cotes en **1711**, Newton proposa (mais sans preuve) une autre forme du polynôme interpolant π donnée par

$$\pi(x) = A_0 + A_1(x - z_0) + A_2(x - z_0)(x - z_1) + ... + A_n(x - z_0)...(x - z_{n-1}), \quad (1.2.8)$$

qui consiste à écrire π dans une autre base dite **de type Newton** plutôt que dans la base de Lagrange. Où, il donna sans démonstration l'expression récursive des A_k. Il a fallu attendre plus de deux siécles pour avoir une démonstration de manière séparée grâce à Aitken (**1929**) et Neville (**1932**). Pour cela, on introduit la notion de différence divisée :

Définition 1.2.1 *Etant données deux suites $z = (z_k)_{k\in\mathbb{N}}$ et $r = (r_k)_{k\in\mathbb{N}}$ de scalaires tels qu'en plus les z_k soient distincts deux à deux, on définit alors*

$$\begin{cases} \delta^0 r[z_i] = r_i \\ \delta^1 r[z_i, z_j] = \dfrac{\delta^0 r[z_j] - \delta^0 r[z_i]}{z_j - z_i} = \dfrac{r_j - r_i}{z_j - z_i}, \ \textit{pour } i \neq j \\ \delta^2 r[z_i, z_j, z_k] = \dfrac{\delta^1 r[z_j, z_k] - \delta^1 r[z_i, z_j]}{z_k - z_i}, \ \textit{pour } i, j, k \textit{ deux à deux distincts} \\ \delta^k r[z_{i_1}, z_{i_2}, ..., z_{i_k}] = \dfrac{\delta^{k-1} r[z_{i_2}, ..., z_{i_k}] - \delta^{k-1} r[z_{i_1}, ..., z_{i_{k-1}}]}{z_{i_k} - z_{i_1}}, \ \textit{pour } i_1, ..., i_k \ k \textit{ indices deux à deux distin} \end{cases}$$

Théorème 1.2.8 *Les coefficients $A_k,\ 0 \leq k \leq n$ de la formule de Newton précédente s'expriment alors comme des différences divisées comme suit*

$$A_k = \delta^k r[z_{0_1}, z_1, ..., z_k].$$

Le calcul des coefficients $A_k,\ 0 \leq k \leq n$ du polynôme π par la méthode des différences divisées peut s'effectuer par le schéma similaire à celui d'Aitken suivant

$k=0$	$k=1$	$k=2$	$\cdots$	$k=n-1$	$k=n$
$r_0=\delta^0 r[z_0]\searrow$					
	$\delta^1 r[z_0,z_1]\searrow$				
$r_1=\delta^0 r[z_1]\nearrow\!\!\searrow$		$\delta^2 r[z_0,z_1,z_2]\searrow$			
	$\delta^1 r[z_1,z_2]\nearrow\!\!\searrow$				
$r_2=\delta^0 r[z_2]\nearrow\!\!\searrow$		$\delta^2 r[z_1,z_2,z_3]\nearrow\!\!\searrow$			
				$\delta^{n-1} r[z_0,z_1,...,z_{n-1}]\searrow$	
					$\delta^n r[z_0,z_1,...,z_n]$
				$\delta^{n-1} r[z_1,z_2,...,z_n]\nearrow$	
$r_{n-1}=\delta^0 r[z_{n-1}]\nearrow\!\!\searrow$		$\delta^2 r[z_{n-2},z_{n-1},z_n]\nearrow$			
	$\delta^1 r[z_{n-1},z_n]\nearrow$				
$r_n=\delta^0 r[z_n]\nearrow$					

Schéma des différences divisées

Remarque 1.2.5 *1. Le calcul des coefficients* $A_k = \delta^n r[z_0, z_1, ..., z_n],\ 0 \leq k \leq n$, *par le schéma des différences précédent, nécessite* $n(n+1)$ *additions et* $\dfrac{n(n+1)}{2}$ *divisions.*

2. L'évaluation du polynôme π *en une valeur* x *s'effectue alors par la méthode d'Horner comme suit*

$$\pi(x) = A_0 + (x - z_0)\left(A_1 + (x - z_1)\left(A_2 + (x - z_2)\left(.... \left(A_{n-1} + (x - z_{n-1})A_n\right)\right)\right)\right),$$

ce qui permet d'optimiser le nombre d'opérations. Le nombre d'opérations supplémentaires pour chaque évaluation étant n *additions et multiplications.*

3. De plus, comme pour la méthode d'Aitken, la formule de Newton est récursive i.e, si on rajoute un noeud il suffit de rajouter dans le tableau ci-dessus une colonne et une ligne.

Chapitre -2

RMVPIA : un nouvel algorithme de calcul d'interpolation de Lagrange à plusieurs variables dans le cas d'une grille

Dans ce chapitre nous allons présenter un nouvel algorithme intitulé RMVPIA : Recursive MultiVariate Polynomial Interpolation Algorithm. Il s'agit de résoudre le problème d'interpolation de Lagrange à plusieurs variables dans le cas où l'ensemble des noeuds est une grille de $\mathbb{K}^p$.

Pour la lisibilité des résultats et des preuves, nous allons uniquement détailler l'étude du cas $p = 2$ et on se contentera d'en déduire la forme des algorithmes dans le cas général.

Ce chapitre est organisé comme suit : on commence par adapter les notations du problème d'interpolation (1.1.1) dans ce cas particulier, ensuite, nous définissons un ordre total dans l'ensemble des indices $\mathbb{N}_{(n_1,n_2)}$ (voir ci-après) correspondant à un choix

du parcours des noeuds. Puis, nous donnons la formulation matricielle du problème associé. Nous montrons ensuite que le polynôme d'interpolation peut être exprimé comme un complément de Schur. En utilisant l'identité de Sylvester, nous établirons certaines relations de récurrence, le RBVPIA sera déduit et une version simplifiée sera donnée. Nous donnons ensuite, quelques propriétés des algorithmes proposés. Dans la section 3, nous expliquerons comment généraliser ces algorithmes dans le cas de l'interpolation polynomiale à plusieurs variables ($p > 2$). La section 4 est consacrée à quelques exemples. On termine par donner le code Python des différents algorithmes.

2.1 Introduction

Étant données

— deux entiers naturels n_1, n_2.

— $\mathbb{N}_{(n_1,n_2)} = \{(i_1, i_2);\ 0 \leq i_1 \leq n_1 \text{ et } 0 \leq i_2 \leq n_2\}$ l'ensemble des indices.

— l'ensemble des noeuds d'interpolation $Z = \{z_{(i_1,i_2)} = (x_{i_1}, y_{i_2});\ (i_1, i_2) \in \mathbb{N}_{(n_1,n_2)}\}$, une grille dans $\mathbb{K}^2$ formée par

$$\mathbf{x} = [x_0, ..., x_{i_1}, ..., x_{n_1}],$$

et

$$\mathbf{y} = [y_0, ..., y_{i_2}, \cdots, y_{n_2}].$$

— $R = (r_{(i_1,i_2)};\ (i_1, i_2) \in \mathbb{N}_{(n_1,n_2)})$ l'ensemble des valeurs d'interpolation.

— $\Pi_{(n_1,n_2)}$ le sous-espace vectoriel de polynômes P à deux variables vérifiant

$$\deg_x P \leq n_1 \text{ et } \deg_y P \leq n_2,$$

où $\deg_x P$ (respectivement $\deg_y P$) désigne le degré du polynôme P par rapport à la variables x (respectivement y). Le polynôme P peut donc être exprimé sous l'une des formes suivantes

$$P = \sum_{(j_1,j_2)\in\mathbb{N}_{(n_1,n_2)}} a_{(j_1,j_2)} x^{j_1} y^{j_2} = \sum_{j_1=0}^{n_1}\sum_{j_2=0}^{n_2} a_{(j_1,j_2)} x^{j_1} y^{j_2} = \sum_{j_2=0}^{n_2}\sum_{j_1=0}^{n_1} a_{(j_1,j_2)} x^{j_1} y^{j_2}.$$

Comme on peut le constater dans [7, 66, 70, 81], $\Pi_{(n_1,n_2)}$ est l'espace polynomial naturel et optimal[7], adapté à l'interpolation polynomiale de Lagrange dans le cas particulier où l'ensemble des noeuds d'interpolation est une grille, qui est un cas particulier du cas plus général "lower set" introduit et étudié dans [5, 7, 8, 39, 40, 66, 70, 81]. Le problème de l'interpolation polynomiale bivariée (1.1.1) associé à Z et R, revient, dans ce cas, à trouver $P \in \Pi_{(n_1,n_2)}$ vérifiant les conditions d'interpolation suivantes

$$P(z_{(i_1,i_2)}) = r_{(i_1,i_2)}; \quad \text{pour } (i_1, i_2) \in \mathbb{N}_{(n_1,n_2)}.$$

Il est connu que le problème admet une et une seule solution donnée par

$$P = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} r_{(i_1,i_2)} L_{i_1}(x) L_{i_2}(y), \tag{2.1.1}$$

où $L_0(x), ..., L_{n_1}(x)$ et $L_0(y), ..., L_{n_2}(y)$ sont les polynômes d'interpolation de Lagrange caractéristiques associés aux listes $\mathbf{x}$ et $\mathbf{y}$ respectivement

$$L_{i_1}(x) = \prod_{0 \leq j_1 \leq n_1 \ ; \ j_1 \neq i_1} \frac{x - x_{j_1}}{x_{i_1} - x_{j_1}} \text{ et } L_{i_2}(y) = \prod_{0 \leq j_2 \leq n_2 \ ; \ j_2 \neq i_2} \frac{y - y_{j_2}}{y_{i_2} - y_{j_2}}.$$

Dans le paragraphe suivant, on propose une résolution aboutissant à un algorithme récursif et s'appuyant sur le complément de Schur permettant la détermination du polynôme P.

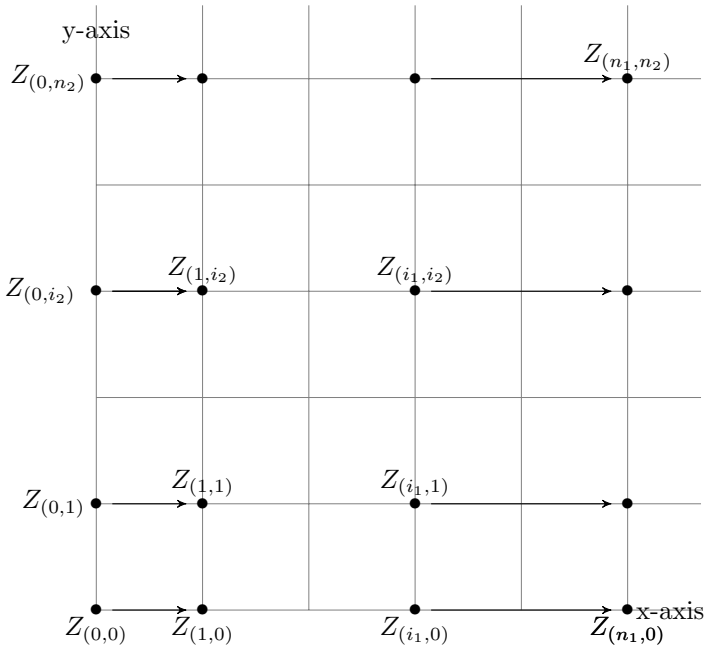

FIGURE -2.1 – Parcours de la grille, choix 1.

2.2 RBVPIA : l'algorithme à deux variables et ses propriétés

2.2.1 Choix d'un parcours des noeuds et quelques conventions

Dans cette section, nous définissons un ordre total dans $\mathbb{N}_{(n_1,n_2)}$ et nous introduisons une notation matricielle qui est adaptée au problème de l'interpolation polynomiale bivariée. Pour parcourir les différents noeuds de la grille Z, nous avons deux possibilités simples illustrées par la figure -2.1 et la figure -2.2.

Pour le premier choix, correspondant à la figure -2.1, le polynôme d'interpolation P peut être exprimé comme suit

$$P = \sum_{j_2=0}^{n_2} \sum_{j_1=0}^{n_1} a_{(j_1,j_2)} x^{j_1} y^{j_2}.$$

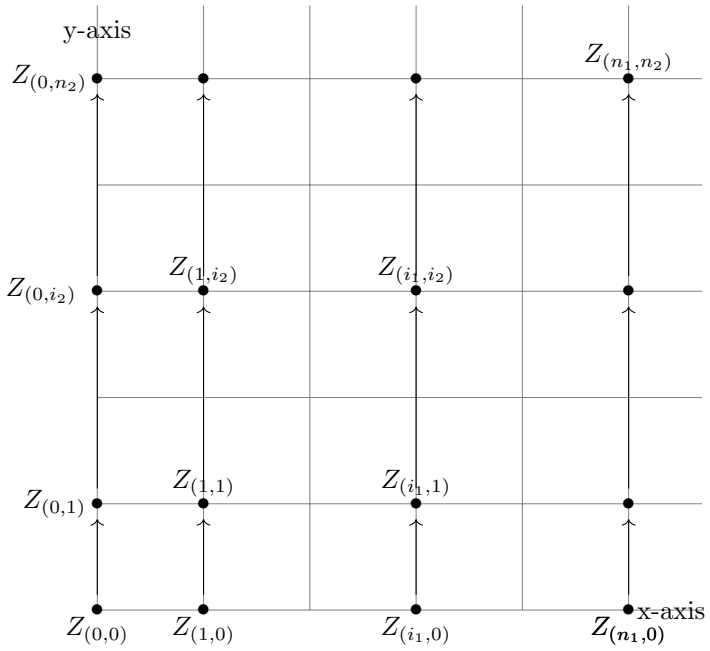

FIGURE -2.2 – Parcours de la grille, choix 2.

Et pour le second choix, figure -2.2, le polynôme d'interpolation P peut être exprimé comme suit

$$P = \sum_{j_1=0}^{n_1} \sum_{j_2=0}^{n_2} a_{(j_1,j_2)} x^{j_1} y^{j_2}.$$

Dans la suite de cette section, nous adoptons l'ordre du premier parcours illustré par la figure -2.1, il est définit comme suit

Soit (i_1, i_2) et (j_1, j_2) deux indices dans $\mathbb{N}_{(n_1,n_2)}$. Alors

$$(i_1, i_2) \leq (j_1, j_2) \text{ si, et seulement si, } (i_2 < j_2) \text{ ou } (i_2 = j_2 \text{ et } i_1 < j_1) \qquad (2.2.1)$$

avec égalité si, et seulement si, $i_1 = j_1$ et $i_2 = j_2$.

Lorsque $(i_1, i_2) \leq (j_1, j_2)$, on considère le sous-ensemble de $\mathbb{N}_{(n_1,n_2)}$

$$[(i_1, i_2), (j_1, j_2)] = \{(i_1, i_2), (i_1{+}1, i_2), ..., (n_1, i_2), (0, i_2{+}1), ..., (n_1, i_2{+}1), ..., (0, j_2), ..., (j_1, j_2)\},$$

qui contient $(j_2 - i_2)(n_1 + 1) + j_1 - i_1 + 1$ éléments.

On définit le prédécesseur d'un élément $(i_1, i_2) \in \mathbb{N}_{(n_1,n_2)}$ distinct de $(0, 0)$, noté $(i_1, i_2)^-$ par

$$(i_1, i_2)^- = \begin{cases} (i_1 - 1, i_2) & \text{si } \ i_1 > 0 \\ (n_1, i_2 - 1) & \text{si } \ i_1 = 0 \end{cases}.$$

Par commodité, nous désignons par

$$\mathbf{I}_{(i_1,i_2)} = [(0, 0), (i_1, i_2)],$$

dont la cardinal noté $d(i_1, i_2)$ est égal à $d(i_1, i_2) = i_2(n_1 + 1) + i_1 + 1$. Notons que $\mathbf{I}_{(n_1,n_2)} = \mathbb{N}_{(n_1,n_2)}$.

Maintenant, et en concordance avec l'ordre défini ci-dessus, nous introduisons une autre façon d'indexer les coefficients des matrices qui seront utilisées.

Soient (i_1, i_2) et (j_1, j_2) deux indices dans $\mathbb{N}_{(n_1,n_2)}$.

1. Un vecteur ligne de taille $d(i_1, i_2)$ sera représenté par

$$a = [a_{(0,0)}...a_{(n_1,0)}\ a_{(0,1)}...a_{(n_1,1)}...a_{(0,i_2)}...a_{(i_1,i_2)}].$$

L'ensemble de ces vecteurs sera noté $\mathbb{K}^{(i_1,i_2)}$.

2. Une matrice $A \in \mathbb{K}^{(i_1,i_2)\times(j_1,j_2)}$ sera représentée comme suit

$$A = \left[a_{(q_1,q_2),(r_1,r_2)}\right]_{\substack{(q_1,q_2)\in \mathbf{I}_{(i_1,i_2)} \\ (r_1,r_2)\in \mathbf{I}_{(j_1,j_2)}}}$$

$$= \begin{bmatrix} a_{(0,0),(0,0)} & \cdots & a_{(0,0),(n_1,0)} & a_{(0,0),(0,1)} & \cdots & a_{(0,0),(r_1,r_2)} & \cdots & a_{(0,0),(j_1,j_2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(n_1,0),(0,0)} & \cdots & a_{(n_1,0),(n_1,0)} & a_{(n_1,0),(0,1)} & \cdots & a_{(n_1,0),(r_1,r_2)} & \cdots & a_{(n_1,0),(j_1,j_2)} \\ a_{(0,1),(0,0)} & \cdots & a_{(0,1),(n_1,0)} & a_{(0,1),(0,1)} & \cdots & a_{(0,1),(r_1,r_2)} & \cdots & a_{(0,1),(j_1,j_2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(0,q_2),(0,0)} & \cdots & a_{(0,q_2),(n_1,0)} & a_{(0,q_2),(0,1)} & \cdots & a_{(0,q_2),(r_1,r_2)} & \cdots & a_{(0,q_2),(j_1,j_2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(q_1,q_2),(0,0)} & \cdots & a_{(q_1,q_2),(n_1,0)} & a_{(q_1,q_2),(0,1)} & \cdots & a_{(q_1,q_2),(r_1,r_2)} & \cdots & a_{(q_1,q_2),(j_1,j_2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{(i_1,i_2),(0,0)} & \cdots & a_{(i_1,i_2),(n_1,0)} & a_{(i_1,i_2),(0,1)} & \cdots & a_{(i_1,i_2),(r_1,r_2)} & \cdots & a_{(i_1,i_2),(j_1,j_2)} \end{bmatrix}.$$

L'ensemble de ces matrices sera noté $\mathbb{K}^{(i_1,i_2)\times(j_1,j_2)}$. Et lorsque $(i_1,i_2) = (j_1,j_2)$ la matrice A sera appelée une matrice carrée d'ordre (i_1,i_2).

Remarque 2.2.1 .

Selon le deuxième choix du parcours de la grille figure -2.2, l'ordre sera défini par

$$(i_1,i_2) \leq (j_1,j_2) \text{ si et seulement si } (i_1 < j_1) \text{ ou } (i_1 = j_1 \text{ et } i_2 < j_2), \qquad (2.2.2)$$

$$\text{avec égalité si } i_1 = j_1 \text{ et } i_2 = j_2.$$

Dans ce cas, les conventions et définitions ci-dessus seront naturellement adaptées.

2.2.2 La construction de l'algorithme RBVPIA

Nous commençons cette section en donnant la formulation matricielle de l'interpolation polynomiale bivariée. Ensuite, nous montrerons que le polynôme interpolant peut

être exprimé comme un complément de Schur. Pour calculer récursivement ce polynôme, nous utiliserons l'identité matricielle de Sylvester et nous donnerons le RBVPIA : Recursive Bivariate Polynomial interpolation algorithm.

Dans la suite, nous désignons par $\mathbf{U}_{(n_1,n_2)}$ la base canonique de $\Pi_{(n_1,n_2)}$ qui peut être exprimée comme un produit de Kronecker

$$\begin{aligned}\mathbf{U}_{(n_1,n_2)} &= [x^{j_1}y^{j_2}]_{(j_1,j_2)\in\mathbb{N}_{(n_1,n_2)}} \\ &= [1 \ x \ ... \ x^{n_1} \ y \ xy \ ... \ x^{n_1}y \ ... \ y^{n_2} \ ... \ x^{n_1}y^{n_2}] \\ &= [1 \ y \ ... \ y^{n_2}] \otimes [1 \ x \ ... \ x^{n_1}].\end{aligned} \tag{2.2.3}$$

En outre, pour tout $(i_1, i_2), (j_1, j_2) \in \mathbb{N}_{(n_1,n_2)}$, nous adoptons la notation

$$z_{(i_1,i_2)}^{(j_1,j_2)} = x_{i_1}^{j_1} y_{i_2}^{j_2}.$$

Le problème d'interpolation polynomiale bivariée (1.1.1) associé à (Z, R), consiste à déterminer le polynôme $P \in \Pi_{(n_1,n_2)}$

$$P = \sum_{j_2=0}^{n_2} \sum_{j_1=0}^{n_1} a_{(j_1,j_2)} x^{j_1} y^{j_2},$$

vérifiant

$$P\left(z_{(i_1,i_2)}\right) = \sum_{j_2=0}^{n_2} \sum_{j_1=0}^{n_1} a_{(j_1,j_2)} z_{(i_1,i_2)}^{(j_1,j_2)} = r_{(i_1,i_2)}, \ \text{ pour } \ (i_1, i_2) \in \mathbb{N}_{(n_1,n_2)}. \tag{2.2.4}$$

Introduisons la matrice carrée de $\mathbb{K}^{(n_1,n_2)\times(n_1,n_2)}$

$$\mathbf{D}_{(n_1,n_2)} = [z_{(i_1,i_2)}^{(j_1,j_2)}]_{(0,0)\leq(i_1,i_2),(j_1,j_2)\leq(n_1,n_2)} \tag{2.2.5}$$

$$= \begin{bmatrix} 1 & x_0 & \cdots & x_0^{n_1} & y_0 & \cdots & x_0^{n_1}y_0 & \cdots & y_0^{n_2} & \cdots & x_0^{n_1}y_0^{n_2} \\ 1 & x_1 & \cdots & x_1^{n_1} & y_0 & \cdots & x_1^{n_1}y_0 & \cdots & y_0^{n_2} & \cdots & x_1^{n_1}y_0^{n_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n_1} & \cdots & x_{n_1}^{n_1} & y_0 & \cdots & x_{n_1}^{n_1}y_0 & \cdots & y_0^{n_2} & \cdots & x_{n_1}^{n_1}y_0^{n_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_0 & \cdots & x_0^{n_1} & y_{n_2} & \cdots & x_0^{n_1}y_{n_2} & \cdots & y_{n_2}^{n_2} & \cdots & x_0^{n_1}y_{n_2}^{n_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n_1} & \cdots & x_{n_1}^{n_1} & y_{n_2} & \cdots & x_{n_1}^{n_1}y_{n_2} & \cdots & y_{n_2}^{n_2} & \cdots & x_{n_1}^{n_1}y_{n_2}^{n_2} \end{bmatrix}.$$

On a

Lemme 2.2.1 $\mathbf{D}_{(n_1,n_2)}$ *est une matrice inversible.*

Preuve: En associant à $\mathbf{x} = [x_0, ..., x_{n_1}]$ et $\mathbf{y} = [y_0, ..., y_{n_2}]$ les matrices de Vandermonde notées

$$\mathbf{V}_{n_1}(\mathbf{x}) = [x_{i_1}^{j_1}]_{0\leq i_1,j_1\leq n_1} = \begin{bmatrix} 1 & x_0 & \cdots & x_0^{n_1} \\ 1 & x_1 & \cdots & x_1^{n_1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n_1} & \cdots & x_{n_1}^{n_1} \end{bmatrix} ; \mathbf{V}_{n_2}(\mathbf{y}) = [y_{i_2}^{j_2}]_{0\leq i_2,j_2\leq n_2} = \begin{bmatrix} 1 & y_0 & \cdots & y_0^{n_2} \\ 1 & y_1 & \cdots & y_1^{n_2} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & y_{n_2} & \cdots & y_{n_2}^{n_2} \end{bmatrix},$$

on remarque que $\mathbf{D}_{(n_1,n_2)}$ s'écrit par blocs sous la forme

$$\mathbf{D}_{(n_1,n_2)} = \left[\begin{array}{c|c|c|c} \mathbf{V}_{n_1}(\mathbf{x}) & y_0\mathbf{V}_{n_1}(\mathbf{x}) & \cdots & y_0^{n_2}\mathbf{V}_{n_1}(\mathbf{x}) \\ \hline \mathbf{V}_{n_1}(\mathbf{x}) & y_1\mathbf{V}_{n_1}(\mathbf{x}) & \cdots & y_1^{n_2}\mathbf{V}_{n_1}(\mathbf{x}) \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{V}_{n_1}(\mathbf{x}) & y_{n_2}\mathbf{V}_{n_1}(\mathbf{x}) & \cdots & y_{n_2}^{n_2}\mathbf{V}_{n_1}(\mathbf{x}) \end{array}\right],$$

ce qui permet de reconnaitre un produit de Kronecker

$$\mathbf{D}_{(n_1,n_2)} = \mathbf{V}_{n_2}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}),$$

ce qui entraîne en particulier que

$$\begin{aligned}\det \mathbf{D}_{(n_1,n_2)} &= (\det \mathbf{V}_{n_2}(\mathbf{y}))^{n_1+1} (\det \mathbf{V}_{n_1}(\mathbf{x}))^{n_2+1} \\ &= \prod_{0\le i_2<j_2\le n_2} (y_{j_2}-y_{i_2})^{n_1+1} \prod_{0\le i_1<j_1\le n_1} (x_{j_1}-x_{i_1})^{n_2+1},\end{aligned}$$

et comme les x_{i_1} d'un côté et les y_{i_2} de l'autre sont distincts deux à deux, alors $\det \mathbf{D}_{(n_1,n_2)} \neq 0$. Ce qui montre que la matrice $\mathbf{D}_{(n_1,n_2)}$ est bien inversible. ■

Nous allons maintenant montrer que le polynôme d'interpolation P peut être exprimé comme un complément de Schur.

Théorème 2.2.1 .

Si $\mathbf{a} = [a_{(0,0)}...a_{(n_1,0)}\ a_{(0,1)}...a_{(n_1,1)}...a_{(0,n_2)}...a_{(n_1,n_2)}]^T$ *est le vecteur colonne des coefficients du polynôme d'interpolation* P *alors* $\mathbf{a}$ *est la solution unique du système linéaire*

$$\mathbf{D}_{(n_1,n_2)}\ \mathbf{a} = \mathbf{r},$$

où

$$\mathbf{r} = [r_{(0,0)}...r_{(n_1,0)}\ r_{(0,1)}...r_{(n_1,1)}...r_{(0,n_2)}...r_{(n_1,n_2)}]^T, \tag{2.2.6}$$

est le vecteur colonne formé par les valeurs d'interpolation. Donc, P *peut s'écrire matriciellement sous la forme*

$$P = \mathbf{U}_{(n_1,n_2)}\mathbf{D}^{-1}_{(n_1,n_2)}\ \mathbf{r}, \tag{2.2.7}$$

où $\mathbf{U}_{(n_1,n_2)}$ *est donné par (2.2.3). Ce qui peut se mettre aussi sous la forme d'un complément de Schur*

$$P = -\left(\left[\begin{array}{c|c} \mathbf{0} & \mathbf{U}_{(n_1,n_2)} \\ \hline \mathbf{r} & \mathbf{D}_{(n_1,n_2)} \end{array}\right] / \mathbf{D}_{(n_1,n_2)}\right). \tag{2.2.8}$$

Preuve: Les coefficients du polynôme d'interpolation solution du problème (2.2.4) vérifient le système d'équations linéaires

$$\sum_{(j_1,j_2)\in\mathbb{N}_{(n_1,n_2)}} a_{(j_1,j_2)} z^{(j_1,j_2)}_{(i_1,i_2)} = r_{(i_1,i_2)}\ ;\ \forall (i_1,i_2)\in\mathbb{N}_{(n_1,n_2)}\ .$$

Ce qui s'écrit

$$\mathbf{D}_{(n_1,n_2)}\ \mathbf{a} = \mathbf{r}\ ,$$

ou encore

$$\mathbf{a} = \mathbf{D}^{-1}_{(n_1,n_2)}\ r\ ,$$

par suite

$$\begin{aligned} P &= \sum_{(j_1,j_2)\in\mathbb{N}_{(n_1,n_2)}} a_{(j_1,j_2)} x^{j_1} y^{j_2} \\ &= \mathbf{U}_{(n_1,n_2)}\ \mathbf{a} \\ &= \mathbf{U}_{(n_1,n_2)}\mathbf{D}^{-1}_{(n_1,n_2)}\mathbf{r}, \end{aligned}$$

où $\mathbf{U}_{(n_1,n_2)}$ est donnée par 2.2.3. Ce qui conduit au résultat. ∎

En tant que complément de Schur, P peut être déterminé récursivement en appliquant l'identité matricielle de Sylvester. Pour cela, nous aurons besoin de montrer que la matrice $\mathbf{D}_{(n_1,n_2)}$ est fortement inversible. Ce qui signifie que toutes ses matrices mineurs principales sont inversibles.

Pour chaque (k_1,k_2) de $\mathbb{N}_{(n_1,n_2)}$ notons $\mathbf{D}_{(k_1,k_2)}$ la matrice mineur principale d'ordre (k_1,k_2) de $\mathbf{D}_{(n_1,n_2)}$ définie par

$$\begin{aligned} \mathbf{D}_{(k_1,k_2)} &= [z^{(j_1,j_2)}_{(i_1,i_2)}]_{(0,0)\le(i_1,i_2),(j_1,j_2)\le(k_1,k_2)} \qquad (2.2.9) \\ &= \begin{bmatrix} 1 & x_0 & \cdots & x_0^{n_1} & y_0 & \cdots & x_0^{n_1}y_0 & \cdots & y_0^{k_2} & \cdots & x_0^{k_1}y_0^{k_2} \\ 1 & x_1 & \cdots & x_1^{n_1} & y_0 & \cdots & x_1^{n_1}y_0 & \cdots & y_0^{k_2} & \cdots & x_1^{k_1}y_0^{k_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n_1} & \cdots & x_{n_1}^{n_1} & y_0 & \cdots & x_{n_1}^{n_1}y_0 & \cdots & y_0^{k_2} & \cdots & x_{n_1}^{k_1}y_0^{k_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_0 & \cdots & x_0^{n_1} & y_{k_2} & \cdots & x_0^{n_1}y_{k_2} & \cdots & y_{k_2}^{k_2} & \cdots & x_0^{k_1}y_{k_2}^{k_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{k_1} & \cdots & x_{k_1}^{n_1} & y_{k_2} & \cdots & x_{k_1}^{n_1}y_{k_2} & \cdots & y_{k_2}^{k_2} & \cdots & x_{k_1}^{k_1}y_{k_2}^{k_2} \end{bmatrix}. \end{aligned}$$

Tout d'abord on va établir une relation de récurrence entre $\det \mathbf{D}_{(k_1,k_2)}$ et $\det \mathbf{D}_{(k_1,k_2)^-}$ pour tout (k_1,k_2) de $\mathbb{N}_{(n_1,n_2)}$ distinct de $(0,0)$, où on rappelle que

$$(k_1,k_2)^- = \begin{cases} (k_1-1,k_2) \text{ si } k_1 \geq 1 \\ (n_1,k_2-1) \text{ si } k_1 = 0. \end{cases}$$

Lemme 2.2.2 *Pour chaque (k_1,k_2) de $\mathbb{N}_{(n_1,n_2)}$ distinct de $(0,0)$, on a*

$$\det \mathbf{D}_{(k_1,k_2)} = \prod_{i_1=0}^{k_1-1} (x_{k_1} - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y_{k_2} - y_{i_2}) \det \mathbf{D}_{(k_1,k_2)^-}. \tag{2.2.10}$$

Preuve: Remarquons d'abord que $\mathbf{D}_{(k_1,k_2)}$ peut se décomposer en blocs comme suit

$$\mathbf{D}_{(k_1,k_2)} = \begin{bmatrix} \mathbf{V}_{n_1}(\mathbf{x}) & y_0\mathbf{V}_{n_1}(\mathbf{x}) & \cdots & y_0^{k_2-1}\mathbf{V}_{n_1}(\mathbf{x}) & y_0^{k_2}\mathbf{V}_{n_1,k_1}(\mathbf{x}) \\ \mathbf{V}_{n_1}(\mathbf{x}) & y_1\mathbf{V}_{n_1}(\mathbf{x}) & \cdots & y_1^{k_2-1}\mathbf{V}_{n_1}(\mathbf{x}) & y_1^{k_2}\mathbf{V}_{n_1,k_1}(\mathbf{x}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{V}_{n_1}(\mathbf{x}) & y_{k_2-1}\mathbf{V}_{n_1}(\mathbf{x}) & \cdots & y_{k_2-1}^{k_2-1}\mathbf{V}_{n_1}(\mathbf{x}) & y_{k_2-1}^{k_2}\mathbf{V}_{n_1,k_1}(\mathbf{x}) \\ \mathbf{V}_{k_1,n_1}(\mathbf{x}) & y_{k_2}\mathbf{V}_{k_1,n_1}(\mathbf{x}) & \cdots & y_{k_2}^{k_2-1}\mathbf{V}_{k_1,n_1}(\mathbf{x}) & y_{k_2}^{k_2}\mathbf{V}_{k_1}(\mathbf{x}) \end{bmatrix}$$

$$= \left[\begin{array}{cccc|c} & & & & y_0^{k_2}\mathbf{V}_{n_1,k_1}(\mathbf{x}) \\ & \mathbf{V}_{k_2-1}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}) & & & y_1^{k_2}\mathbf{V}_{n_1,k_1}(\mathbf{x}) \\ & & & & \vdots \\ & & & & y_{k_2-1}^{k_2}\mathbf{V}_{n_1,k_1}(\mathbf{x}) \\ \hline \mathbf{V}_{k_1,n_1}(\mathbf{x}) & y_{k_2}\mathbf{V}_{k_1,n_1}(\mathbf{x}) & \cdots & y_{k_2}^{k_2-1}\mathbf{V}_{k_1,n_1}(\mathbf{x}) & y_{k_2}^{k_2}\mathbf{V}_{k_1}(\mathbf{x}) \end{array}\right],$$

où $\mathbf{V}_{k,l}(\mathbf{x})$ désigne la matrice obtenue de la matrice de Vandermonde $\mathbf{V}_{n_1}(\mathbf{x})$ en ne retenant que les $k+1$ premières lignes et les $l+1$ premières colonnes

$$\mathbf{V}_{k,l}(\mathbf{x}) = \begin{bmatrix} 1 & x_0 & \cdots & x_0^l \\ 1 & x_1 & \cdots & x_1^l \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_k & \cdots & x_k^l \end{bmatrix}.$$

Ensuite remarquons que si $k_2 = 0$ alors $\mathbf{D}_{(k_1,k_2)} = \mathbf{D}_{(k_1,0)} = \mathbf{V}_{k_1}(\mathbf{x})$, ce qui montre que

$$\det \mathbf{D}_{(k_1,k_2)} = \prod_{0 \leq i_1 < j_1 \leq k_1} (x_{j_1} - x_{i_1}) = \prod_{i_1=0}^{k_1-1} (x_{k_1} - x_{i_1}) \det \mathbf{D}_{(k_1,k_2)^-},$$

puisque $\mathbf{D}_{(k_1,k_2)^-} = \mathbf{D}_{(k_{1-1},0)} = \mathbf{V}_{k_{1-1}}(\mathbf{x})$, d'où, la relation dans ce cas.

Supposons maintenant que $k_2 \geq 1$. Deux cas se présentent.

Premier cas : $k_1 = 0$. On considère le polynôme en y défini par le déterminant

$$Q := \left|\begin{array}{c|c} \mathbf{V}_{k_2-1}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}) & \begin{array}{c} y_0^{k_2}\mathbf{V}_{n_1,0}(\mathbf{x}) \\ y_1^{k_2}\mathbf{V}_{n_1,0}(\mathbf{x}) \\ \vdots \\ y_{k_2-1}^{k_2}\mathbf{V}_{n_1,0}(\mathbf{x}) \end{array} \\ \hline 1\ x_0 \cdots x_0^{n_1} \quad y\ \cdots x_0^{n_1} y \quad \cdots \quad y^{k_2-1}\ \cdots x_0^{n_1} y^{k_2-1} & y^{k_2} \end{array}\right|.$$

Q est un polynôme en y de degré k_2 et de coefficient dominant $\det(\mathbf{V}_{k_2-1}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}))$. D'autre part, on remarque que : $Q(y_0) = Q(y_1) = ... = Q(y_{k_2-1}) = 0$. Ce qui montre que

$$Q = \prod_{i_2=0}^{k_2-1} (y - y_{i_2})\ \det \mathbf{V}_{k_2-1}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}),$$

on en déduit que

$$\begin{aligned} \det \mathbf{D}_{(0,k_2)} &= \prod_{i_2=0}^{k_2-1} (y - y_{i_2})\ \det \mathbf{V}_{k_2-1}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}) \\ &= \prod_{i_2=0}^{k_2-1} (y - y_{i_2})\ \det \mathbf{D}_{(n_1,k_2-1)}, \end{aligned}$$

ce qui assure la formule de récurrence dans ce cas.

Deuxième cas : $k_1 \geq 1$. On considère le polynôme à deux variables W défini par

$$W := \left|\begin{array}{c|c|c} \mathbf{V}_{k_2-1}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}) & \begin{array}{c} y_0^{k_2}\mathbf{V}_{n_1,k_1-1}(\mathbf{x}) \\ y_1^{k_2}\mathbf{V}_{n_1,k_1-1}(\mathbf{x}) \\ \vdots \\ y_{k_2-1}^{k_2}\mathbf{V}_{n_1,k_1-1}(\mathbf{x}) \end{array} & \begin{array}{c} y_0^{k_2}\mathbf{C}_{n_1,k_1}(\mathbf{x}) \\ y_1^{k_2}\mathbf{C}_{n_1,k_1}(\mathbf{x}) \\ \vdots \\ y_{k_2-1}^{k_2}\mathbf{C}_{n_1,k_1}(\mathbf{x}) \end{array} \\ \hline \mathbf{V}_{k_{1-1},n_1}(\mathbf{x}) \cdots y_{k_2}^{k_2-1}\mathbf{V}_{k_{1-1},n_1}(\mathbf{x}) & y_{k_2}^{k_2}\mathbf{V}_{k_{1-1}}(\mathbf{x}) & y_{k_2}^{k_2}\mathbf{C}_{k_{1-1},k_1}(\mathbf{x}) \\ \hline 1\ x...x^{n_1} \cdots y^{k_2-1}...x^{n_1}y^{k_2-1} & y^{k_2}...x^{k_1-1}y^{k_2} & x^{k_1}y^{k_2} \end{array}\right|, \tag{2.2.11}$$

où, $\mathbf{C}_{i,k_1}(\mathbf{x})$ désigne la dernière colonne de $\mathbf{V}_{i,k_1}(\mathbf{x})$ pour $i = n_1$ ou $i = k_{1-1}$. On voit alors que W s'écrit sous la forme

$$W = \sum_{j_2=0}^{k_2} P_{j_2}(x) y^{j_2},$$

où P_{j_2} est un polynôme en x de degré $\leq n_1$ pour $j_2 = 0, ..., k_2 - 1$ et de degré $\leq k_1$ pour $j_2 = k_2$. On remarque aussi que $W(z_{(k_1,k_2)}) = \det \mathbf{D}_{(k_1,k_2)}$ et que le coefficient du terme en $x^{k_1}y^{k_2}$ est $\det \mathbf{D}_{(k_1-1,k_2)} = \det \mathbf{D}_{(k_1,k_2)^-}$. D'autre part, on a

$$\begin{array}{l} W\left(z_{(i_1,i_2)}\right) = 0 \ \forall 0 \leq i_1 \leq n_1 \text{ et } 0 \leq i_2 \leq k_2 - 1 \\ W\left(z_{(i_1,k_2)}\right) = 0 \ \forall 0 \leq i_1 \leq k_1 - 1 \end{array},$$

ce qui entraîne qu'en particulier, pour tout $0 \leq i_1 \leq k_1 - 1$ fixé, le polynôme en y

$$W(x_{i_1}, y) = \sum_{j_2=0}^{k_2} P_{j_2}(x_{i_1}) y^{j_2},$$

de degré $\leq k_2$ et admet $k_2 + 1$ racines distinctes, il est donc nul ($\mathbb{K}$ étant un corps infini). Ceci montre que les coefficients $P_{j_2}(x_{i_1}) = 0$ et que chacun des polynômes $P_{j_2}(x)$ se factorise sous la forme

$$P_{j_2}(x) = \prod_{i_1=0}^{k_1-1} (x - x_{i_1})\ Q_{j_2}(x),$$

où, $Q_{j_2}(x)$ est un polynôme de degré $\leq n_1 - k_1$ pour $j_2 = 0, ..., k_2 - 1$ et $Q_{k_2}(x)$ est un polynôme constant. Cela permet de factoriser W sous la forme

$$W = \prod_{i_1=0}^{k_1-1} (x - x_{i_1})\ S,$$

où $S = \left(\sum_{j_2=0}^{k_2} Q_{j_2}(x) y^{j_2} \right)$ est un polynôme à deux variables tel que

$$\deg_x S \leq n_1 - k_1,$$

$$\deg_y S \leq k_2,$$

et vérifie de plus

$$S\left(x_{i_1}, y_{i_2}\right) = 0 \ \forall k_1 \leq i_1 \leq n_1 \text{ et } 0 \leq i_2 \leq k_2 - 1,$$

ce qui entraîne par un raisonnement analogue au précédent mais cette fois-ci sur x, que S se factorise sous la forme

$$S = \prod_{i_2=0}^{k_2-1} (y - y_{i_2})\ T,$$

où T est alors un polynôme tel que $\deg_y T(x, y) \leq 0$ et $\deg_x T \leq n_1 - k_1$. Ainsi, W se met sous la forme

$$W = \prod_{i_1=0}^{k_1-1} (x - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y - y_{i_2})\ T(x),$$

en remarquant enfin l'absence, dans 2.2.11,des termes en $x^{j_1}y^{k_2}$ tel que $j_1 > k_1$. On en déduit que T est nécessairement constant égal au coefficient du terme en $x^{k_1}y^{k_2}$. En définitive, on arrive à

$$W = \prod_{i_1=0}^{k_1-1}(x - x_{i_1}) \prod_{i_2=0}^{k_2-1}(y - y_{i_2}) \ \det \mathbf{D}_{(k_1,k_2)^-}, \tag{2.2.12}$$

ce qui permet de conclure. ■

Lemme 2.2.3 *Pour chaque (k_1, k_2) de $\mathbb{N}_{(n_1,n_2)}$ on a*

$$\begin{aligned}\det \mathbf{D}_{(k_1,k_2)} \ = & \prod_{0\le i_1<j_1\le k_1}(x_{k_1} - x_{i_1}) \prod_{i_2=0}^{k_2-1}(y_{k_2} - y_{i_2})^{k_1+1} \times \\ & \prod_{0\le i_1<j_1\le n_1}(x_{j_1} - x_{i_1})^{k_2} \prod_{0\le i_2<j_2\le k_2-1}(y_{j_2} - y_{i_2})^{n_1+1}.\end{aligned} \tag{2.2.13}$$

Preuve: Remarquons d'abord que si $k_2 = 0$ alors $\mathbf{D}_{(k_1,k_2)} = \mathbf{D}_{(k_1,0)} = \mathbf{V}_{k_1}(\mathbf{x})$, ce qui montre que

$$\det \mathbf{D}_{(k_1,0)} = \prod_{0\le i_1<j_1\le k_1}(x_{j_1} - x_{i_1})$$

et justifie la formule. De même on peut remarquer que lorsque $k_1 = n_1$ pour tout $0 \le k_2 \le n_2$, on a

$$\mathbf{D}_{(k_1,k_2)} \ = \mathbf{D}_{(n_1,k_2)} = \mathbf{V}_{k_2}(\mathbf{y}) \otimes \mathbf{V}_{n_1}(\mathbf{x}),$$

il s'en suit que

$$\begin{aligned}\det \mathbf{D}_{(n_1,k_2)} &= (\det \mathbf{V}_{k_2}(\mathbf{y}))^{n_1+1} (\det \mathbf{V}_{n_1}(\mathbf{x}))^{k_2+1} \\ &= \prod_{0\le i_1<j_1\le n_1}(x_{j_1} - x_{i_1})^{k_2+1} \prod_{0\le i_2<j_2\le k_2}(y_{k_2} - y_{i_2})^{n_1+1},\end{aligned}$$

ce qui est cohérent avec la formule. Supposons $k_2 \ge 1$, alors pour $k_1 = 0$, on obtient par application du lemme précédent

$$\begin{aligned}\det \mathbf{D}_{(k_1,k_2)} \ &= \det \mathbf{D}_{(0,k_2)} = \prod_{i_2=0}^{k_2-1}(y_{k_2} - y_{i_2}) \det \mathbf{D}_{(n_1,k_2-1)} \\ &= \prod_{i_2=0}^{k_2-1}(y_{k_2} - y_{i_2}) \prod_{0\le i_1<j_1\le n_1}(x_{j_1} - x_{i_1})^{k_2} \prod_{0\le i_2<j_2\le k_2-1}(y_{j_2} - y_{i_2})^{n_1+1},\end{aligned}$$

ce qui s'accorde avec la formule. Maintenant, si on suppose la formule vérifiée pour $k_1 - 1$ telle que $1 \leq k_1 \leq n_1$ alors et par application du lemme précédent

$$\begin{aligned}
\det \mathbf{D}_{(k_1,k_2)} &= \prod_{i_1=0}^{k_1-1} (x_{k_1} - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y_{k_2} - y_{i_2}) \det \mathbf{D}_{(k_1-1,k_2)} \\
&= \prod_{i_1=0}^{k_1-1} (x_{k_1} - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y_{k_2} - y_{i_2}) \prod_{0 \leq i_1 < j_1 \leq k_1-1} (x_{j_1} - x_{i_1}) \times \\
&\prod_{i_2=0}^{k_2-1} (y_{k_2} - y_{i_2})^{k_1} \prod_{0 \leq i_1 < j_1 \leq n_1} (x_{j_1} - x_{i_1})^{k_2} \prod_{0 \leq i_2 < j_2 \leq k_2-1} (y_{j_2} - y_{i_2})^{n_1+1} \\
&= \prod_{0 \leq i_1 < j_1 \leq k_1} (x_{j_1} - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y_{k_2} - y_{i_2})^{k_1+1} \times \\
&\prod_{0 \leq i_1 < j_1 \leq n_1} (x_{j_1} - x_{i_1})^{k_2} \prod_{0 \leq i_2 < j_2 \leq k_2-1} (y_{j_2} - y_{i_2})^{n_1+1},
\end{aligned}$$

ce qui s'accorde avec la formule. D'où le résultat. ∎

On déduit du résultat précédent que tous les mineurs principaux de la matrice $\mathbf{D}_{(n_1,n_2)}$ sont non nuls, c'est à dire

Corollaire 2.2.1 $\mathbf{D}_{(n_1,n_2)}$ *est une matrice fortement inversible.*

Afin de pouvoir déterminer le polynôme d'interpolation P de manière récursive à partir de la formule 2.2.8, nous utiliserons l'identité matricielle de Sylvester 1.1.7, et certaines des propriétés des compléments de Schur.

On introduit les polynômes à deux variables définis par le complément de Schur comme suit

$$P_{(k_1,k_2)} = -\left(\mathbf{M}_{(k_1,k_2)}/\mathbf{D}_{(k_1,k_2)}\right), \tag{2.2.14}$$

pour tout $(k_1, k_2) \in \mathbb{N}_{(n_1,n_2)}$ où

$$\mathbf{M}_{(k_1,k_2)} = \left[\begin{array}{c|c} \mathbf{0} & \mathbf{U}_{(k_1,k_2)} \\ \hline \mathbf{r}_{(k_1,k_2)} & \mathbf{D}_{(k_1,k_2)} \end{array}\right],$$

avec

$$\mathbf{U}_{(k_1,k_2)} = [1\ x\ ...\ x^{n_1}\ y\ xy\ ...\ x^{n_1}y\ ...\ y^{n_2}\ xy^{k_2}\ ...\ x^{k_1}y^{k_2}] = [x^{j_1}y^{j_2}]_{(0,0)\leq(j_1,j_2)\leq(k_1,k_2)}\ ,$$
$$\mathbf{r}_{(k_1,k_2)} = [r_{(0,0)}\ r_{(1,0)}\quad r_{(n_1,0)}\ r_{(0,1)}\quad r_{(n_1,1)}...r_{(0,k_2)}...r_{(k_1,k_2)}]^T = [r_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)},$$

et $\mathbf{D}_{(k_1,k_2)}$ est la matrice définie par 2.2.9. Le polynôme $P_{(n_1,n_2)}$ étant exactement le polynôme d'interpolation solution de notre problème.

On va introduire aussi les polynômes $g_{(j_1,j_2),(k_1,k_2)}$ pour tout $(j_1,j_2),(k_1,k_2)$ de $\mathbb{N}_{(n_1,n_2)}$ vérifiant $(j_1,j_2)<(k_1,k_2)$ définis par le complément de Schur suivant

$$g_{(j_1,j_2),(k_1,k_2)} = \left(\left[\begin{array}{c|c} x^{k_1}y^{k_2} & \mathbf{U}_{(j_1,j_2)} \\ \hline [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(j_1,j_2)} & \mathbf{D}_{(j_1,j_2)} \end{array} \right] /\mathbf{D}_{(j_1,j_2)} \right), \qquad (2.2.15)$$

Alors et comme dans [59], on obtient

Théorème 2.2.2 *Pour tout $(k_1,k_2)\in\mathbb{N}_{(n_1,n_2)}$ on a*

$$P_{(k_1,k_2)} = P_{(k_1,k_2)^-} + \frac{r_{(k_1,k_2)} - P_{(k_1,k_2)^-}(z_{(k_1,k_2)})}{g_{(k_1,k_2)^-,(k_1,k_2)}(z_{(k_1,k_2)})} g_{(k_1,k_2)^-,(k_1,k_2)}. \qquad (2.2.16)$$

Et pour tout $(j_1,j_2)\in\mathbb{N}_{(n_1,n_2)}$ tel que $(0,0)\leq(j_1,j_2)<(k_1,k_2)$, on a

$$g_{(j_1,j_2),(k_1,k_2)} = g_{(j_1,j_2)^-,(k_1,k_2)} - \frac{g_{(j_1,j_2)^-,(k_1,k_2)}(z_{(j_1,j_2)})}{g_{(j_1,j_2)^-,(j_1,j_2)}(z_{(j_1,j_2)})} g_{(j_1,j_2)^-,(j_1,j_2)}, \qquad (2.2.17)$$

où $P_{(0,0)^-}$ coïncide avec le polynôme nul, et $g_{(0,0)^-,(k_1,k_2)} = x^{k_1}y^{k_2}$.

Preuve: D'abord remarquons qu'avec les conventions $P_{(0,0)^-}=\mathbf{0}$ et $g_{(0,0)^-,(k_1,k_2)} = x^{k_1}y^{k_2}$; les deux relations 2.2.16 et 2.2.17 sont vérifiées lorsque $(k_1,k_2)=(0,0)$. En effet, et par définition on a

$$\begin{aligned} P_{(0,0)} &= -\left(\left[\begin{array}{c|c} \mathbf{0} & \mathbf{U}_{(0,0)} \\ \hline \mathbf{r}_{(0,0)} & \mathbf{D}_{(0,0)} \end{array} \right] /\mathbf{D}_{(0,0)} \right) \\ &= -\left(-\mathbf{U}_{(0,0)}\ \mathbf{D}_{(0,0)}\ \mathbf{r}_{(0,0)}\right) \\ &= r_{(0,0)}. \end{aligned}$$

Puisque $\mathbf{U}_{(0,0)}$ est identifié à 1, $\mathbf{D}_{(0,0)} = 1$ et $\mathbf{r}_{(0,0)}$ est identifié à son unique élément $r_{(0,0)}$, 2.2.16 est vérifié lorsque $(k_1, k_2) = (0, 0)$. De même et par définition (2.2.15)

$$\begin{aligned} g_{(0,0),(k_1,k_2)} &= \left(\left[\begin{array}{c|c} x^{k_1}y^{k_2} & \mathbf{U}_{(0,0)} \\ \hline x^{k_1}_{(0,1)}x^{k_2}_{(0,2)} & \mathbf{D}_{(0,0)} \end{array} \right] / \mathbf{D}_{(0,0)} \right) \\ &= x^{k_1}y^{k_2} - z^{(k_1,k_2)}_{(0,0)}, \end{aligned}$$

ce qui permet de vérifier 2.2.17 lorsque $(j_1, j_2) = (0, 0)$ pour tout $(k_1, k_2) \in \mathbb{N}_{(n_1,n_2)}$ distinct de $(0, 0)$. On suppose, désormais que (k_1, k_2) est distinct de $(0, 0)$. Alors et toujours par définition

$$\begin{aligned} P_{(k_1,k_2)} &= -\left(\mathbf{M}_{(k_1,k_2)} / \mathbf{D}_{(k_1,k_2)} \right) \\ &= -\left(\left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{U}_{(k_1,k_2)^-} & x^{k_1}y^{k_2} \\ \hline \mathbf{r}_{(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} & C \\ \hline r_{(k_1,k_2)} & L & z^{(k_1,k_2)}_{(k_1,k_2)} \end{array} \right] / \mathbf{D}_{(k_1,k_2)} \right), \end{aligned}$$

où C et L désignent respectivement la matrice colonne et la matrice ligne

$$\begin{aligned} C &= [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)^-}, \\ L &= [z^{(j_1,j_2)}_{(k_1,k_2)}]_{(0,0)\leq(j_1,j_2)\leq(k_1,k_2)^-}, \end{aligned}$$

alors et en appliquant l'identité matricielle de Sylvester 1.1.7, nous obtenons

$$\begin{aligned} P_{(k_1,k_2)} = &-\left(\left[\begin{array}{c|c} \mathbf{0} & \mathbf{U}_{(k_1,k_2)^-} \\ \hline \mathbf{r}_{(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} \end{array} \right] / \mathbf{D}_{(k_1,k_2)^-} \right) + \\ &\left(\left[\begin{array}{c|c} \mathbf{U}_{(k_1,k_2)^-} & x^{k_1}y^{k_2} \\ \hline \mathbf{D}_{(k_1,k_2)^-} & C \end{array} \right] / \mathbf{D}_{(k_1,k_2)^-} \right) \left(\mathbf{D}_{(k_1,k_2)} / \mathbf{D}_{(k_1,k_2)^-} \right)^{-1} \left(\left[\begin{array}{c|c} \mathbf{r}_{(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} \\ \hline r_{(k_1,k_2)} & L \end{array} \right] / \mathbf{D}_{(k_1,k_2)^-} \right), \end{aligned}$$

or et par définition de $P_{(k_1,k_2)}$ on a

$$\left(\left[\begin{array}{c|c} \mathbf{0} & \mathbf{U}_{(k_1,k_2)^-} \\ \hline \mathbf{r}_{(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} \end{array} \right] / \mathbf{D}_{(k_1,k_2)^-} \right) = P_{(k_1,k_2)^-}.$$

D'un autre côté, et par définition de $g_{(k_1,k_2)^-,(k_1,k_2)}$ 2.2.15 et en appliquant la propriété 1.1.2, on

obtient

$$g_{(k_1,k_2)^-,(k_1,k_2)} = \left(\left[\begin{array}{c|c} x^{k_1}y^{k_2} & \mathbf{U}_{(k_1,k_2)^-} \\ \hline [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} \end{array} \right] /\mathbf{D}_{(k_1,k_2)^-} \right) \tag{2.2.18}$$

$$= \left(\left[\begin{array}{c|c} \mathbf{U}_{(k_1,k_2)^-} & x^{k_1}y^{k_2} \\ \hline \mathbf{D}_{(k_1,k_2)^-} & C \end{array} \right] /\mathbf{D}_{(k_1,k_2)^-} \right), \tag{2.2.19}$$

puis, et en évaluant $g_{(k_1,k_2)^-,(k_1,k_2)}$ en $(x_{(k_1,1),(k_2,2)})$ et en appliquant 1.1.2, on obtient

$$g_{(k_1,k_2)^-,(k_1,k_2)}(x_{(k_1,1)}, x_{(k_2,2)}) = \left(\left[\begin{array}{c|c} z^{(k_1,k_2)}_{(k_1,k_2)} & L \\ \hline [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} \end{array} \right] /\mathbf{D}_{(k_1,k_2)^-} \right) \tag{2.2.20}$$

$$= \left(\left[\begin{array}{c|c} \mathbf{D}_{(k_1,k_2)^-} & C \\ \hline L & z^{(k_1,k_2)}_{(k_1,k_2)} \end{array} \right] /\mathbf{D}_{(k_1,k_2)^-} \right)$$

$$= \left(\mathbf{D}_{(k_1,k_2)} / \mathbf{D}_{(k_1,k_2)^-} \right),$$

enfin, et en évaluant $P_{(k_1,k_2)^-}$ en $(x_{(k_1,1),(k_2,2)})$, on obtient

$$P_{(k_1,k_2)^-}(x_{(k_1,1),(k_2,2)}) = -\left(\left[\begin{array}{c|c} 0 & L \\ \hline C & \mathbf{D}_{(k_1,k_2)^-} \end{array} \right] /\mathbf{D}_{(k_1,k_2)} \right)$$

$$= L\mathbf{D}^{-1}_{(k_1,k_2)^-}C,$$

il s'en suit, et par définition du complément de Schur, que

$$\left(\left[\begin{array}{c|c} C & \mathbf{D}_{(k_1,k_2)^-} \\ \hline r_{(k_1,k_2)} & L \end{array} \right] /\mathbf{D}_{(k_1,k_2)^-} \right) = r_{(k_1,k_2)} - L\mathbf{D}^{-1}_{(k_1,k_2)^-}C \tag{2.2.21}$$

$$= r_{(k_1,k_2)} - P_{(k_1,k_2)^-}(z_{(k_1,k_2)}).$$

Ainsi, et en combinant 2.2.18, 2.2.20 et 2.2.21, on déduit 2.2.16.

En appliquant l'identité matricielle de Sylvester 1.1.7 au complément de Schur définissant $g_{(k_1,k_2),(j_1,j_2)}$, et par un raisonnement similaire au précédent, on déduit 2.2.17. ∎

2.2.3 le RBVPIA version 1

On déduit des formules 2.2.16 et 2.2.17 du théorème précédent, l'algorithme nommé Recursive BiVariate Polynomial Interpolation Algorithm (RBVPIA) permettant de déterminer récursivement le polynôme d'interpolation $P_{(n_1,n_2)}$.

Algorithm -2.1 : RBVPIA

Initialization : $P_{(0,0)^-} = \mathbf{0}$

for $(k_1, k_2) = (0,0)$ to (n_1, n_2) :

$g_{(0,0)^-,(k_1,k_2)} = x^{k_1} y^{k_2}$

for $(i_1, i_2) = (0,0)$ to $(k_1, k_2)^-$:

$$g_{(i_1,i_2),(k_1,k_2)} = g_{(i_1,i_2)^-,(k_1,k_2)} - \frac{g_{(i_1,i_2)^-,(k_1,k_2)}\left(z_{(i_1,i_2)}\right)}{g_{(i_1,i_2)^-,(i_1,i_2)}\left(z_{(i_1,i_2)}\right)} g_{(i_1,i_2)^-,(i_1,i_2)}$$

end

$$P_{(k_1,k_2)} = P_{(k_1,k_2)^-} + \frac{r_{(k_1,k_2)} - P_{(k_1,k_2)^-}(z_{(k_1,k_2)})}{g_{(k_1,k_2)^-,(k_1,k_2)}(z_{(k_1,k_2)})} g_{(k_1,k_2)^-,(k_1,k_2)}$$

end

return $P_{(n_1,n_2)}$.

2.2.4 Une version simplifiée du RBVPIA

Nous allons maintenant donner une version simplifiée du RBVPIA, qui est facile à mettre en oeuvre, ne nécessite pas de stockage et effectue moins de calculs.

En conservant les notations et les conventions précédentes, on peut remarquer que

Lemme 2.2.4 *Pour tout* $(k_1, k_2) \in \mathbb{N}_{(n_1,n_2)}$

$$g_{(k_1,k_2)^-,(k_1,k_2)} = \prod_{i_1=0}^{k_1-1} (x - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y - y_{i_2}), \tag{2.2.22}$$

avec la convention $g_{(0,0)^-,(0,0)} = 1$.

Preuve: Il suffit de remarquer qu'en appliquant 1.1.2, on obtient

$$g_{(k_1,k_2)^-,(k_1,k_2)} = \left(\left[\begin{array}{c|c} x^{k_1}y^{k_2} & \mathbf{U}_{(k_1,k_2)^-} \\ \hline [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)^-} & \mathbf{D}_{(k_1,k_2)^-} \end{array} \right] / \mathbf{D}_{(k_1,k_2)^-} \right)$$
$$= \left(\left[\begin{array}{c|c} \mathbf{D}_{(k_1,k_2)^-} & [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)^-} \\ \hline \mathbf{U}_{(k_1,k_2)^-} & x^{k_1}y^{k_2} \end{array} \right] / \mathbf{D}_{(k_1,k_2)^-} \right),$$

et en comparant avec le polynôme W défini par 2.2.11

$$\det \left[\begin{array}{c|c} \mathbf{D}_{(k_1,k_2)^-} & [z^{(k_1,k_2)}_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)^-} \\ \hline \mathbf{U}_{(k_1,k_2)^-} & x^{k_1}y^{k_2} \end{array} \right] = W.$$

Il s'en suit en appliquant 2 que

$$g_{(k_1,k_2)^-,(k_1,k_2)} = \frac{W}{\det \mathbf{D}_{(k_1,k_2)^-}}$$
$$= \prod_{i_1=0}^{k_1-1} (x - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y - y_{i_2}).$$

■

Corollaire 2.2.2 *Les polynômes d'interpolation $P_{(k_1,k_2)}$ définis ci-dessus, vérifient la propriété suivante*

$$P_{(k_1,k_2)} = P_{(k_1,k_2)^-} + \alpha_{(k_1,k_2)} \prod_{i_1=0}^{k_1-1} (x - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y - y_{i_2}),$$

où $\alpha_{(k_1,k_2)}$ est le scalaire

$$\alpha_{(k_1,k_2)} = \frac{r_{(k_1,k_2)} - P_{(k_1,k_2)^-}(z_{(k_1,k_2)})}{\prod_{i_1=0}^{k_1-1} (x_{k_1} - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y_{k_2} - y_{i_2})}.$$

On en déduit la version simplifiée du RBVPIA.

Algorithm -2.2 : RBVPIA (simplified version)

Initialization : $P = 0$, $P_1 = 1$, $P_2 = 1$

for $(k_1, k_2) = (0, 0)$ to (n_1, n_2) :

if $k_1 > 0$:

$P_1 = (x - x_{k_1-1}) \times P_1$

elif $k_2 > 0$:

$P_1 = 1$, $P_2 = (y - y_{k_2-1}) \times P_2$

end

$$\alpha = \frac{r_{(k_1,k_2)} - P\left(z_{(k_1,k_2)}\right)}{P_1(x_{k_1})P_2(y_{k_2})}$$

$P = P + \alpha P_1 P_2$

end

return P.

Nous donnons maintenant quelques remarques intéressantes sur l'algorithme.

Remarque 2.2.2 .

1. *Si nous choisissons l'ordre 2.2.2 de la grille Z correspondant au parcours, figure -2.2, l'algorithme -2.2 doit être remplacé par l'algorithme -2.3 , donné ci-dessous.*

2. *Les deux algorithmes -2.2 et -2.3, ne nécessitent pas de stockage.*

3. *Si nous ajoutons une abscisse x_{n_1+1}, figure -2.3, ci-dessous (respectivement une ordonnée y_{n_2+1}, figure -2.4 ci-dessous), alors nous conservons les calculs précédents pour calculer l'interpolant $P_{(n_1,n_2)}$ et on continue avec l'algorithme -2.3 (respectivement l'algorithme -2.2) pour calculer les polynômes $P_{(n_1+1,0)}, P_{(n_1+1,1)}, ..., P_{(n_1+1,n_2)}$ (respectivement $P_{(0,n_2+1)}, P_{(1,n_2+1)}, ..., P_{(n_1,n_2+1)}$).*

4. *De même si on enlève dans la grille Z une abscisse x_{i_1} (respectivement une ordonnée y_{i_2}) alors on applique l'algorithme -2.3 (respectivement l'algorithme -2.2),*

Algorithm -2.3 : RBVPIA (simplified version with the order (2.2.2))

Initialization : $P = 0$, $P_1 = 1$, $P_2 = 1$

for $(k_1, k_2) = (0, 0)$ to (n_1, n_2) :

if $k_2 > 0$:

$P_2 = (y - y_{k_2-1}) \times P_2$

elif $k_1 > 0$:

$P_2 = 1$, $P_1 = (x - x_{k_1-1}) \times P_1$

end

$$\alpha = \frac{r_{(k_1,k_2)} - P\left(z_{(k_1,k_2)}\right)}{P_1(x_{k_1})P_2(y_{k_2})}$$

$$P = P + \alpha P_1 P_2$$

end

return P.

les calculs effectués avant l'étape i_1 (respectivement i_2) sont maintenus et les nouveaux calculs commencent à partir de l'étape $i_1 + 1$ (respectivement $i_2 + 1$).

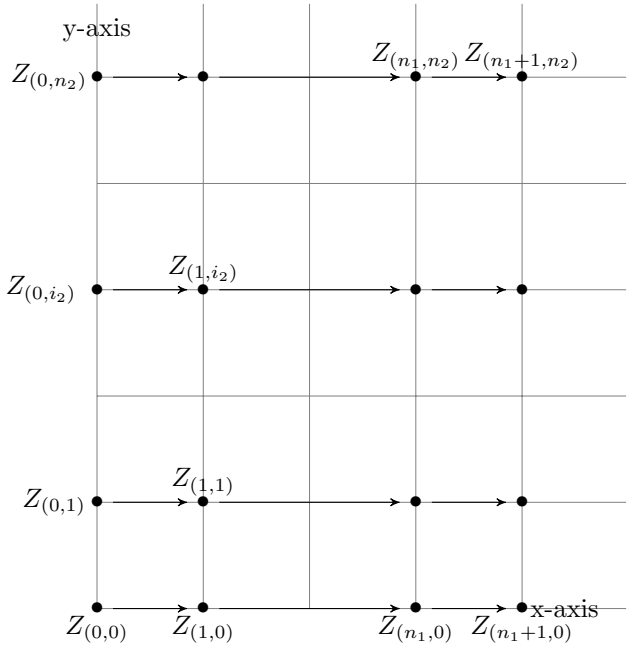

Figure -2.3 – La grille avec ajout d'une abscisse.

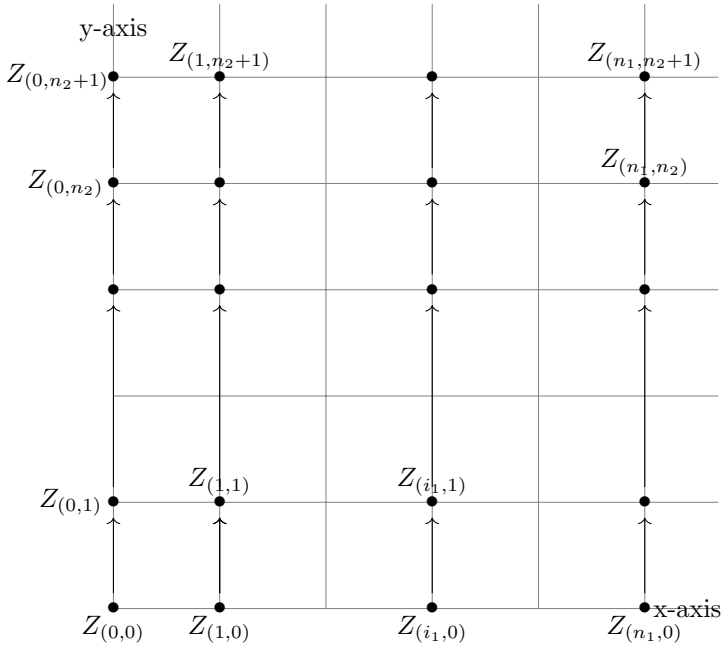

Figure -2.4 – La grille avec ajout d'une ordonnée.

2.2.5 Propriétés du RBVPIA

On conserve les notations et les résultats des sections précédentes. En outre, pour tout $(k_1, k_2) \in \mathbb{N}_{(n_1,n_2)}$, on désigne par $\Pi_{(k_1,k_2)}$ le sous espace vectoriel de $\Pi_{(n_1,n_1)}$ engendré par la famille de polynômes $\mathbf{U}_{(k_1,k_2)} = [x^{j_1}y^{j_2}]_{(0,0)\leq(j_1,j_2)\leq(k_1,k_2)}$ et on pose

$$G_{(k_1,k_2)} = g_{(k_1,k_2)^-,(k_1,k_2)}$$
$$\mathbf{G}_{(k_1,k_2)} = [G_{(0,0)}...G_{(n_1,0)}...G_{(0,k_2)}...G_{(k_1,k_2)}],$$

et d'après 2.2.22

$$G_{(k_1,k_2)} = \prod_{i_1=0}^{k_1-1}(x - x_{i_1}) \prod_{i_2=0}^{k_2-1}(y - y_{i_2}).$$

On obtient

Proposition 2.2.3 *1. $G_{(k_1,k_2)}$ est une combinaison linéaire des monômes $x^{j_1}y^{j_2}$ tels que $0 \leq j_1 \leq k_1$ et $0 \leq j_2 \leq k_2$, le coefficient du terme en $x^{k_1}y^{k_2}$ est 1 et les autres coefficients peuvent être obtenus grâce aux fonctions symétriques élémentaires de $(x_0, ..., x_{k_1-1})$ et celles de $(y_0, ..., y_{k_2-1})$.*

2. *La famille $\mathbf{G}_{(n_1,n_2)} = [G_{(k_1,k_2)}(x, y)]_{(0,0)\leq(k_1,k_2)\leq(n_1,n_2)}$ est une base de $\Pi_{(n_1,n_2)}$.*
3. *Pour tout $(k_1, k_2) \in \mathbb{N}_{(n_1,n_2)}$, la famille des polynômes*

$$\mathbf{G}_{(k_1,k_2)} = [G_{(j_1,j_2)}(x, y)]_{(0,0)\leq(j_1,j_2)\leq(k_1,k_2)},$$

est une base de $\Pi_{(k_1,k_2)}$.

4. *La matrice de passage de la base $\mathbf{U}_{(k_1,k_2)}$ à la nouvelle base $\mathbf{G}_{(k_1,k_2)}$ est une matrice triangulaire supérieure d'ordre (k_1, k_2) et de diagonale unité.*

Une autre propriété intéressante qu'on déduit aisément du RBVPIA est la suivante.

Proposition 2.2.4 *Pour tout $(k_1, k_2) \in \mathbb{N}_{(n_1,n_2)}$, $P_{(k_1,k_2)}$ est l'unique polynôme de $\Pi_{(k_1,k_2)}$ vérifiant les conditions d'interpolation suivantes*

$$P_{(k_1,k_2)}\left(z_{(i_1,i_2)}\right) = r_{(i_1,i_2)}, \text{ pour tout } (0, 0) \leq (i_1, i_2) \leq (k_1, k_2).$$

Remarque 2.2.3 *Il ne faut pas confondre le polynôme d'interpolation $P_{(k_1,k_2)}$ aux points $z_{(i_1,i_2)}$ tels que $(0,0) \leq (i_1,i_2) \leq (k_1,k_2)$ avec le polynôme d'interpolation de Lagrange aux points $z_{(i_1,i_2)}$ tels que $0 \leq i_1 \leq k_1$ et $0 \leq i_2 \leq k_2$. Puisque si $k_2 \geq 1$ et $k_1 \neq n_1$ alors $P_{(k_1,k_2)}$ ne satisfait pas la condition $\deg_x P \leq k_1$ et l'ensemble des noeuds correspondant n'est pas un produit cartésien.*

Pour tout $(i_1,i_2),(k_1,k_2) \in \mathbb{N}_{(n_1,n_2)}$, on définit l'opérateur linéaire $C^*_{(i_1,i_2)}$ par

$$C^*_{(i_1,i_2)}(P) = P\left(z_{(i_1,i_2)}\right).$$

Pour tout $P \in \Pi_{n_1,n_2}$, on pose

$$\mathbf{C}^*_{(k_1,k_2)} = [C^*_{(0,0)} \cdots C^*_{(n_1,0)} \cdots C^*_{(0,k_2)} \cdots C^*_{(k_1,k_2)}]^T,$$

de sorte que pour tout $P \in \Pi_{n_1,n_2}$, on ait

$$\mathbf{C}^*_{(k_1,k_2)}P = [C^*_{(0,0)}(P) \cdots C^*_{(n_1,0)}(P) \cdots C^*_{(0,k_2)}(P) \cdots C^*_{(k_1,k_2)}(P)]^T.$$

Plus généralement, pour tout $(j_1,j_2) \in \mathbb{N}_{(n_1,n_2)}$, $\mathbf{C}^*_{(k_1,k_2)}$ s'applique à toute matrice ligne $M = [M_{(l_1,l_2)}]_{(0,0)\leq(l_1,l_2)\leq(j_1,j_2)}$ dont les coefficients sont des polynômes à deux indéterminées, le résultat étant alors une matrice de taille $(k_1,k_2)\times(j_1,j_2)$. On obtient

$$\mathbf{C}^*_{(k_1,k_2)}\mathbf{U}_{(k_1,k_2)} = [z^{(j_1,j_2)}_{(i_1,i_2)}]_{(0,0)\leq(i_1,i_2),(j_1,j_2)\leq(k_1,k_2)} = \mathbf{D}_{(k_1,k_2)}.$$

On introduit alors la matrice

$$\mathbf{S}_{(k_1,k_2)} = \mathbf{U}_{(k_1,k_2)}\mathbf{D}^{-1}_{(k_1,k_2)}\mathbf{C}^*_{(k_1,k_2)},$$

considérée comme un opérateur linéaire sur Π_{n_1,n_2}. On obtient

Proposition 2.2.5 *Pour tout $(0,0) \leq (k_1,k_2) \leq (n_1,n_2)$ on a*

1. $\mathbf{S}^2_{(k_1,k_2)} = \mathbf{S}_{(k_1,k_2)}$.
2. $\mathbf{S}_{(k_1,k_2)}$ *est une projection oblique sur* $\Pi_{(k_1,k_2)}$ *dans la direction de* $vect(G_{(j_1,j_2)})_{(k_1,k_2)<(j_1,j_2)\leq(n_1,n_2)}$.

3. $\mathbf{S}_{(k_1,k_2)} \circ \mathbf{S}_{(j_1,j_2)} = \mathbf{S}_{(j_1,j_2)} \circ \mathbf{S}_{(k_1,k_2)} = \mathbf{S}_{(k_1,k_2)}$ *pour tout* $(j_1,j_2) \in \mathbb{N}_{(n_1,n_2)}$; $(k_1,k_2) \leq (j_1,j_2)$.

4. $(\mathbf{I} - \mathbf{S}_{(k_1,k_2)})(G_{j_1,j_2}) = G_{(j_1,j_2)}$ *pour tout* $(j_1,j_2) \in \mathbb{N}_{(n_1,n_2)}$; $(k_1,k_2) \leq (j_1,j_2)$; *où* $\mathbf{I}$ *désigne l'application identité de* $\Pi_{(n_1,n_2)}$.

Preuve: Un calcul matriciel direct montre le premier point. Il s'avère que $\mathbf{S}_{(k_1,k_2)}$ est une projection oblique et comme

$$\begin{aligned}\mathbf{S}_{(k_1,k_2)}\mathbf{U}_{(k_1,k_2)} &= \mathbf{U}_{(k_1,k_2)}\mathbf{D}^{-1}_{(k_1,k_2)}\mathbf{D}_{(k_1,k_2)} \\ &= \mathbf{U}_{(k_1,k_2)},\end{aligned}$$

on déduit que les polynômes de $\Pi_{(k_1,k_2)}$ sont invariants par $\mathbf{S}_{(k_1,k_2)}$. D'autre part, pour tout $(j_1,j_2) \in \mathbb{N}_{(n_1,n_2)},\ (k_1,k_2) < (j_1,j_2)$ on a : pour tout $(0,0) \leq (i_1,i_2) \leq (k_1,k_2)$, soit $i_2 < j_2$ ou bien $i_2 = j_2$ et $i_1 < j_1$, ce qui entraîne, en appliquant 2.2.22, que

$$C^*_{(i_1,i_2)}(G_{(j_1,j_2)}) = 0.$$

Ce qui montre que $\mathbf{S}_{(k_1,k_2)}(G_{(j_1,j_2)}) = 0$. Il s'en suit que $\mathbf{S}_{(k_1,k_2)}$ est bien la projection oblique sur $\Pi_{(k_1,k_2)}$ dans la direction du sous espace supplémentaire $vect(G_{(j_1,j_2)})_{(k_1,k_2)<(j_1,j_2)\leq(n_1,n_2)}$. Pour le troisème point, il suffit de remarquer que $\mathbf{S}_{(k_1,k_2)} \circ \mathbf{S}_{(j_1,j_2)}$ laisse invariant les élément de $\mathbf{U}_{(k_1,k_2)}$ et annule $G_{(i_1,i_2)}$ lorsque $(i_1,i_2) < (k_1,k_2)$. Pour le dernier point il suffit de remarquer que $\mathbf{I} - \mathbf{S}_{(k_1,k_2)}$ est la projection oblique conjuguée de $\mathbf{S}_{(k_1,k_2)}$. ■

D'autre part, et à partir de la définition 2.2.22, on obtient aussi

Proposition 2.2.6 1. $\mathbf{S}_{(k_1,k_2)} = \mathbf{G}_{(k_1,k_2)} \left[\mathbf{C}^*_{(k_1,k_2)}\mathbf{G}_{(k_1,k_2)}\right]^{-1} \mathbf{C}^*_{(k_1,k_2)}$.

2. $\mathbf{C}^*_{(k_1,k_2)}\mathbf{G}_{(k_1,k_2)}$ *est une matrice triangulaire inférieure et pour tout* $(0,0) \leq (j_1,j_2) \leq (q_1,q_2) \leq (k_1,k_2)$, *le coefficient d'indice* $((q_1,q_2),(j_1,j_2))$ *est égal à*

$$\prod_{i_1=0}^{j_1-1}(x_{q_1} - x_{i_1}) \prod_{i_2=0}^{j_2-1}(y_{q_2} - y_{i_2}).$$

Preuve: Pour le premier point on raisonne comme dans la preuve de la proposition précédente en remplaçant $\mathbf{U}_{(k_1,k_2)}$ par $\mathbf{G}_{(k_1,k_2)}$. Le second point est dû à l'expression 2.2.22 de $G_{(j_1,j_2)}$. ■

Corollaire 2.2.3 *Pour tout* (k_1, k_2), *en posant* $\omega_{(k_1,k_2)} = [\omega_{(0,0)}, ..., \omega_{(n_1,0)}, ..., \omega_{(k_1,k_2)}]^T$ *l'unique solution du système de Cramer triangulaire*

$$\mathbf{C}^*_{(k_1,k_2)}\mathbf{G}_{(k_1,k_2)}\omega_{(k_1,k_2)} = \mathbf{r}_{(k_1,k_2)},$$

où, on rappelle que

$$\mathbf{r}_{(k_1,k_2)} = [r_{(i_1,i_2)}]^T_{(0,0)\leq(i_1,i_2)\leq(k_1,k_2)},$$

on obtient l'expression du polynôme interpolant $P_{(k_1,k_2)}$ *dans la base* $\mathbf{G}_{(k_1,k_2)}$

$$P_{(k_1,k_2)} = \sum_{(0,0)\leq(j_1,j_2)\leq(k_1,k_2)} \omega_{(j_1,j_2)} G_{(k_1,k_2)}.$$

Preuve: Il suffit de remarquer que

$$\mathbf{S}_{(k_1,k_2)}P_{(k_1,k_2)} = \mathbf{G}_{(k_1,k_2)}\left[\mathbf{C}^*_{(k_1,k_2)}\mathbf{G}_{(k_1,k_2)}\right]^{-1}\mathbf{C}^*_{(k_1,k_2)}P_{(k_1,k_2)},$$

et que

$$\mathbf{C}^*_{(k_1,k_2)}P_{(k_1,k_2)} = \mathbf{r}_{(k_1,k_2)},$$

d'après la proposition 2.2.4. ■

2.3 Généralisation : Le RMVPIA

Dans cette section, nous rappelons d'abord le problème d'interpolation polynomiale à p variables tel que $p > 2$. Puis nous donnons une généralisation du RBVPIA : Recursive MultiVariate Polynomial Interpolation Algorithm (RMVPIA) . Une version simplifiée de cet algorithme sera également donnée.

2.3.1 Formulation du problème

Soient $n_1, ..., n_p$ des entiers naturels et

— $\mathbb{N}_{(n_1,...,n_p)} = \{(i_1, ..., i_p) \in \mathbb{N}^p;\ 0 \leq i_k \leq n_k \text{ pour } k \in \{1, ..., p\}\}$ l'ensemble des indices.

— $Z = \{z_{(i_1,...,i_p)}\ ;\ (i_1, ..., i_p) \in \mathbb{N}_{(n_1,...,n_p)}\}$ un ensemble formant une grille dans $\mathbb{K}^p$, ce qui signifie que

$$z_{(i_1,...,i_p)} = (x_{(i_1,1)}, ..., x_{(i_p,p)}),$$

avec $\mathbf{x}_k = [x_{(0,k)}, ..., x_{(i_k,k)}, ..., x_{(n_k,k)}]$ une subdivision du $k^{ème}$ axe formée de n_k+1 scalaires distincts deux à deux, pour chaque $k \in \{1, ..., p\}$.

— $R = (r_{(i_1,...,i_p)}\ ;\ (i_1, ..., i_p) \in \mathbb{N}_{(n_1,...,n_p)})$ un ensemble de valeurs quelconques dans $\mathbb{K}$.

— $\Pi_{(n_1,...,n_p)} = \text{vect}(x_1^{j_1} x_2^{j_2} ... x_p^{j_p}; (j_1, ..., j_p) \in \mathbb{N}_{(n_1,...,n_p)})$ l'espace d'interpolation.

Le problème d'interpolation polynomiale multivariée de Lagrange (1.1.1) consiste à trouver le polynôme d'interpolation $P_{(n_1,...,n_p)} \in \Pi_{(n_1,...,n_p)}$, qui vérifie les conditions d'interpolation suivantes

$$P_{(n_1,...,n_p)}(z_{(i_1,...,i_p)}) = r_{(i_1,...,i_p)}, \quad \text{pour} \quad (i_1, ..., i_p) \in \mathbb{N}_{(n_1,...,n_p)}\ . \tag{2.3.1}$$

2.3.2 Le RMVPIA

La conception de cet algorithme repose essentiellement, comme dans le cas à deux variables, sur l'établissement d'un ordre total dans $\mathbb{N}_{(n_1,...,n_p)}$ qu'on défini comme suit. Étant donnés deux multi-indices $i = (i_1, i_2, ..., i_p)$ et $j = (j_1, j_2, ..., j_p)$ de $\mathbb{N}_{(n_1,...,n_p)}$ distincts, on considère $m(i,j) = \max\{k \in \{1, ..., p\} \ ; \ i_k \neq j_k\}$ et on impose

$$\begin{aligned}(i_1, i_2, ..., i_p) < (j_1, j_2, ..., j_p) &\Longleftrightarrow i_{m(i,j)} < j_{m(i,j)} \\ &\Longleftrightarrow (i_p < j_p) \text{ ou bien } (i_p = j_p \text{ et } i_{p-1} < j_{p-1}) \\ &\text{ou bien...ou bien } (i_p = j_p, ..., i_2 = j_2 \text{ et } i_1 < j_1),\end{aligned}$$

grâce à cet ordre on définit le prédécesseur : $(i_1, i_2, ..., i_p)^-$ de tout élément $(i_1, i_2, ..., i_p)$ dans $\mathbb{N}_{(n_1,...,n_p)}$ distinct du premier $(0, ..., 0)$ comme suit

$$(i_1, i_2, ..., i_p)^- := (n_1, n_2, ..., n_k, i_{k+1} - 1, i_{k+2}, ..., i_p),$$

où k est tel que $i_1 = i_2 = ... = i_k = 0$ et $i_{k+1} \neq 0$. Avec cet ordre, et par analogie avec le cas $p = 2$, le RMVPIA en version compacte devient

Algorithm -2.4 : RMVPIA

Initialization : $P_{(0,...,0)^-} = \mathbf{0}$

for $(k_1, k_2, ..., k_p) = (0, ..., 0)$ to $(n_1, n_2, ..., n_p)$:

$g_{(0,...,0)^-,(k_1,...,k_p)} = x^{k_1}...X_p^{k_p}$

for $(i_1, ..., i_p) = (0, ..., 0)$ to $(k_1, ..., k_p)^-$:

$$g_{(i_1,...,i_p),(k_1,...,k_p)} = g_{(i_1,...,i_p)^-,(k_1,...,k_p)} - \frac{g_{(i_1,...,i_p)^-,(k_1,...,k_p)}\left(z_{(i_1,i_2,...,i_p)}\right)}{g_{(i_1,...,i_p)^-,(i_1,...,i_p)}\left(z_{(i_1,i_2,...,i_p)}\right)} g_{(i_1,...,i_p)^-,(i_1,...,i_p)}$$

end

$$P_{(k_1,...,k_p)} = P_{(k_1,...,k_p)^-} + \frac{r_{(k_1,...,k_p)} - P_{(k_1,...,k_p)^-}\left(z_{(k_1,...,k_p)}\right)}{g_{(k_1,...,k_p)^-,(k_1,...,k_p)}\left(z_{(k_1,...,k_p)}\right)} g_{(k_1,...,k_p)^-,(k_1,...,k_p)}$$

end

return $P_{(n_1,...,n_p)}$.

Maintenant, et comme pour le cas bivarié, nous établissons les résultats suivants pour donner une version simplifiée de l'algorithme 4.

Lemme 2.3.1 .

Pour $(k_1, ..., k_p) \in \mathbb{N}_{(n_1,...,n_p)}$

$$g_{(k_1,...,k_p)^-,(k_1,...,k_p)} = \prod_{s=1}^{p} \left(\prod_{i_s=0}^{k_s-1} (x_s - x_{(i_s,s)}) \right), \tag{2.3.2}$$

avec la convention $g_{(0,...,0)^-,(0,...,0)} = 1$.

Ce qui permet d'obtenir la relation récursive suivante.

Corollaire 2.3.1 .

Pour $(k_1, ..., k_p) \in \mathbb{N}_{(n_1,...,n_p)}$

$$P_{(k_1,...,k_p)} = P_{(k_1,...,k_p)^-} + \alpha_{(k_1,...,k_p)} \prod_{s=1}^{p} \left(\prod_{i_s=0}^{k_s-1} (x_s - x_{(i_s,s)}) \right),$$

où $\alpha_{(k_1,...,k_p)}$ *est le scalaire*

$$\alpha_{(k_1,...,k_p)} = \frac{r_{(k_1,...,k_p)} - P_{(k_1,...,k_p)^-}(z_{(k_1,...,k_p)})}{\prod_{s=1}^{p}\left(\prod_{i_s=0}^{k_s-1}(x_{(k_s,s)} - x_{(i_s,s)})\right)},$$

avec la convention : $P_{(0,...,0)^-}(x_1, ..., x_p)$ *est le polynôme nul.*

D'où la version simplifiée suivante du RMVPIA dans le cas général (p quelconque)

Algorithm -2.5 : RMVPIA (simplified version)

Initialization : $P = 0$, $(P_1, P_2, ..., P_p) = (1, ..., 1)$;

for $(k_1, k_2, ..., k_p) = (0, ..., 0)$ to $(n_1, n_2, ..., n_p)$:

$s = 1$;

while $s < p + 1$ and $k_s = 0$ do

$s = s + 1$;

end

if $s < p + 1$ and $k_s > 0$ then

for $i = 1$ to $s - 1$ do

$P_i = 1$

end

$P_s = P_s \times \left(x_s - x_{(k_s-1,s)}\right)$

end

$$\alpha = \frac{r_{(k_1,k_2,...,k_p)} - P\left(z_{(k_1,k_2,...,k_p)}\right)}{P_1\left(x_{(k_1,1)}\right) \times P_2\left(x_{(k_2,2)}\right) \cdots P_p\left(x_{(k_p,p)}\right)} ;$$

$$P = P + \alpha P_1 \times P_2 \cdots P_p$$

end

return P.

Remarque 2.3.1 *Comme dans le cas bivarié, le RMVPIA ne nécessite aucun stockage. De plus, en ajoutant ou en supprimant une coordonnée dans la grille, on conserve tout ou une partie des calculs précédents.*

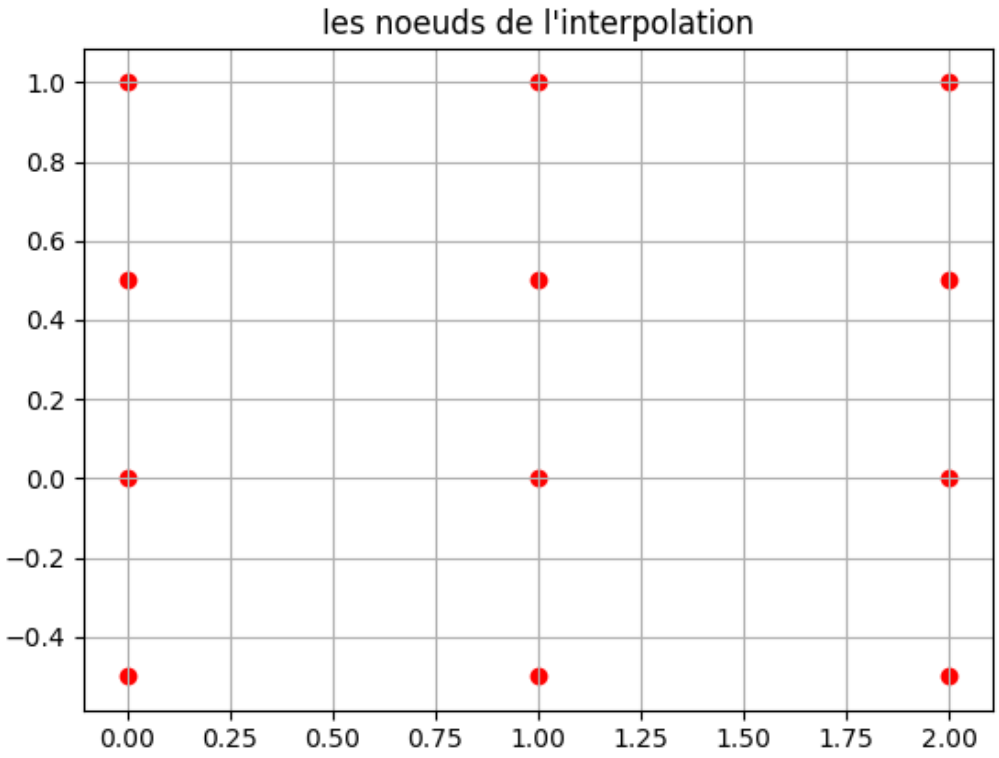

FIGURE -2.5 – L'ensemble des noeuds est une grille.

2.4 Exemples

Dans cette section, nous donnons un exemple pour tester le RBVPIA et un autre pour le RMVPIA où $p = 3$. Ces exemples sont donnés en arithmétique exacte en utilisant les versions simplifiées.

Exemple 1

Dans cet exemple, figure -2.5, nous prenons $n_1 = 2$, $n_2 = 3$ et nous choisissons

$$Z = \{0, 1, 2\} \times \{0, 1, -\frac{1}{2}, \frac{1}{2}\}, \ R = (1, 0, -2, -1, 1, \frac{1}{2}, -1, 1, 0, \frac{3}{2}, \frac{7}{3}, -4).$$

En appliquant le RBVPIA, nous obtenons le polynôme d'interpolation associé

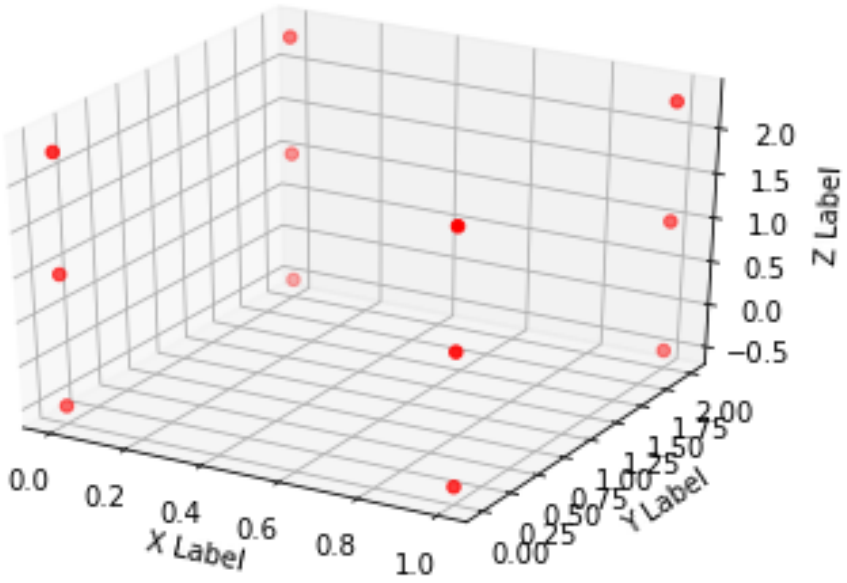

FIGURE -2.6 – L'ensemble des noeuds est une grille de $\mathbb{R}^3$.

$$\begin{aligned} P_{(2,3)} =& 1 - \frac{1}{2}x - \frac{1}{2}x^2 + 3y + \frac{71}{12}xy - \frac{21}{4}x^2y - 3y^2 \\ &+ \frac{107}{6}xy^2 - \frac{49}{6}x^2y^2 - 2y^3 - 20xy^3 + \frac{38}{3}x^2y^3. \end{aligned}$$

Exemple 2

Dans cet exemple, figure -2.6, nous prenons $n_1 = n_2 = 1, n_3 = 2$ et nous choisissons

$$Z = \{0,1\} \times \{2,0\} \times \{1, -\frac{1}{2}, \frac{7}{3}\},\ R = (1,0,-2,-1,1,\frac{1}{2},-1,1,\frac{22}{7},0,\frac{9}{2},-3).$$

Ainsi, en appliquant le RMVPIA, nous obtenons le polynôme d'interpolation associé

$$\begin{aligned} P_{(1,1,2)} =& \frac{-943}{408} + \frac{1091}{408}x + \frac{8647}{5712}y - \frac{8899}{5712}xy + \frac{139}{408}xz + \frac{3887}{5712}yz \\ &- \frac{671}{408}z - \frac{1283}{5712}xyz + \frac{133}{68}z^2 - \frac{137}{68}xz^2 - \frac{661}{952}yz^2 + \frac{745}{952}xyz^2. \end{aligned}$$

2.5 Annexe : Code Python du RMVPIA

Dans ce paragraphe, on donne le code Python des algorithmes vus dans le présent chapitre.

Commençons avec les modules scientifiques du langage Python employés :

```
#### Imporation des modules
from sympy import *
import numpy as np
from fractions import Fraction
X, Y, Z = symbols('X Y Z')
init_printing()
```

Les fonctions ci-dessous sont des fonctions auxiliaires employées dans les algorithmes RBVPIA et RMVPIA :

```
########## fonctions utiles #################

def suivant(k,In): # le suivant  dans I(n1,.....np)
    p = len(k)
    zero = [0 for i in range(p)]
    for j in range(p):
        if k[j] < In[j]:
                k[ : j] = zero[ : j]
                k[j] += 1
                return k
    return False

def vecteur_indet(p): # vecteur des inconnues X1...Xp
    X  =  ['X%d'%i for i in range(p)]
    for i in range(p):
        X[i] = symbols('X%d'%i)
    return X

def initialiser(X): #  les polynomes P1... Pp
    p = len(X)
    P = [0 for i in range(p)]
    for i in range(p):
        P[i] = 1 + 0*X[i]
    return P
```

Voici le code du RBVPIA ($p = 2$) (version simplifiée) :

```
### Le RBVPIA

def RBVPIA(x,y,v_interp):
    n1, n2 = len(x) - 1, len(y) - 1
    def r(k1, k2):
        d = k2*(n1 + 1) + k1
        return v_interp[d]
    P = 0*X + r(0,0)
    Q, T = 1+ 0*X, 1 + 0*Y
    In = [n1, n2]
    k = [1 , 0]
    while(k):
        k1, k2 = k[0], k[1]
        if k1> 0:
            Q = (X - x[k1 - 1])*Q
        else:
            Q = 1+ 0*X
            T = (Y - y[k2 - 1])*T
        a =  r(k1,k2)-P.subs([(X,x[k1]),(Y,y[k2])])
        b = Q.subs(X,x[k1])*T.subs(Y,y[k2])
        alpha = a/b
        P = P + alpha*Q*T
        if k :
            k = suivant(k, In)
    return P.simplify()
```

Ici, on donne le code du RMVPIA avec $p = 3$:

```
### RMVPIA cas p = 3

def RMVPIA3(x,y,z,v_interp):
    P, P1, P2, P3 = 0*X, 1+ 0*X, 1+ 0*Y, 1+ 0*Z
    n1, n2, n3 = len(x) - 1, len(y) - 1, len(z) - 1
    In = [n1,n2, n3]
    def r(k1,k2,k3):
        d = (n1+1)*(n2 + 1)*k3 +(n1 + 1)*k2 + k1
        return v_interp[d]
    P = 0*X
    Q, T = 1+ 0*X, 1 + 0*Y
    k =[0,0,0]
    while(k):
        k1, k2, k3 = k[0], k[1], k[2]
        if k1 > 0 :
            P1 = P1*(X - x[k1 - 1])
        elif k2 > 0 :
            P1 = 1+ 0*X
            P2 = P2*(Y - y[k2 - 1])
        elif k3 > 0 :
            P1 = 1+ 0*X
            P2 = 1+ 0*Y
            P3 = P3*(Z - z[k3 - 1])
        a =  r(k1,k2,k3) - P.subs([(X,x[k1]),(Y,y[k2]),(Z,z[k3])])
        b = P1.subs(X,x[k1])*P2.subs(Y,y[k2])*P3.subs(Z,z[k3])
        alpha = a/b
        P = P + alpha * P1 * P2 * P3
        if k :
            k = suivant(k, In)

    return P.simplify()
```

Enfin, le code du RMVPIA, cas général :

```
### Le RMVPIA  cas général

def RMVPIA(grille, v_inter):
    p = len(grille)
    In = [len(grille[i]) - 1 for i in range(p)]
    k = [0 for i in range(p)]
    zeros = [0 for i in range(p)]
    X = vecteur_indet(p)
    Pi = initialiser(X)
    P = 0 + 0*X[0]
    def r(k):
        d = 0
        k0 = [0 for i in range(p)]
        while  k0 != k:
            d += 1
            k0 = suivant(k0, In)
        return v_interp[d]
    while k:
        s = 0
        while s < p and k[s] == 0:
            s = s + 1
        if s < p and k[s] > 0 :
            for j in range(s):
                Pi[j] = 1 + 0*X[j]
            Pi[s] = Pi[s]*( X[s] - grille[s][k[s] -1])
        a = r(k) - P.subs([(X[i],grille[i][k1]) for i,k1 in zip(range(p), k)])
        b = np.product([Pi[j].subs(X[j],grille[j][k1])for j, k1 in␣
↪zip(range(p),k)])
        alpha = a/b
        Q = 1 + X[0]*0
        for j in range(p):
            Q = Q*Pi[j]
        P = P + alpha * Q
        if k :
            k = suivant(k, In)
    return P.simplify()
```

Chapitre -3

Deux algorithmes récursifs d'interpolation à plusieurs variables dans le cas d'un ensemble d'interpolation quelconque

Dans le but de résoudre le problème d'interpolation polynomiale de Lagrange à plusieurs variables sur une configuration finie quelconque (1.1.1), nous proposons dans le présent chapitre, un nouvel algorithme de calcul du polynôme et d'espace d'interpolation, que nous allons appeler RLMVPIA : Recursive Lagrange MultiVariate Polynomial Interpolation. Nous allons proposer deux versions de cet algorithme tout en étudiant les propriétés et les avantages. Tous les algorithmes donnés sont testés sur des exemples et des cas particuliers seront étudiés.

Ce chapitre est organisé comme suit : dans la section 2, nous présentons l'algorithme RLMVIA avec ordre, qui en définissant un ordre sur Z, permet de construire un espace d'interpolation associé à Z et le polynôme d'interpolation dans cet espace, pour un

ensemble de valeurs. Ensuite, par soucis d'optimalité, nous proposons une autre variante de RLMVPIA, appelée RLMVPIA aléatoire, qui utilise la notion de (Z, z)-partition.

3.1 Introduction

Le principe de notre approche est de résoudre[1] $\mathcal{P}(Z_n)$ connaissant une solution de $\mathcal{P}(Z_{n-1})$ où $Z_{n-1} = \{z_1, \ldots, z_{n-1}\}$ est un sous-ensemble de Z_n avec $n-1$ noeuds. Plus précisément, si $\mathcal{P}_{n-1}$ est un espace d'interpolation pour Z_{n-1}, et $P_{n-1} \in \mathcal{P}_{n-1}$ vérifiant $P_{n-1}(z) = r_z,\ z \in Z_{n-1}$, alors on construit, d'une certaine manière, un polynôme Q_n vérifiant

$$Q_n(z) = 0,\ \forall z \in Z_{n-1}, \tag{3.1.1}$$

et

$$Q_n(z_n) \neq 0. \tag{3.1.2}$$

Ainsi, $\mathcal{P}_n = \mathcal{P}_{n-1} \oplus \mathbb{K}Q_n$ est une solution de $\mathcal{P}(Z_n)$ et la solution de (1.1.1) dans $\mathcal{P}_n$ est un polynôme de la forme

$$P_n = P_{n-1} + s_n Q_n, \tag{3.1.3}$$

où s_n est un scalaire à calculer.

Pour résoudre, de manière récursive, le problème $\mathcal{P}(Z_n)$, on commence par choisir $\mathcal{P}_1 = vect\{(1)\}$ comme solution évidente du problème $\mathcal{P}(Z_1)$, car on a : pour tout $r_1 \in \mathbb{K}$, le polynôme constant

$$P_1 = r_1,$$

est le polynôme d'interpolation de $R_1 = (r_1)$ sur $Z_1 = \{z_1\}$ dans $\mathcal{P}_1$. Nous prenons donc $Q_1 = 1$ comme base de $\mathcal{P}_1$. Ensuite nous utiliserons dans un premier lieu, un ordre sur Z_n, et dans un second lieu, la notion de partition (Z, z) pour calculer les polynômes Q_i, pour $i \in [\![2; n]\!]$, afin de donner les deux algorithmes RLMVPIA.

1. Pour une raison d'organisation, dans le présent chapitre, l'ensemble des noeuds d'interpolation sera noté Z_n au lieu de Z.

3.2 RLMVPIA avec ordre

3.2.1 Construction du RLMVPIA avec ordre

Pour simplifier la présentation on traite, comme dans le chapitre précédent le cas $p = 2$.

Notation Dans la suite, on notera $x(z), y(z)$ l'abscisse et l'ordonnée respectives d'un élément $z \in \mathbb{K}^2$.

On considère les ensembles des abscisses (resp. des ordonnées) des noeuds de l'ensemble de l'interpolation Z_n qu'on numérote comme suit :

$$Z_x = \{x_1, \dots x_{n_1}\}$$
$$Z_y = \{y_1, \dots y_{n_2}\}.$$

On définit une relation d'ordre strict sur Z_n par, si $z = (x_{i_1}, y_{j_1})$ et $t = (x_{i_2}, y_{j_2})$ sont deux noeuds de Z alors

$$z < t \Longleftrightarrow (j_1 < j_2) \text{ ou bien } (j_1 = j_2 \text{ et } i_1 < i_2) \tag{3.2.1}$$

ensuite, on définit une relation d'ordre total sur Z par

$$z \leq t \Longleftrightarrow (z = t) \text{ ou bien } (z < t) \tag{3.2.2}$$

On ordonne aussi les éléments de Z_x et de Z_y selon l'ordre naturel de leurs indices i.e.

$$x_i \leq x_j \text{ (respectivement } y_i \leq y_j) \Longleftrightarrow (i \leq j).$$

Remarque 3.2.1 *Lorsque le corps $\mathbb{K}$ admet un ordre total comme le cas de $\mathbb{R}$, on peut choisir une numérotation des éléments de Z_x et de Z_y cohérente avec l'ordre du corps.*

On pose alors : $Z_n = \{z_1, ..., z_n\}$, tels que :

$$z_1 < z_2 < ... < z_n.$$

En concordance avec cet ordre, nous allons définir une suite de polynômes dans Π_2 qui vont servir à construire des polynômes vérifiant les conditions 3.1.1-3.1.2, en parcourant l'ensemble d'interpolation Z_n.

Définition 3.2.1 *On construit, par récurrence, une suite double finie de polynômes $(\mathrm{Q}_i, R_{y(z_i)})_{1\leq i\leq n}$ associée à Z_n par*

- *on pose $\mathrm{Q}_1 = 1$ et $R_{y(z_1)} = 1$*
- *pour $1 < i \leq n$, deux cas de figure se présentent*
  - *si $y(z_i) = y(z_{i-1})$ on pose*
    - $\mathrm{Q}_i = (x - x(z_{i-1}))\mathrm{Q}_{i-1}$
    - $R_{y(z_i)} = R_{y(z_{i-1})}$
  - *sinon, on pose*
    - $\mathrm{Q}_i = (y - y(z_{i-1}))\, R_{y(z_{i-1})}$
    - $R_{y(z_i)} = \mathrm{Q}_i$

La proposition suivante donne les expressions explicites de ces polynômes.

Proposition 3.2.1 *Soit $i \in [\![1; n]\!]$. Notons p_i le premier indice tel que, pour tout $k \in [\![p_i; i]\!]$, $y(z_k) = y(z_i)$, et $m_{p_i} \in [\![1; n_2]\!]$ tel que $y_{m_{p_i}} = y(z_{p_i})$, alors*

$$R_{y(z_i)} = \prod_{1\leq j\leq m_{p_i}-1} (y - y_j) \text{ et } Q_i = R_{y(z_i)} \prod_{p_i\leq j\leq i-1} (x - x(z_j)).$$

En particulier, si $p_i = 1$ alors

$$Q_i = \prod_{1\leq j\leq i-1} (x - x(z_j)) \text{ et } R_{y(z_i)} = 1,$$

avec la convention que chacun de ces trois termes vaut 1*, si l'ensemble des indices associé est vide.*

Preuve:

On démontre le résultat par récurrence sur $i \in [\![1; n]\!]$.

Pour $i = 1$. on a $z_1 = \mathrm{Min}(Z_n)$, donc $p_1 = 1$ et $m_{p_1} = 1$.

Soit $1 \leq i < n$, supposons que le résultat est établi pour i et montrons le pour $i + 1$.

Soit p_i le premier indice tel que, pour tout $k \in [\![p_i; i]\!]$, $y(z_k) = y(z_i)$, on note $m_{p_i} \in [\![1; n_2]\!]$ tel que $y_{m_{p_i}} = y(z_{p_i})$, on a, d'après l'hypothèse de récurrence

$$R_{y(z_i)} = \prod_{1 \leq j \leq m_{p_i}-1} (y - y_j) \text{ et } Q_i = R_{y(z_i)} \prod_{p_i \leq j \leq i-1} (x - x(z_j)) .$$

On traite deux cas

Cas 1 : $y(z_{i+1}) = y(z_i)$, donc on a $p_{i+1} = p_i$, et d'après la définition, 3.2.1, on a :

$$Q_{i+1} = (x - x(z_i))Q_i = R_{y(z_i)}(x - x_{(z_i)}) \prod_{p_i \leq j \leq i-1} (x - x(z_j)) ,$$

donc

$$Q_{i+1} = R_{y(z_i)} \prod_{p_i \leq j \leq i} (x - x(z_j)) ,$$

comme $p_{i+1} = p_i$, donc $R_{y(z_{i+1})} = R_{y(z_i)}$, on obtient

$$Q_{i+1} = R_{y(z_{i+1})} \prod_{p_i \leq j \leq i} (x - x(z_j)) ,$$

et

$$R_{y(z_{i+1})} = \prod_{1 \leq j \leq m_{p_{i+1}}-1} (y - y_j) ,$$

d'où le résultat dans ce cas.

Cas 2 : $y(z_i) < y(z_{i+1})$, donc $p_{i+1} = i+1$, et d'après l'hypothèse de récurrence,

$$R_{y(z_i)} = \prod_{1 \le j \le m_{p_i}-1} (y - y_j) \text{ et } Q_i = Q_{y(z_i)} \prod_{p_i \le j \le i-1} (x - x(z_j)),$$

d'après la définition 3.2.1

$$Q_{i+1} = (y - y(z_i))\, R_{y(z_i)} \text{ et } R_{y(z_{i+1})} = Q_{i+1},$$

donc vu l'expression de $R_{y(z_i)}$, on obtient

$$Q_{i+1} = (y - y(z_i)) \prod_{1 \le j \le m_{p_i}-1} (y - y_j),$$

d'autre part, on a $y(z_i) < y(z_{i+1})$ et $y_{m_{p_i}} = y(z_i)$, donc on déduit que $m_{p_{i+1}} = m_{p_i} + 1$; en effet, z_{i+1} est le plus petit élément de Z plus grand que z_i, on a donc

$$Q_{i+1} = \prod_{1 \le j \le m_{p_i}} (y - y_j) = \prod_{1 \le j \le m_{p_{i+1}-1}} (y - y_j),$$

avec $p_{i+1} = i+1$, il s'en suit que

$$R_{y(z_{i+1})} = \prod_{1 \le j \le m_{p_{i+1}-1}} (y - y_j) \text{ et } Q_{i+1} = R_{y(z_{i+1})} \prod_{p_{i+1} \le j \le i} (x - x(z_j)),$$

le terme le plus à droite vaut 1 (l'ensemble des indices étant vide).

D'où le résultat, d'après le principe de récurrence. ∎

Maintenant, on va démontrer que les polynômes $Q_i,\ 1 \le i \le n$ vérifient les conditions (3.1.1-3.1.2) lorsqu'on remplace Z_n et z_n par $Z_i = \{z_1, ..., z_{i-1}\}$ et z_i respectivement, $i \in [\![1; n]\!]$.

Proposition 3.2.2 *Soit $i \in [\![1; n]\!]$, pour tout $k \in [\![1; i[\![$, $Q_i(z_k) = 0$ et $Q_i(z_i) \neq 0$*

Preuve:

D'après la proposition 3.2.1, on a

$$Q_i = \prod_{1 \le j \le m_{p_i}-1} (y - y_j) \prod_{p_i \le j \le i-1} (x - x(z_j)),$$

avec p_i le premier indice tels que ; pour tout $j \in [\![p_i; i]\!]$, $y(z_j) = y(z_i)$.

Soit $k \in [\![1; i[\![$.

si $k \geq p_i$, alors $x(z_k)$ est une racine du deuxième terme dans l'expression de Q_i, donc $Q_i(z_k) = 0$,

sinon $k < p_i$, donc $z_k < z_{p_i}$ et par suite $y(z_k) \leq y(z_{p_i})$, et comme $k < p_i$, donc l'inégalité est stricte, $y(z_k) < y(z_{p_i}) = y_{m_{p_i}}$, par la suite $y(z_k)$ est une racine du premier terme dans l'expression de Q_i, il s'en suit que $Q_i(z_k) = 0$.

Soit $j \in [\![p_i; i[\![$ on a $y(z_j) = y(z_i)$ et $z_j < z_i$ donc $x(z_j) < x(z_i)$. D'autre part, pour $j \in [\![1; m_{p_i} - 1]\!]$, et avec $y_{m_{p_i}} = y(z_i)$, il s'en suit que $y_j < y(z_i)$. en déduit alors que $Q_i(z_i) \neq 0$.
■

Dans la proposition suivante nous démontrons que les polynômes $Q_i,\ 1 \leq i \leq n$ sont linéairement indépendants et engendrent un sous-espace polynomial de dimension n.

Proposition 3.2.3 *On note $\mathcal{P}_n = \text{vect}\,(Q_i)_{1\leq i\leq n}$, $\mathcal{P}_n$ est un sous espace vectoriel de Π^2 de dimension n.*

Preuve:

On montre, par l'absurde, que la famille $(Q_i)_{1\leq i\leq n}$ est libre. Soient $(\alpha_i)_{1\leq i\leq n}$ une famille non nulle de scalaires telle que $\sum_{i=1}^{n} \alpha_i Q_i = 0$ et soit k le premier indice tel que $\alpha_i \neq 0$, donc on a $\sum_{i=k}^{n} \alpha_i Q_i = 0$, en substituant par z_k, il vient, par application de la proposition 3.2.2, que $\alpha_k = 0$. Ce qui est absurde. Donc, la famille est libre et c'est une base de $\mathcal{P}_n$. ■

Nous allons, maintenant, exploiter les polynômes Q_i définis ci-dessus pour construire récursivement une solution du problème d'interpolation $\mathcal{P}(Z_n)$. Pour cela, on se donne un ensemble de valeurs d'interpolation $R_n \in \mathbb{K}^{Z_n}$.

Théorème 3.2.4 *Pour $1 \leq i \leq n$, on note $Z_i = \{z_1, \ldots z_i\}$. On définit par récurrence les polynômes $P_i \in \mathcal{P}_n,\ 0 \leq i \leq n$ comme suit*

— *on pose* $P_0 = 0$,

— *pour* $1 \leq i \leq n$

$$P_i = P_{i-1} + \alpha_i \times Q_i,$$

où $\alpha_i = \dfrac{r_{z_i} - P_{i-1}(z_i)}{Q_i(z_i)}$.

Alors, on a

1. $\forall i \in \{1, \ldots, n\}$, P_i *est un polynôme d'interpolation associé à* (Z_i, R_i).
2. $\forall i \in \{1, \ldots n\}, \quad P_i \in vect(Q_k)_{1 \leq k \leq i}$.
3. P_n *est une solution, dans* $\mathcal{P}_n$, *associée à* (Z_n, R_n).

Preuve:

On montre les résultats 1. et 2. par récurrence sur i.

Pour $i = 1$ le résultat est clair (P_1 est le polynôme constant r_{z_1}).

Soit $1 < i \leq n$. On suppose que les propriétés 1. et 2. sont vérifiées pour $i - 1$. Or, par définition, $P_i = P_{i-1} + \alpha_i \times Q_i$ avec $\alpha_i = \dfrac{r_{z_i} - P_{i-1}(z_i)}{Q_i(z_i)}$ et $Q_i(z_i) \neq 0$.

Soit $j \in [\![1; i-1]\!]$. On a $Q_i(z_j) = 0$; (proposition 3.2.2), donc $P_i(z_j) = P_{i-1}(z_j) = r_{z_j}$, car P_{i-1} est une solution associée à (Z_{i-1}, R_{i-1}). Pour $j = i$, toujours d'après la proposition 3.2.2, $Q_i(z_i) \neq 0$, donc vu l'expression de α_i, $P_i(z_i) = r_{z_i}$. On conclut que P_i est une solution associée à (Z_i, R_i).

D'autre part, comme $P_{i-1} \in vect(Q_k)_{1 \leq k \leq i-1}$, il s'en suit que $P_i = P_{i-1} + \alpha_i \times Q_i \in vect(Q_k)_{1 \leq k \leq i}$. D'où la récurrence. D'où, les propriétés 1. et 2..

En particulier, pour $i = n$, on déduit que P_n est bien une solution associée à (Z_n, R_n) . Ce qui achève la démonstration du théorème. ■

Le résultat suivant permet de conclure que Z_n est bien posé dans $\mathcal{P}_n$.

Théorème 3.2.5 *(**d'existence et d'unicité**) L'ensemble d'interpolation fini* Z_n *étant fixé, on lui associe le sous espace polynomial* $\mathcal{P}_n$ *engendré par les polynômes* Q_i,

$1 \leq i \leq n$ définis par 3.2.1. Alors, l'application notée $\mathcal{L}_{\mathcal{P}_n,Z_n}$ qui à tout ensemble de valeurs (d'interpolation) $R \in \mathbb{K}^{Z_n}$, associe le polynôme interpolant $\mathcal{L}_{\mathcal{P}_n,Z_n}(R) = P_n$ construit récursivement par le théorème 3.2.4 est un isomorphisme d'espaces vectoriels de $\mathbb{K}^{Z_n}$ vers $\mathcal{P}_n$. Donc Z_n est bien posé dans $\mathcal{P}_n$.

Preuve:

L'application qui à chaque polynôme P dans $\mathcal{P}_n$, associe l'ensemble de valeurs d'interpolation $R = P(Z_n)$ est clairement linéaire de $\mathcal{P}$ vers l'espace $\mathbb{K}^{Z_n}$. D'autre part, le théorème 3.2.4 montre qu'elle est surjective, et comme les deux espaces vectoriels $\mathcal{P}_n$ et $\mathbb{K}^{Z_n}$ sont de même dimension n, alors on déduit qu'il s'agit d'un isomorphisme d'espaces vectoriels et $\mathcal{L}_{\mathcal{P}_n,Z_n}$ est l'isomorphisme réciproque.

■

3.2.2 L'algorithme : RLMVPIA avec ordre

Nous allons maintenant décrire l'algorithme RLMVPIA avec ordre qui permet de construire à la fois la base Q_i, $1 \leq i \leq n$ du sous espace d'interpolation $\mathcal{P}_n$ associé à l'ensemble d'interpolation Z_n ordonné et le polynôme $\mathcal{L}_{\mathcal{P}_n,Z_n}(R)$ interpolant un ensemble de valeurs d'interpolation R sur Z_n.

RLMVPIA with order :

Initialization : $P = 0$, $Q = 1$, $Q_x = 1$, $Q_y = 1$

n = length(Z)

for $i = 1$ to n :

$z = Z[i]$

if $i > 1$:

$t = Z[i-1]$

if $y(z) = y(t)$

$Q_x = (x - x(t)) \times Q_x$

$Q = Q_x$

else :

$Q_y = (y - y(t)) \times Q_y$

$Q_x = Q_y$

$Q = Q_y$,

$$\alpha = \frac{r_z - P(z)}{Q(z)}$$

$P = P + \alpha Q$

return P

Dans la suite de cette section, nous allons étudier deux cas particuliers de configurations "régulières" de l'ensemble de l'interpolation Z à savoir le cas où Z est une grille ou un triangle de $\mathbb{K}^2$. On va voir que, dans le cas d'une grille, le RLMVPIA avec ordre est équivalent à l'algorithme RMVPIA de [30] et conduit aux mêmes solutions.

3.2.3 Cas où Z est une grille

On suppose, dans ce paragraphe, que l'ensemble de l'interpolation Z est une grille dans $\mathbb{K}^2$ de la forme

$$Z = (z_{(i_1,i_2)} = (x_{i_1}, y_{i_2});\ (i_1, i_2) \in [\![0; n_1]\!] \times [\![0; n_2]\!]),$$

où, n_1, n_2 sont deux entiers naturels. On munit alors l'ensemble des indices

$$\mathbb{N}_{(n_1,n_2)} = [\![0; n_1]\!] \times [\![0; n_2]\!],$$

par l'ordre défini dans le chapitre 2

$$(i_1, i_2) < (j_1, j_2) \Leftrightarrow i_2 < j_2 \text{ ou } i_2 = j_2 \text{ et } i_1 < j_1,$$

et

$$(i_1, i_2) \leq (j_1, j_2) \Leftrightarrow (i_1, i_2) < (j_1, j_2) \text{ ou } (i_1, i_2) = (j_1, j_2).$$

Ainsi l'ordre strict et total définis sur Z par (3.2.1) et (3.2.2) prennent la forme suivante :

$$z_{(i_1,i_2)} < z_{(j_1,j_2)} \Leftrightarrow (i_1, i_2) < (j_1, j_2)$$
$$z_{(i_1,i_2)} \leq z_{(j_1,j_2)} \Leftrightarrow z_{(i_1,i_2)} < z_{(j_1,j_2)} \text{ ou } z_{(i_1,i_2)} = z_{(j_1,j_2)}.$$

Le prédécesseur $(i_1, i_2)^-$ de $(i_1, i_2) \in \mathbb{N}_{(n_1,n_2)} \setminus (0, 0)$ introduit dans le chapitre 2, est défini par

$$(i_1, i_2)^- = \begin{cases} (i_1 - 1, i_2) \text{ si } i_1 > 0 \\ (n_1, i_2 - 1) \text{ si } i_1 = 0 \end{cases}.$$

Grâce à cette notion, les polynômes Q_i associés à Z selon la définition 3.2.1 prennent la forme suivante

— $\mathrm{Q}_{(0,0)} = 1$,
— Et pour $(0, 0) < (k_1, k_2) \leq (n_1, n_2)$

$$\mathrm{Q}_{(k_1,k_2)} = \begin{cases} (x - x_{k_1-1})\mathrm{Q}_{(k_1-1,k_2)}, & \text{si} \quad k_1 > 0 \\ (y - y_{k_2-1})\mathrm{Q}_{(0,k_2-1)} & \text{sinon,} \end{cases}$$

ce qui conduit à l'expression explicite suivante

$$\mathrm{Q}_{(k_1,k_2)} = \prod_{i_1=0}^{k_1-1} (x - x_{i_1}) \prod_{i_2=0}^{k_2-1} (y - y_{i_2}), \ \forall (k_1, k_2) \in [\![(0, 0); (n_1, n_2)]\!].$$

On déduit alors que le sous-espace d'interpolation $\mathcal{P} = vect(Q_{(k_1,k_2)})_{(k_1,k_2)\in[\![(0,0);(n_1,n_2)]\!]}$ associé à la grille Z n'est autre que le sous espace $\Pi_{(n_1,n_2)}$ des polynômes $P \in \Pi^2$ tels que $\deg_x(P) \leq n_1$ et $\deg_y(P) \leq n_2$, c'est le sous espace $\Pi_{(n_1,n_2)}$ de l'algorithme RMVPIA. De plus, tenant compte de ces considérations, l'algorithme RLMVPIA avec ordre prend la forme simplifiée suivante

RLMVPIA with order, case of grid

Initialization : $P = 0$, $Q = 1$, $Q_x = 1$, $Q_y = 1$

$(n_1, n_2) = \text{shape}(Z)$

for $(i_1, i_2) = (0, 0)$ to (n_1, n_2) :

    $z = Z[(i_1, i_2)]$

    if $i_1 > 0$:

        $t = Z[(i_1 - 1, i_2)]$

        $Q_x = (x - x(t)) \times Q_x$

        $Q = Q_x$

    elif $i_2 > 0$:

        $t = Z[(n_1, i_2 - 1)]$

        $Q_y = (y - y(t)) \times Q_y$

        $Q_x = Q_y$

        $Q = Q_y$

    $\alpha = \dfrac{r_z - P(z)}{Q(z)}$

    $P = P + \alpha Q$

return P

On en déduit que, dans le cas particulier d'une grille, RLMVPIA est équivalent à l'algorithme RMVPIA [30] et fournit donc les mêmes polynômes interpolant.

3.2.4 Cas où Z est un triangle

On considère ici le cas particulier, où Z est un triangle droit de $\mathbb{K}^2$ de la forme

$$Z = (z_{(i,j)} = (x_i, y_j) \in \mathbb{K}^2;\ (i,j) \in \mathbb{N}^2; 0 \leq i+j \leq n),$$

où, n est un entier naturel. On notera $\mathbb{T}_n = \{(i,j) \in \mathbb{N}^2; 0 \leq i+j \leq n\}$ qu'on peut considérer comme une partie du carré $\mathbb{N}_{(n,n)}$ qu'on munit de l'ordre total défini dans la sous-section précédente. On munit donc $\mathbb{T}_n$ de l'ordre induit et Z de l'ordre total correspondant : pour $(i_1,i_2),(j_1,j_2) \in \mathbb{T}_n$

$$z_{(i_1,i_2)} < z_{(j_1,j_2)} \Leftrightarrow (i_1,i_2) < (j_1,j_2)$$
$$z_{(i_1,i_2)} \leq z_{(j_1,j_2)} \Leftrightarrow z_{(i_1,i_2)} < z_{(j_1,j_2)} \text{ ou } z_{(i_1,i_2)} = z_{(j_1,j_2)}.$$

En cohérence avec cet ordre, les polynômes de la définition 3.2.1 associés à Z prennent la forme suivante

— $\mathrm{Q}_{(0,0)} = 1$,
— Et pour $(k_1,k_2) \in \mathbb{T}_n \setminus \{(0,0)\}$

$$\mathrm{Q}_{(k_1,k_2)} = \begin{cases} (x - x_{k_1-1})\mathrm{Q}_{(k_1-1,k_2)} & \text{si} \quad k_1 > 0 \\ (y - y_{k_2-1})\mathrm{Q}_{(0,k_2-1)} & \text{sinon} \end{cases},$$

ce qui conduit, comme dans le cas d'une grille, à l'expression explicite suivante

$$\mathrm{Q}_{(k_1,k_2)} = \prod_{i_1=0}^{k_1-1}(x - x_{i_1}) \prod_{i_2=0}^{k_2-1}(y - y_{i_2}),\ \forall (k_1,k_2) \in \mathbb{T}_n$$

On déduit alors que le sous-espace d'interpolation $\mathcal{P} = vect(Q_{(k_1,k_2)})_{(k_1,k_2)\in\mathbb{T}_n}$ associé à Z n'est autre que le sous espace des polynômes $P \in \Pi^2$ de degré total n'excédant pas n i.e $\deg_x(P) + \deg_y(P) \leq n$. Comme on peut le constater dans [7, 66, 70, 81] c'est le sous espace naturel d'interpolation et l'espace optimal introduit dans [5, 7, 8]. Notons ici que le cas où l'ensemble des indices associé à Z est un triangle, est un cas particulier de "Lower set"[39, 40, 66, 70]. De plus, l'algorithme RLMVPIA avec ordre prend la forme simplifiée suivante

RLMVPIA with order, case of right triangle

Initialization : $P = 0$, $Q = 1$, $Q_x = 1$, $Q_y = 1$

for $i_2 = 0$ to n :

  for $i_1 = 0$ to $n - i_2$:

    $z = Z[(i_1, i_2)]$

    if $i_1 > 0$:

      $t = Z[(i_1 - 1, i_2)]$

      $Q_x = (x - x(t)) \times Q_x$

      $Q = Q_x$

    elif $i_2 > 0$:

      $t = Z[(n - i_2 + 1, i_2 - 1)]$

      $Q_y = (y - y(t)) \times Q_y$

      $Q_x = Q_y$

      $Q = Q_y$

  $\alpha = \dfrac{r_z - P(z)}{Q(z)}$

  $P = P + \alpha Q$

return P

3.2.5 Généralisation RLMVPIA avec ordre

Dans ce paragraphe, nous allons proposer une généralisation de l'algorithme RLMVPIA, dans le cas où le nombre de variables est un entier $p \in \mathbb{N}^*$ quelconque. On se donne donc un ensemble d'interpolation $Z \subset \mathbb{K}^p$.

La conception de cet algorithme repose essentiellement, comme dans le cas à deux variables, sur la définition d'un ordre sur Z.

Pour $i \in [\![1; p]\!]$, on note $Z_{x_i} = \{x_{(i,1)}, \dots x_{(i,n_i)}\}$ la projection de Z sur le i-ème axe de $\mathbb{K}^p$.

On construit alors, un ordre total sur l'ensemble Z à partir de l'ordre total sur l'ensemble des indices $\mathbb{N}_{(n_1,...,n_p)}$ définie comme suit

Soient $(i_1, i_2, ..., i_p)$ et $(j_1, j_2, ..., j_p)$ deux indice dans $\mathbb{N}_{(n_1,...,n_p)}$, on pose $m(i,j) := \max\{k \in \{1, ..., p\} \ ; \ i_k \neq j_k\}$ on dit que

$$\begin{aligned}(i_1, i_2, ..., i_p) < (j_1, j_2, ..., j_p) &\Longleftrightarrow i_{m(i,j)} < j_{m(i,j)} \\ &\Longleftrightarrow (i_p < j_p) \text{ ou } (i_p = j_p \text{ et } i_{p-1} < j_{p-1}) \\ &\text{ou...ou } (i_p = j_p, ..., i_2 = j_2 \text{ et } i_1 < j_1)\end{aligned}$$

Soient $z_{(i_1,i_2,...,i_p)}$ et $z_{(j_1,j_2,...,j_p)}$ on définit un ordre sur Z par ;

$$z_{(i_1,i_2,...,i_p)} \leq z_{(j_1,j_2,...,j_p)} \Leftrightarrow (i_1, i_2, ..., i_p) \leq (j_1, j_2, ..., j_p)$$

Par analogie avec le cas $p = 2$, on définit une suite de polynômes qui va engendrer le sous espace vectoriel polynomial d'interpolation $\mathcal{P}$ de Π^p dans lequel Z est bien posé.

On pose alors $Z = (z_1, \ldots, z_n)$, tels que

$$z_1 < z_2 < \cdots < z_n.$$

Pour $i \in [\![1; p]\!]$, on note par x_i la projection suivant le i-ème axe de $\mathbb{K}^p$.

Définition 3.2.2 *On pose $Q_1 = 1$ le polynôme constant et pour $k \in [\![1; p]\!]$, on pose $Q_{(1,k)} = 1$ (polynôme constant) et pour $i \in [\![2; n]\!]$, on pose $k_i := \max\{j \in \{1, ..., p\} \ ; \ x_j(z_{i-1}) \neq x_j(z_i)\}$ et*

- *$Q_{(i,k_i)} = (x_{k_i} - x_{k_i}(z_{i-1}))\, Q_{(i-1,k_i)}$ et $Q_i = Q_{(i,k_i)}$*
- *pour $j \in [\![1; k_i - 1]\!]$, $Q_{(i,j)} = Q_i$*
- *pour $j \in [\![k_i + 1, p]\!]$, $Q_{(i,j)} = Q_{(i-1,j)}$*

Nous allons maintenant décrire l'algorithme RLMVPIA avec ordre qui utilise les polynômes définis ci-dessus 3.2.2.

Les données de l'algorithme sont, donc, une partie $Z \subset \mathbb{K}^p$ ordonnée et un ensemble de valeurs $R := \{r_z,\ z \in Z\}$.

Algorithme : RLMVPIA : with order

Initialization : $P = 0$, $(Q, Q_1, Q_2, ..., Q_p) = (1, ..., 1)$;

$n = \operatorname{card} Z$

for $i = 1$ to n :

    if $i > 1$ then

        $z1, z2 = Z[i-1], Z[i]$

        $j = p$

        while $j > 1$ and $z1[j] == z2[j]$:

            $j = j - 1$

        end

        $Q_j = Q_j \times (x_j - x_j(z_1))$

        $Q = Q_j$

        for $k = 1$ to $j - 1$:

            $Q_k = Q$

        end

    end

    $\alpha = \dfrac{r_{z2} - P(z2)}{Q(z2)}$

    $P = P + \alpha Q$

end

return P

3.2.6 Exemples

Exemple 1

Pour vérifier numériquement que les deux algorithmes RLMVPIA avec ordre et RMVPIA sont équivalents dans le cas d'une grille, nous allons appliquer RLMVPIA sur l'exemple suivant, figure -2.5, étudié dans chapitre 2, où, l'ensemble des noeuds Z est la grille

$$Z = \{0, 1, 2\} \times \{0, 1, -\frac{1}{2}, \frac{1}{2}\} \ \subset \mathbb{R}^2,$$

et les valeurs d'interpolation associées sont

$$R = (1, 0, -2, -1, 1, \frac{1}{2}, -1, 1, 0, \frac{3}{2}, \frac{7}{3}, -4).$$

En appliquant RLMVPIA avec ordre, on obtient le polynôme interpolant

$$\begin{aligned}\mathcal{L}_{\mathcal{P},Z}(R) = 1 - \frac{1}{2}x - \frac{1}{2}x^2 + 3y + \frac{71}{12}xy - \frac{21}{4}x^2y + \\ - 3y^2 + \frac{107}{6}xy^2 - \frac{49}{6}x^2y^2 - 2y^3 - 20xy^3 + \frac{38}{3}x^2y^3.\end{aligned}$$

qui coïncide bien avec la solution déterminée par RMVPIA dans [30].

Exemple 2

Dans cet exemple ; figure -3.1, l'ensemble d'interpolation est le triangle du plan $\mathbb{R}^2$

$$Z = ((0,0),(1,0),(2,0),(3,0),(4,0),(0,1),(1,1),(2,1),(3,1),(0,2),(1,2),(2,2),(0,3),(1,3),(0,4))$$

et les valeurs d'interpolation choisies sont les suivantes

$$R = (1, -10, -19, -26, -31, 1, -9, -17, -23, 1, -8, -15, 1, -7, 1).$$

La solution obtenue par RLMVPIA avec ordre est le polynôme de degré ≤ 2

$$\mathcal{L}_{\mathcal{P},Z}(R) = x^2 + xy - 12x + 1$$

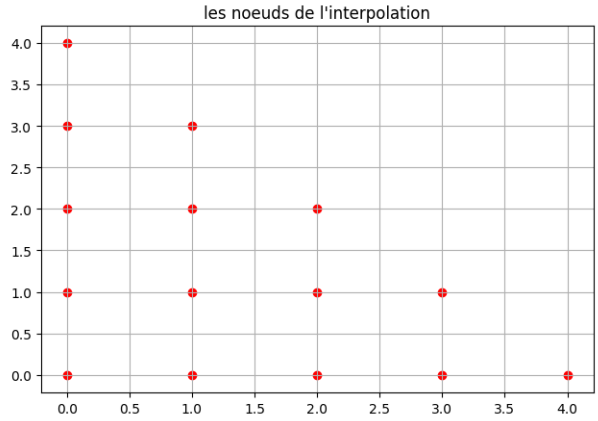

FIGURE -3.1 – L'ensemble des noeuds forme un triangle.

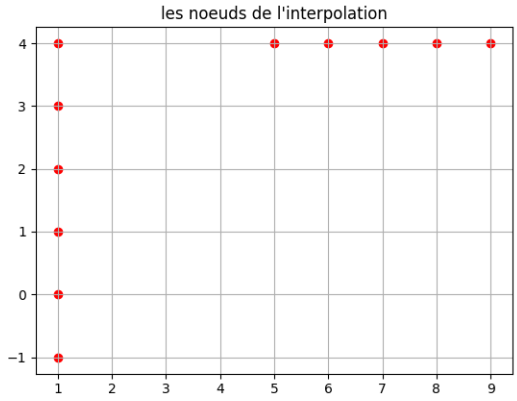

FIGURE -3.2 – L'ensemble des noeuds est aléatoire.

Exemple 3

Dans cet exemple,-3.2, on prend

$$Z = ((1,-1),(1,0),(1,1),(1,2),(1,3),(1,4),(5,4),(6,4),(7,4),(8,4),(9,4))$$

et pour valeurs d'interpolation

$$R = [34, 34, -19, -16, -4, -18, -1, 24, 18, -27, -4]$$

Par application du RLMVPIA avec ordre on obtient

$$\begin{aligned}\mathcal{L}_{\mathcal{P},Z}(R) =& \frac{79}{16800}x^5y^5 - \frac{79}{3360}x^5y^4 + \frac{79}{3360}x^5y^3 + \frac{79}{3360}x^5y^2 - \frac{79}{2800}x^5y - \frac{359}{2880}x^4y^5 + \\ & \frac{359}{576}x^4y^4 - \frac{359}{576}x^4y^3 - \frac{359}{576}x^4y^2 + \frac{359}{480}x^4y + \frac{3103}{2520}x^3y^5 - \frac{3103}{504}x^3y^4 + \\ & \frac{3103}{504}x^3y^3 + \frac{3103}{504}x^3y^2 - \frac{3103}{420}x^3y - \frac{15961}{2880}x^2y^5 + \frac{15961}{576}x^2y^4 - \frac{15961}{576}x^2y^3 - \\ & \frac{15961}{576}x^2y^2 + \frac{15961}{480}x^2y + \frac{548803}{50400}xy^5 - \frac{548803}{10080}xy^4 + \frac{548803}{10080}xy^3 + \\ & \frac{548803}{10080}xy^2 - \frac{548803}{8400}xy - \frac{607}{120}y^5 + \frac{451}{24}y^4 + \frac{47}{8}y^3 - \frac{1087}{24}y^2 - \frac{1639}{60}y + 34.\end{aligned}$$

Exemple 4, cas d'une grille avec $p = 3$

Dans cet exemple on prend la grille -2.6, (voir chapitre 2) de $\mathbb{R}^3$ considérée dans [30].

$$Z = \{0, 1\} \times \{2, 0\} \times \{1, -\frac{1}{2}, \frac{7}{3}\},$$

et la fonction de valeurs

$$R = (1, 0, -2, -1, 1, \frac{1}{2}, -1, 1, \frac{22}{7}, 0, \frac{9}{2}, -3).$$

En appliquant RLMVPIA avec ordre ou aléatoire, nous obtenons la même solution

$$\begin{aligned}\mathcal{L}_{\mathcal{P},Z}(R) = & \frac{-943}{408} + \frac{1091}{408}x + \frac{8647}{5712}y - \frac{8899}{5712}y + \frac{139}{408}xz + \frac{3887}{5712}yz \\ & - \frac{671}{408}z - \frac{1283}{5712}xyz + \frac{133}{68}z^2 - \frac{137}{68}xz^2 - \frac{661}{952}yz^2 + \frac{745}{952}xyz^2,\end{aligned}$$

qui coïncide avec la solution obtenue par RMVPIA dans [30].

Exemple 5, $p = 3$

Dans cet exemple, -3.3, nous considérons un ensemble d'interpolation, d'une autre forme, dans $\mathbb{R}^3$.

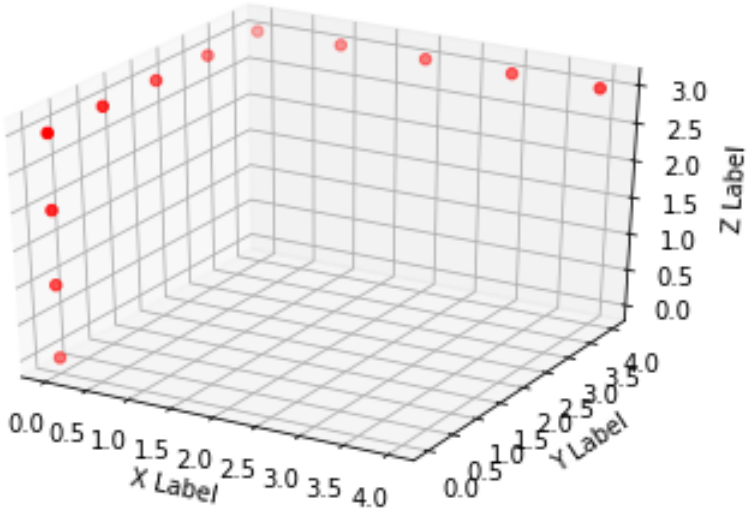

FIGURE -3.3 – L'ensemble des noeuds est aléatoire de $\mathbb{R}^3$.

$$Z = ((0,0,0),(0,0,1),(0,0,2),(0,0,3),(0,1,3),(0,2,3),(0,3,3),$$
$$(0,4,3),(1,4,3),(2,4,3),(3,4,3),(4,4,3)).$$

et on prend pour valeurs d'interpolation

$$R = (0, 4, \frac{1}{3}, 1, -1, 7, \frac{4}{3}, 2, 3, \frac{1}{2}, -2, 5).$$

En appliquant RLMVPIA avec ordre nous obtenons pour polynôme interpolant

$$\begin{aligned}\mathcal{L}_{\mathcal{P},Z}(R) =& \frac{1}{576}x^4y^4z^3 - \frac{1}{192}x^4y^4z^2 + \frac{1}{288}x^4y^4z - \frac{1}{96}x^4y^3z^3 + \frac{1}{32}x^4y^3z^2 - \frac{1}{48}x^4y^3z + \\ & \frac{11}{576}x^4y^2z^3 - \frac{11}{192}x^4y^2z^2 + \frac{11}{288}x^4y^2z - \frac{1}{96}x^4yz^3 + \frac{1}{32}x^4yz^2 - \frac{1}{48}x^4yz - \\ & \frac{11}{1728}x^3y^4z^3 + \frac{11}{576}x^3y^4z^2 - \frac{11}{864}x^3y^4z + \frac{11}{288}x^3y^3z^3 - \frac{11}{96}x^3y^3z^2 + \\ & \frac{11}{144}x^3y^3z - \frac{121}{1728}x^3y^2z^3 + \frac{121}{576}x^3y^2z^2 - \frac{121}{864}x^3y^2z + \frac{11}{288}x^3yz^3 - \\ & \frac{11}{96}x^3yz^2 + \frac{11}{144}x^3yz - \frac{1}{192}x^2y^4z^3 + \frac{1}{64}x^2y^4z^2 - \frac{1}{96}x^2y^4z + \frac{1}{32}x^2y^3z^3 - \\ & \frac{3}{32}x^2y^3z^2 + \frac{1}{16}x^2y^3z - \frac{11}{192}x^2y^2z^3 + \frac{11}{64}x^2y^2z^2 - \frac{11}{96}x^2y^2z + \frac{1}{32}x^2yz^3 - \\ & \frac{3}{32}x^2yz^2 + \frac{1}{16}x^2yz + \frac{29}{1728}xy^4z^3 - \frac{29}{576}xy^4z^2 + \frac{29}{864}xy^4z - \frac{29}{288}xy^3z^3 + \\ & \frac{29}{96}xy^3z^2 - \frac{29}{144}xy^3z + \frac{319}{1728}xy^2z^3 - \frac{319}{576}xy^2z^2 + \frac{319}{864}xy^2z - \\ & \frac{29}{288}xyz^3 + \frac{29}{96}xyz^2 - \frac{29}{144}xyz + \frac{131}{432}y^4z^3 - \frac{131}{144}y^4z^2 + \frac{131}{216}y^4z - \\ & \frac{535}{216}y^3z^3 + \frac{535}{72}y^3z^2 - \frac{535}{108}y^3z + \frac{2653}{432}y^2z^3 - \frac{2653}{144}y^2z^2 + \frac{2653}{216}y^2z - \\ & \frac{929}{216}yz^3 + \frac{929}{72}yz^2 - \frac{929}{108}yz + 2z^3 - \frac{59}{6}z^2 + \frac{71}{6}z.\end{aligned}$$

RLMVPIA avec ordre est un algorithme qui a l'avantage de construire des interpolants selon un schéma qui ne prend pas en considération la répartition des noeuds de l'ensemble d'interpolation Z. Mais on constate alors que les solutions ne sont pas toujours optimales dans le sens où les polynômes Q_i qui forment la base de $\mathcal{P}$ peuvent contenir un nombre excessif de termes ce qui rend le degré total des solutions très important et par la suite le coût des évaluations assez élevé. Dans la section suivante, on présentera, une deuxième approche qui consiste à construire des parcours aléatoires de Z qui vont nous permettre de déterminer des polynômes interpolant de degrés plus faibles, en général.

3.3 RLMVPIA : Approche aléatoire

Cette approche prend en considération la répartition des noeuds de l'ensemble d'interpolation considéré Z en introduisant une nouvelle notion décrite dans le paragraphe suivant. Nous définissons une notion de $(Z,z)-$partition et nous présentons un algorithme aléatoire de calcul des polynômes Q_i, pour $i \in [\![1;n]\!]$, qui sera utilisé pour donner l'algorithme RLMVPIA aléatoire .

3.3.1 Notion de (Z,z)-partition

Dans cette section, Z est un ensemble fini de $\mathbb{K}^p$, $p \in \mathbb{N}^*$ et $z \in \mathbb{K}^p \setminus Z$. Pour $k \in \{1,\ldots,p\}$, on note x_k la projection canonique sur le k-ème axe, définie par

$$x_k : \begin{array}{ccc} \mathbb{K}^p & \to & \mathbb{K} \\ t=(\alpha_1,\ldots,\alpha_p) & \mapsto & x_k(t)=\alpha_k \end{array},$$

et $x_k(Z) = \{x_k(t) : \ t \in Z\}$.

Définition 3.3.1 *Soit $I_k \subset x_k(Z)$, pour tout $k \in [\![1;p]\!]$.*

1. *On dira que $(I_1,\ldots,I_p)$ est une Z-partition si ;*

$$\forall t \in Z, \exists k \in [\![1;p]\!], x_k(t) \in I_k.$$

Dans ce cas $\sum_{k=0}^{p} \mathrm{card}(I_k)$ est dit indice de la Z-partition $(I_1,\ldots,I_p)$.

2. *Soit $z \in \mathbb{K}^p \setminus Z$. On dit que $(I_1,\ldots,I_p)$ est une (Z,z)-partition si ;*

   (a) $(I_1,\ldots,I_p)$ est une Z-partition

   (b) $\forall k \in [\![1;p]\!]$ $x_k(z) \notin I_k$

Proposition 3.3.1 *Pour tout $z \in \mathbb{K}^p \setminus Z$, une (Z,z)-partition existe toujours.*

Preuve: On note pour $k \in [\![1;p]\!]$, $H_k = \{t \in Z | x_k(t) = x_k(z)\}$.

Soit ϕ l'application définie par

$$\begin{array}{rcl} Z & \to & [\![1;p]\!] \\ t & \mapsto & \max\{k \ : \ x_k(t) \neq x_k(z)\}, \end{array}$$

ϕ est bien définie, en effet : $z \notin Z$. On pose

$$\phi(Z) = \{p_1, \ldots, p_i\} \subset [\![1;p]\!],$$

et pour $k \in [\![1;p]\!]$ on prend

$$I_k = \begin{cases} x_k(\phi^{-1}(\{k\})) & \text{si } k \in \{p_1, \ldots, p_i\} \\ \emptyset & \text{sinon ,} \end{cases}$$

On a bien $(I_1, \ldots, I_p)$ est une (Z, z)-partition. ■

Pour illustrer la preuve de la proposition, nous donnons ci-dessous quelques exemples dans les cas $p = 2$ et $p = 3$.

Exemple 1

Soient

$$Z = ((1,1),\ (2,1),\ (0,2),\ (1,2),\ (2,2)) \text{ et } z = (3,1),$$

nous avons $\phi(Z) = \{1, 2\}$ et

$$\phi^{-1}(\{1\}) = \{(1,1),\ (2,1)\},\ \phi^{-1}(\{2\}) = \{(0,2),\ (1,2),\ (2,2)\},$$

il s'ensuit que

$$I_1 = x_1(\phi^{-1})(\{1\}) = \{1, 2\},\ I_2 = x_2(\phi^{-1})(\{2\}) = \{2\}.$$

Exemple 2

Avec

$$Z = ((1,-1),(1,0),(1,1),(1,2),(1,3),(1,4),(5,4),(6,4),(7,4),(8,4),(9,4)),$$

et $z = (11, 5)$, on a $\phi(Z) = \{2\}$, on obtient alors

$$I_1 = \emptyset \text{ et } I_2 = \{-1,\ 0,\ 1,\ 2,\ 3,\ 4\}.$$

Exemple 3

On prend ici

$$Z = ((0,1,0),\ (1,1,0),\ (7,-1,0),\ (2,2,-1),\ (2,2,3),\ (0,3,1),\ (5,3,1),\ (-10,3,1)),$$

avec $z = (2, 3, 1)$, on a $\phi(Z) = \{1, 3\}$,

$$\phi^{-1}(\{1\}) = \{(0,3,1),\ (5,3,1),\ (-10,3,1)\},$$

et

$$\phi^{-1}(\{3\}) = \{(0,1,0),\ (1,1,0),\ (7,-1,0),\ (2,2,-1),\ (2,2,3)\},$$

donc

$$I_1 = \{0, 5,\ -10\},\ I_2 = \emptyset \text{ et } I_3 = \{0,\ -1,\ 3\}.$$

Par définition d'une (Z, z)-partition, on a la proposition suivante.

Proposition 3.3.2

Soit $\mathcal{I} = (I_1, \ldots, I_p)$ une (Z, z)-partition. On lui associe le polynôme noté $Q_{\mathcal{I}}$ suivant

$$Q_{\mathcal{I}} = \prod_{k=1}^{p} \prod_{\alpha \in I_k} (x_k - \alpha),$$

avec la convention que le produit est égal à 1 lorsque l'ensemble des indices est vide. On a donc

— $\forall t \in Z,\ Q_{\mathcal{I}}(t) = 0$,

— $Q_{\mathcal{I}}(z) \neq 0$.

L'algorithme suivant, appelé $ZPNA$: (Z, z)-Partition Newton Algorithm, construit aléatoirement une (Z, z)-partition et le polynôme associé.

Algorithm : ZPNA

input : Z, z

Initialization : $I_1, \ldots, I_p$: p empty sets

for t in Z (a random choice)

    Index = $\{1, \ldots, p\}$

    i = a random number of Index

    while $x_i(t) = x_i(z)$:

        remove i from Index

        i = a random number of Index

    end

    remove i from Index

    for j in Index :

        if $x_j(t)$ in I_j :

            pass to next element in Z

        end

    end

    if $x_i(t)$ not in I_i

        add $x_i(t)$ to I_i

        $Q = Q * (x_i - x_i(t))$

    end

end

return Q, $I_1, \ldots, I_p$

Remarque 3.3.1

1. *Toutes les opérations dans l'algorithme $ZPNA$ ont des coûts constants, donc la complexité dépend linéairement de la taille de l'ensemble Z.*
2. *Pour une (Z, z)-partition, le degré total du polynôme associé est égal à l'indice de la partition.*

3. *L'algorithme $ZPNA$ n'est pas déterministe. Si on applique l'algorithme plusieurs fois, on peut obtenir plusieurs (Z, z)-partitions parmi lesquelles on peut choisir celles d'indice minimal.*

Exemple

Avec $Z = ((1,-1),(1,0),(1,1),(1,2),(1,3),(1,4),(5,4),(6,4),(7,4),(8,4),(9,4))$ et $z = (11,5)$, en appliquant plusieurs fois l'algorithme $ZPNA$, on a obtenu les (Z, z)-partitions suivants.

1. $I_1 = \{1\}$, $I_2 = \{4\}$, et $Q = (x-1)(y-4)$
2. $I_1 = \{1,5,6,7\}$, $I_2 = \{-1,4\}$,
   et $Q = (x-7)(x-6)(x-5)(x-1)(y-4)(y+1)$
3. $I_1 = \{1,5,6,7,8\}$, $I_2 = \{-1,0,1,2,4\}$,
   et $Q = y(x-8)(x-7)(x-6)(x-5)(x-1)(y-4)(y-2)(y-1)(y+1)$

Il est clair que la première solution est la meilleure.

Théorème 3.3.3 *L'algorithme $ZPNA$ est correct.*

Preuve: Pour le prouver, nous utilisons un invariant de boucle qui nous aide à comprendre pourquoi un algorithme est correct. Nous devons montrer trois choses à propos d'un invariant de boucle, (voir 1.1.3)

Initialisation : Il est vrai avant la première itération de la boucle.

Maintenance : S'il est vrai avant une itération de la boucle, il reste vrai avant l'itération suivante.

Terminaison : Lorsque la boucle se termine, l'invariant nous donne une propriété utile montrant que l'algorithme est correct.

Pour l'algorithme $ZPNA$, on note pour $k \in [\![1; n]\!]$, $z_1, \ldots z_k$ les éléments traités dans les k premières itérations, $Z_k = \{z_1, \ldots z_k\}$ et $I_1^k, \ldots, I_p^k$ les valeurs de $I_1, \ldots, I_p$ dans l'itération k. La propriété suivante est donc un invariant de boucle

Au début de chaque itération de la boucle for, les $I_1^{k-1}, \ldots, I_p^{k-1}$ sont une (Z_{k-1}, z)-partition.

Pour l'initialisation, nous commençons par montrer que l'invariant de boucle tient avant la première itération de la boucle : lorsque $k = 1$, les $I_1^{k-1}, \ldots, I_p^{k-1}$ sont vides, et aucun élément n'est traité, on prend $Z_0 = \emptyset$, donc par convention $I_1^{k-1}, \ldots, I_p^{k-1}$ est une (Z_{k-1}, z)-partition.

Pour la maintenance, nous devons montrer que chaque itération maintient l'invariant de boucle.

Supposons que $I_1^{k-1}, \ldots, I_p^{k-1}$ est une (Z_{k-1}, z)-partition, on note z_k l'élément choisi au début de l'itération k, comme $z \notin Z$, la boucle (*while* $x_i(t) = x_i(z)$) se termine, on note donc i l'indice trouvé tel que $x_i(z_k) \neq x_i(z)$. L'analyse de la séquence de l'itération k permet sans difficulté d'affirmer que $I_j^k = I_j^{k-1}$ si $j \neq i$ et pour $j = i$ deux cas se présentent : soit $I_i^k = I_i^{k-1}$ dans le cas où $x_j(z_k)$ est déjà présent dans l'un des I_j, sinon $I_i^k = I_i^{k-1} \cup \{x_i(z_k)\}$. Dans les deux cas, on a $I_1^k, \ldots, I_p^k$ est une (Z_k, z)-partition : pour $t \in Z_k$, si $t \in Z_{k-1}$ alors il existe un $j \in [\![1; n]\!]$ tel que $x_j(t) \in I_j$ car $I_1^{k-1}, \ldots, I_p^{k-1}$ est une (Z_{k-1}, z)-partition, si $t = z_k$ par construction on a aussi le résultat. D'autre part, comme $x_j(z)$ n'est dans aucune des $I_j^{k-1}, j \in [\![1; n]\!]$, et comme par construction $x_i(z_k) \neq x_i(z)$, il s'ensuit que $x_i(z) \notin I_i^k$. D'où le résultat.

Pour la terminaison, on examine finalement ce qui se passera lorsque la boucle se terminera. Lorsque la boucle se termine (*for t in Z (a random choice)*), l'ensemble $Z_n = Z$, avec l'invariant de la boucle nous avons $I_1^n, \ldots, I_p^n$ (qui sont les ensembles retournés par l'algorithme) est une $(Z_n = Z, z)$-partition. L'algorithme est donc correct. ■

3.3.2 Construction du RLMVPIA aléatoire

Pour résoudre le problème $\mathcal{P}(Z_n)$, récursivement, cette fois-ci d'une façon aléatoire, nous commençons par choisir $\mathcal{P}_1 = vect\{(1)\}$ comme solution évidente du problème $\mathcal{P}(Z_1)$, nous prenons donc $Q_1 = 1$ comme base de $\mathcal{P}_1$ et nous utiliserons la notion de (Z, z)-partition pour calculer les polynômes Q_i, pour $i \in [\![2; n]\!]$, afin de donner l'algorithme RLMVPIA aléatoire.

Le résultat suivant montre comment la solution du problème $\mathcal{P}(Z_n)$ peut être

construite récursivement en utilisant la relation (3.1.3).

Théorème 3.3.4 *Pour $k \in \{2, \ldots, n\}$, soit $\mathcal{I}^{(k)} = (I_1^{(k)}, \ldots, I_p^{(k)})$ une (Z_{k-1}, z_k)-partition et soit*

$$Q_k = \prod_{i=1}^{p} \prod_{\alpha \in I_i^{(k)}} (x_i - \alpha),$$

le polynôme associé. Alors l'espace $\mathcal{P}_k = \text{vect}\{Q_1, \ldots Q_k\}$ est une solution du problème $\mathcal{P}(Z_k)$. Plus précisément, étant donné $R_k = (r_i : i = 1, \ldots, k)$ un ensemble de valeurs, le polynôme d'interpolation pour R_k sur Z_k dans $\mathcal{P}_k$ est donné par P_k

$$P_k = P_{k-1} + s_k Q_k, \tag{3.3.1}$$

où P_{k-1} est le polynôme d'interpolation pour $R_{k-1} = (r_i : i = 1, \ldots, k-1)$ sur Z_{k-1} dans $\mathcal{P}_{k-1} = \text{vect}\{Q_1, \ldots, Q_{k-1}\}$ et

$$s_k = \frac{r_k - P_{k-1}(z_k)}{Q_k(z_k)}.$$

Preuve: Soient $R_k = (r_i : i = 1, \ldots, k)$ l'ensemble de valeurs d'interpolation, et considérons P_{k-1} le polynôme d'interpolation $R_{k-1} = (r_i : i = 1, \ldots, k-1)$ sur Z_{k-1} dans $\mathcal{P}_{k-1}$. Pour $i = 1, \ldots, k-1$, comme $(I_1^{(k)}, \ldots, I_p^{(k)})$ est une (Z_{k-1}, z_k)-partition, il s'ensuit que $Q_k(z_i) = 0$. Donc, le polynôme P_k définie par l'expression ci-dessus (3.3.1) vérifie $P_k(z_i) = P_{k-1}(z_i) = r_i,\ i = 1, \ldots, k-1$. D'autre part, nous avons par définition $x_i(z_k) \notin I_i^k,\ \forall i \in \{1, \ldots, p\}$, donc $Q_k(z_k) \neq 0$. Donc, en prenant

$$s_k = \frac{r_k - P_{k-1}(z_k)}{Q_k(z_k)},$$

on obtient $P_k(z_k) = r_k$. Nous concluons que

$$P_k(z_i) = r_i,\ \forall i = 1, \ldots, k,$$

donc, P_k est un polynôme d'interpolation pour R_k sur Z_k dans $\mathcal{P}_k = \text{vect}\{Q_1, \ldots, Q_k\}$. Nous déduisons que l'application linéaire

$$P \in \mathcal{P}_k \mapsto (P(z_1), \ldots, P(z_k)) \in \mathbb{K}^{Z_k}$$

est surjective. Mais comme $\dim \mathcal{P}_k \leq k$ nous concluons que l'application est un isomorphisme et que $\mathcal{P}_k$ est un espace d'interpolation pour Z_k. D'où le résultat.

■

Remarque 3.3.2

Les polynômes d'interpolation P_k obtenus par le théorème précédent dépendent du choix d'indexation des noeuds de Z_n et aussi de (Z, z)-partition.

Pour construire une solution de $\mathcal{P}(Z_n)$, l'algorithme suivant RLMVPIA aléatoire, choisit une indexation aléatoire des noeuds d'interpolation. C'est une traduction directe du théorème 3.3.4.

Algorithm : RLMVPIA random

Input : Z, R

---

n = lenght(Z)

$P_0 = 0$

$Z_0 = [\,]$

for $k = 1$ to n :

z_k = a random point of Z

$Q_k = \text{ZPNA}(Z_{k-1}, z_k)[1]$

$Z_k = Z_{k-1} \cup \{z_k\}$

remove z_k from Z

$s_k = \dfrac{r_k - P_{k-1}(z_k)}{Q_k(z_k)}$

$P_k = P_{k-1} + s_k Q_k$

end

return P_n

3.3.3 Exemples

Nous allons donner deux exemples, le premier concerne le cas particulier où l'ensemble d'interpolation est une grille complète. Nous verrons que le RLMVPIA aléatoire, et le RMVPIA[30] sont équivalents (Notamment RLVMPIA avec ordre). Le second concerne une configuration aléatoire.

Exemple 1, cas d'une grille

Lorsque l'ensemble d'interpolation est une grille complète, RLMVPIA donne un résultat similaire à celui obtenu dans [30, 66]. Dans l'exemple suivant déjà étudié dans [30] où, l'ensemble de noeuds $Z_n = Z_{(n_1+1)(n_2+1)}$ est présenté dans la figure -2.5. On a

$$\begin{aligned} Z=&\{0,1,2\} \times \{0,1,-\frac{1}{2},\frac{1}{2}\},\\ R=&(1,0,-2,-1,1,\frac{1}{2},-1,1,0,\frac{3}{2},\frac{7}{3},-4). \end{aligned}$$

En appliquant le RLMVPIA aléatoire, plusieurs fois, on obtient le même polynôme d'interpolation donné par RMVPIA et RLMVPIA avec ordre

$$\begin{aligned} \mathcal{L}(\mathcal{P},Z)(R) = 1 - \frac{1}{2}x - \frac{1}{2}x^2 + 3y + \frac{71}{12}xy - \frac{21}{4}x^2y + \\ - 3y^2 + \frac{107}{6}xy^2 - \frac{49}{6}x^2y^2 - 2y^3 - 20xy^3 + \frac{38}{3}x^2y^3. \end{aligned}$$

Pour une configuration aléatoire utilisant le RLMVPIA aléatoire, nous obtenons des solutions différentes, comme on peut le voir dans l'exemple suivant.

Exemple 2

Dans cet exemple, nous prenons l'ensemble d'interpolation, figure -3.2,

$$Z_{11} = ((1,-1),(1,0),(1,1),(1,2),(1,3),(1,4),(5,4),(6,4),(7,4),(8,4),(9,4)),$$

et pour les valeurs d'interpolation

$$R_{11} = (34, 34, -19, -16, -4, -18, -1, 24, 18, -27, -4),$$

en appliquant plusieurs fois le RLMVPIA aléatoire, on donne parmi les solutions obtenues les deux suivantes. La première

$$Sol_1 = \frac{79}{140}x^5 - \frac{359}{24}x^4 + \frac{3103}{21}x^3 - \frac{15961}{24}x^2 + \frac{548803}{420}x + \frac{7}{5}y^5 - \frac{27}{2}y^4 + \frac{229}{6}y^3 - 13y^2 - \frac{991}{15}y - 741,$$

et la seconde

$$Sol_2 = -\frac{1}{4800}x^5y^5 + \frac{53}{40320}x^5y^4 - \frac{1}{630}x^5y^3 - \frac{53}{40320}x^5y^2 + \frac{181}{100800}x^5y + \frac{187}{336}x^5 + \frac{7}{960}x^4y^5 - \frac{53}{1152}x^4y^4 + \frac{203}{2880}x^4y^3 + \frac{53}{1152}x^4y^2 - \frac{7}{90}x^4y - \frac{187}{12}x^4 - \frac{97}{960}x^3y^5 + \frac{5141}{8064}x^3y^4 - \frac{21841}{20160}x^3y^3 - \frac{5141}{8064}x^3y^2 + \frac{11939}{10080}x^3y + \frac{4559}{28}x^3 + \frac{133}{192}x^2y^5 - \frac{5035}{1152}x^2y^4 + \frac{21607}{2880}x^2y^3 + \frac{5035}{1152}x^2y^2 - \frac{11801}{1440}x^2y - \frac{18547}{24}x^2 - \frac{1879}{800}xy^5 + \frac{103507}{6720}xy^4 - \frac{175969}{6720}xy^3 - \frac{103507}{6720}xy^2 + \frac{918863}{33600}xy + \frac{531425}{336}x + \frac{63}{20}y^5 - \frac{201}{8}y^4 + \frac{5555}{96}y^3 - \frac{11}{8}y^2 - \frac{41437}{480}y - \frac{7381}{8}.$$

Exemple 3 : cas d'une grille, avec p = 3

Dans cet exemple, figure -3.3, dans $\mathbb{R}^3$

$$Z = ((0,0,0), (0,0,1), (0,0,2), (0,0,3), (0,1,3), (0,2,3), (0,3,3), (0,4,3), (1,4,3), (2,4,3), (3,4,3), (4,4,3)).$$

et on prend pour valeurs d'interpolations

$$R = (0, 4, \frac{1}{3}, 1, -1, 7, \frac{4}{3}, 2, 3, \frac{1}{2}, -2, 5).$$

En appliquant RLMVPIA aléatoire plusieurs fois, nous obtenons d'autres solutions différentes. En voici celle qui a le degré le plus bas. A comparer avec celle que nous avons donné dans le RLMVPIA avec ordre !

$$\begin{aligned}\text{sol2} = &\frac{1}{4}x^4 - \frac{11}{12}x^3 - \frac{3}{4}x^2 + \frac{29}{12}x + \frac{131}{72}y^4 + \frac{1}{8}y^3z - \frac{1097}{72}y^3 - \frac{7}{8}y^2z + \frac{1421}{36}y^2 \\ &\frac{1}{2}yz^3 + 3yz^2 - \frac{15}{4}yz - \frac{505}{18}y + 2z^3 - \frac{59}{6}z^2 + \frac{71}{6}z\end{aligned}$$

Remarque 3.3.3

Le dernier exemple montre l'importance de l'approche aléatoire, dans certains cas, pour obtenir des solutions de degré assez faible ce qui peut influencer le coût des évaluations.

En conclusion, dans ce chapitre en utilisant un algorithme récursif RLMVPIA en deux versions avec ordre et aléatoire nous avons proposé de nouvelles approches de résolution du problème de l'interpolation polynomiale multivariée avec n'importe quel ensemble fini de noeuds. Cette étude montre que RLMVPIA est facile à implémenter et ne nécessite aucun stockage, et permet également de généraliser le RMVPIA [30].

3.4 Annexe : Code Python des deux algorithmes

Dans ce paragraphe, on donne le code Python des algorithmes vus dans le présent chapitre.

Commençons avec les modules scientifiques du langage Python employé

```
# Imporation des modules
from copy import deepcopy
from fractions import Fraction
import numpy as np
import numpy.random as nr
from sympy import *
import matplotlib.pyplot as plt
init_printing()
```

Les fonctions ci-dessous sont des fonctions auxiliaires employées dans les deux algorithmes

```
###### fonctions utiles ######

# construction des grilles suivant l'ordre défini ci-dessus
def produit_cartesien2(X, Y):
    Z = []
    for y in Y:
        for x in X:
            Z.append((x, y))
    return Z

def produit_cartesien3(X, Y, Z):
    R = []
    for z in Z:
        for y in Y:
            for x in X:
                R.append((x, y,z))
    return R

def projection(Z): # projection de Z suivant tous les axes
    p = len(Z[0])
    return [set(map(lambda z: z[i], Z)) for i in range(p)]

def vecteur_indet(p):  # vecteurs des inconnues X1...Xp
    X = ['X%d' % (i + 1) for i in range(p)]
    for i in range(p):
        X[i] = symbols('X%d' % (i + 1))
    return X

def intialiser(X):  # les polynomes P1...Pp
    p = len(X)
    P = [0 for i in range(p)]
    for i in range(p):
        P[i] = 1 + 0 * X[i]
    return P
```

Voici le code du RLMVPIA avec ordre et un entier $p > 1$

```python
def RLMVPIA_ordre(Z, R): # à p variables
    n = len(Z)
    p = len(Z[0])
    X = vecteur_indet(p)
    Pi = intialiser(X)
    Q = 1 + 0 * X[0]
    P = 0 + 0 * X[0]
    for i in range(n):
        # je determie le sens de deplacemement entre le noeud z_{i-1} et z_{i}.
        if i > 0:
            z1, z2 = Z[i - 1], Z[i]
            j = p - 1
            while j > 0 and z1[j] == z2[j]:
                j = j - 1
            Pi[j] = Pi[j] * (X[j] - z1[j])
            Q = Pi[j]
            for k in range(j):
                Pi[k] = Q
        a = R[i] - P.subs([(X[i], xi) for i, xi in zip(range(p), Z[i])])
        b = Q.subs([(X[i], xi) for i, xi in zip(range(p), Z[i])])
        alpha = a / b
        P = P + alpha * Q
    return expand(P)
```

Ici on donne le code du ZPNA, qui contruit une (Z, z)-partition

```
# Le ZPNA du (Z,z)-partition
def ZPNA(Z, z):
    p = len(z)
    lp = [set() for k in range(p)]
    X = vecteur_indet(p)
    Q = 1 + 0 * X[0]
    for t in Z:
        I = [ k for k in range(p)]
        i = nr.choice(I)
        while (t[i] == z[i]):
            I.remove(i)
            i = nr.choice(I)
        I.remove(i)
        c = 0
        for j in I:
            if t[j]  in lp[j]:
                break
        else:
            if t[i] not in lp[i]:
                lp[i].add(t[i])
                Q = (X[i] - t[i])*Q
    return lp, Q
```

Enfin le code du RLMVPIA aléatoire cas général

```
def RLMVPIA_randomise(Z, R):
    n = len(Z)
    c_Z = deepcopy(Z)
    c_R = deepcopy(R)
    p = len(Z[0])
    X = vecteur_indet(p)
    P = 0 + 0 * X[0]
    deja_traiter = []
    for i in range(n):
        id = nr.randint(n - i)
        z = c_Z[id]
        r = c_R[id]
        Q = ZPNA(deja_traiter, z)[1]
        a = r - P.subs(((X[j], xj) for j, xj in zip(range(p), z)))
        b = Q.subs(((X[j], xj) for j, xj in zip(range(p), z)))
        alpha = a / b
        P = P + alpha * Q
        c_Z.pop(id)
        c_R.pop(id)
        deja_traiter.append(z)
    return expand(P)
```

Chapitre -4

OLMVPIA : algorithme optimal d'interpolation de Lagrange à plusieurs variables.

Dans le but de trouver un bon espace d'interpolation associé à un ensemble d'interpolation fini quelconque, nous avons proposé l'algorithme RLMVPIA aléatoire, pour cela, il fallait exécuter plusieurs fois l'algorithme, ce qui peut devenir trop coûteux en termes de calculs.

Dans le présent chapitre, nous proposons un nouvel algorithme pour déterminer l'espace d'interpolation optimal pour un ensemble de noeuds quelconque. Cet algorithme appelé OLMVPIA : Optimal Lagrange MultiVariate Polynomial Interpolation Algorithm est basé sur une approche récursive et présente l'avantage, de construire un espace d'interpolation. Il permet aussi, en utilisant les algorithmes d'interpolation à une seule variable, de construire le polynôme d'interpolation associé à un ensemble de valeurs, ce qui entraine un gain de performance et de stabilité numérique.

Ce chapitre est organisé comme suit : dans la section 2 et afin de simplifier la présen-

tation, nous donnons la construction de l'algorithme OLMVPIA dans le cas de deux variables. la généralisation à $p > 2$ variables sera présentée dans la sections 3, ensuite dans les sections 4 et 5 respectivement, on donne des exemples d'applications et le code Python du OLMVPIA.

4.1 Introduction

On se place dans le cadre général, défini dans la section 1.1.1, et soit p et un entier naturel, $p \neq 0$ et $\mathbb{K}$ un corps commutatif. L'espace des polynômes à p variables est noté $\Pi^p = \mathbb{K}[x_1, \ldots, x_p]$.

On se donne un ensemble d'interpolation fini $Z = \{z_1, \ldots, z_n\}$ à n noeuds distincts de $\mathbb{K}^p$. Rappelons le problème de l'interpolation polynomiale de Lagrange 1.1.1 qui consiste à chercher, un sous espace adéquat $\mathcal{P} \subset \Pi^p$, tel que pour tout ensemble de valeurs $R = (r_z : z \in Z) \in \mathbb{K}^Z$, il existe une unique fonction polynomiale $P \in \mathcal{P}$ vérifiant

$$P(Z) = R, \quad \text{i.e.} \quad P(z) = r_z, \quad z \in Z, \tag{4.1.1}$$

Comme les polynômes d'interpolation sont souvent utilisés en théorie de l'approximation, nous devons les évaluer en plusieurs points, ce qui génère un grand nombre d'opérations proportionnel au degré du polynôme d'interpolation utilisé. D'où l'intérêt de rechercher des polynômes d'interpolation de bas degré.

Notation Si $S \subset \Pi^p$ est un sous-espace de dimension finie, on note
$\deg(S) = \max\{\deg(P) : \ P \in S\}$.

Cette question d'optimalité a été examinée par de Boor et Ron ([5]) qui l'ont étudié en profondeur dans une série d'articles [6, 7, 9, 10]. Ils ont introduit un sous-espace polynomial particulier qu'ils ont appelé le moindre choix "least interpolant". Dans la suite, nous dirons qu'un sous-espace $\mathcal{P}$ est un espace d'interpolation optimal pour Z si

— Le problème d'interpolation de Lagrange associé à Z est bien posé dans $\mathcal{P}$.

— $\mathcal{P}$ est de degré minimal ; i.e. si Z est bien posé dans un autre sous-espace d'interpolation S alors $\deg(\mathcal{P}) \leq \deg(S)$.

4.2 Polynôme d'interpolation optimal à deux variables

Pour simplifier, nous nous plaçons dans le contexte des polynômes à deux variables, et nous noterons $x_1(z), x_2(z)$ les abscisses et ordonnées respectives d'un point $z \in \mathbb{K}^2$.

Posons

$$\begin{aligned} x_1(Z) &:= \{x_{1,1}, \ldots, x_{1,m_1}\} \\ x_2(Z) &:= \{x_{2,1}, \ldots, x_{2,m_2}\} \end{aligned},$$

les ensembles des différentes abscisses (resp. ordonnées) de tous les noeuds de Z. On définit une fonction poids f qui à chaque $i = 1, 2$ et à chaque coordonnée $x_{i,h}$: $h = 1, ..., m_i$ associe l'entier naturel

$$f(i,h) := \text{card}\{z \in Z : x_i(z) = x_{i,h}\}.$$

On compare ces poids et on considère le plus grand. Soit $i_1 \in \{1, 2\}$ et $h_1 \in \{1, ..., m_{i_1}\}$ telles que

$$f(i_1, h_1) = \max\{f(i,h) \ : \ i = 1, 2\,;\, h = 1, ..., m_i\}.$$

x_{i_1,h_1} est la coordonnée la plus répétée. On lui associe la partie de Z sous-jacente

$$Z_1 := \{z \in Z \ : x_{i_1}(z) = x_{i_1,h_1}\}.$$

Remarque 4.2.1 *Lorsqu'il n'y a pas de confusion, on remplacera $f(i,h)$ par $f(x_{i,h})$.*

Exemple : Considérons l'ensemble d'interpolation

$$Z = \{(0,1), (0,-1), (0,2), (1,1), (1,-1), (1,2), (2,1), (1,3)\}$$

de sorte que

$$\begin{aligned} x_1(Z) &= \{0, 1, 2\}, \\ x_2(Z) &= \{1, -1, 2, 3\}. \end{aligned}$$

Dressons alors la table des poids des éléments de Z

$x_{2,h}\backslash\ x_{1,h}$	0	1	2	$f(x_{2,h})$
1	·	·	·	3
-1	·	·		2
2	·	·		2
3		·		1
$f(x_{1,h})$	3	4	1	

On en déduit que l'abscisse 1 est la coordonnée de poids maximal égal à 4 et on a

$$Z_1 = \{(1,1),(1,-1),(1,2),(1,3)\}.$$

Revenons au cas général et posons $n_1 := f(i_1,h_1) = card(Z_1)$ puis notons

$$Z_1 := \{z_1,\ldots,z_{n_1}\},\ R_1 := \{r_z,\ z \in Z_1\},$$

$$j_1 = 3 - i_1 = \begin{cases} 2 \text{ si } i_1 = 1 \\ 1 \text{ si } i_1 = 2 \end{cases},$$

$$x_{j_1}(Z_1) = \{x_1^{(1)},\ldots,x_{n_1}^{(1)}\}.$$

$x_{j_1}(Z_1)$ est donc une partie finie à n_1 points distincts de $\mathbb{K}$. Considérons alors l'unique polynôme d'interpolation de Lagrange P_1 en la variable x_{j_1} de degré $< n_1$ vérifiant

$$P_1(x_h^{(1)}) = r_{z_h} : h = 1,...,n_1.$$

Proposition 4.2.1

Nous supposons que $Z\backslash Z_1 = \emptyset$. Alors $\mathcal{P}_1$ est un espace optimal d'interpolation sur Z.

Preuve: $x_{j_1}(Z_1)$ est une partie finie de n_1 points distincts deux à deux dans $\mathbb{K}$. Considérons alors l'unique polynôme d'interpolation de Lagrange $P_1 \in \mathcal{P}_1$ en la variable x_{j_1} de degré $< n_1$ vérifiant

$$P_1(x_{j_1}(z_h)) = r_{z_h} : h = 1,...,n_1.$$

Alors l'application linéaire

$$\Phi :\ P \in \mathcal{P}_1 \to (P(z))_{z\in Z_1} \in \mathbb{K}^{Z_1}$$

est surjective, d'autre part si l'on prend S un autre sous-espace d'interpolation de dimension finie, alors l'espace

$$\tilde{S} = \{\tilde{P}(x_{j_1}) = P(x_{j_1}, x_{i_1,h_1}) :\ P \in S\}$$

est un espace d'interpolation à une variable associée à $x_{j_1}(Z_1)$ alors $deg(\tilde{S}) \geq n_1 - 1$. ■

Remarque 4.2.2 *Le polynôme π_1, définie par*

$$\pi_1\,(x_1, x_2) = P_1(x_{j_1}),$$

est de degré total $< n_1$ interpolant (Z_1, R_1). On peut aussi remarquer que le degré total de π_1 coïncide ici avec le degré partiel selon x_{j_1} :

$$\deg(\pi_1) = \deg_{x_{j_1}}(\pi_1) = \deg(P_1).$$

Théorème 4.2.2

On suppose que $Z\backslash Z_1 = \emptyset$ et $R \in \mathbb{K}^Z$. Alors π_1 est un polynôme interpolant de (Z, R) de degré total minimal et les autres polynômes d'interpolation sont de la forme

$$\pi(x_1, x_2) = \pi_1\,(x_1,\ x_2) + U(x_1, x_2) * \prod_{z\in Z}(x_{j_1} - x_{j_1}(z)) + (x_{i_1} - x_{i_1,h_1})\,V(x_1, x_2).$$

où, $U, V \in \mathbb{K}[x_1, x_2]$. Les polynômes d'interpolation de degré minimal sont de la forme

$$\pi(x_1, x_2) = \pi_1\,(x_1, x_2) + (x_{i_1} - x_{i_1,h_1})\,V(x_1, x_2),$$

où, $V \in \mathbb{K}[x_1, x_2]$ et $\deg_{x_{j_1}}(V) < \deg(\pi_1)$.

Preuve: Comme Z coïncide ici avec Z_1, alors et d'après, le lemme précédent, π_1 est un polynôme d'interpolation de Z sur R. Considérons, alors, un polynôme arbitraire π interpolant Z sur R. C'est à dire

$$\pi(z) = r_z = \pi_1(z) = P_1(x_{j_1}(z)),\ \forall z \in Z.$$

On pose

$$Q_1 := \prod_{z \in Z} (x_{j_1} - x_{j_1}(z)).$$

En regardant $\pi - \pi_1$ comme un polynôme de $\mathbb{K}[x_{i_1}][x_{j_1}]$, on peut affirmer l'existence de deux polynômes $U, V_1 \in \mathbb{K}[x_{i_1}][x_{j_1}]$ tels que

$$\pi - \pi_1 = U * Q_1 + V_1,$$
$$\deg_{x_{j_1}}(V_1) < \deg(Q_1) = n_1.$$

On a alors

$$\pi(x_1, x_2) = P_1(x_{j_1}) + U(x_1, x_2) * Q_1(x_{j_1}) + V_1(x_1, x_2). \tag{4.2.1}$$

Écrivons $V_1(x_1, x_2)$ sous la forme

$$V_1(x_1, x_2) = \sum_{k=0}^{n_1-1} v_k(x_{i_1}) x_{j_1}^k,$$

où, $v_0, \ldots, v_{n_1-1}$ sont des polynômes en la variable x_{i_1}. En évaluant l'expression (4.2.1) en chaque noeud $z \in Z$, on déduit que

$$\sum_{k=0}^{n_1-1} v_k(x_{i_1,h_1})(x_{j_1}(z))^k = 0.$$

Ce qui montre que le polynôme

$$\sum_{k=0}^{n_1-1} v_k(x_{i_1,h_1}) x_{j_1}^k$$

qui est de degré $< n_1 = \mathrm{card}(Z)$, admet n_1 racines distinctes et par suite il est nul. Ceci justifie que x_{i_1,h_1} est une racine commune de tous les polynômes $v_0, \ldots, v_{n_1-1}$. D'où, l'existence d'un polynôme $V \in \mathbb{K}[x_1, x_2]$ dont le degré selon x_{j_1} est strictement inférieur à n_1 et tel que π soit de la forme

$$\pi(x_1, x_2) = \pi_1(x_1, x_2) + U(x_1, x_2) * Q_1(x_{j_1}) + (x_{i_1} - x_{i_1,h_1}) V(x_1, x_2).$$

Il s'en suit que si U est non nul alors

$$\deg(\pi) \geq \deg_{x_{j_1}}(\pi) \geq \deg(Q_1) = n_1 > \deg(\pi_1).$$

Donc, pour que π soit un interpolant de degré minimal, il faut avoir $U = 0$. Ce qui revient à dire que π est de la forme

$$\pi(x_1, x_2) = \pi_1(x_1, x_2) + (x_{i_1} - x_{i_1,h_1})\, V(x_1, x_2). \tag{4.2.2}$$

Par suite, on a encore

$$\deg(\pi) \geq \deg(\pi_1),$$

puisque, si le terme de plus haut degré dans $\pi_1(x_1, x_2) = P_1(x_{j_1})$ est éliminé par $(x_{i_1} - x_{i_1,h_1})\, V(x_1, x_2)$ alors cela entraînera qu'il sera éliminé par $-x_{i_1,h_1} V(x_1, x_2)$ et par suite on aura $\deg(V(x_1, x_2)) \geq \deg(\pi_1)$, ce qui va entraîner que $\deg(\pi) > \deg(\pi_1)$. En conclusion, π_1 est un interpolant de Z *sur* R de degré minimal et tout autre polynôme π interpolant de degré minimal est de la forme

$$\pi(x_1, x_2) = \pi_1(x_1, x_2) + (x_{i_1} - x_{i_1,h_1})\, V(x_1, x_2) \tag{4.2.3}$$

avec la condition

$$\deg(V) < deg(\pi_1).$$

■

Dans le cas où l'ensemble $Z \backslash Z_1 \neq \emptyset$, le théorème suivant donne une relation entre le sous espace d'interpolation polynomial optimal associé à Z et celui associé à $Z \backslash Z_1$.

Théorème 4.2.3 *Si $Z \backslash Z_1 \neq \emptyset$, nous supposons que nous connaissons un espace d'interpolation $\tilde{\mathcal{P}}$ associé à $\tilde{Z} = Z \backslash Z_1$. Alors l'espace*

$$\mathcal{P} = \mathcal{P}_1 \oplus (x_{i_1} - x_{i_1,h_1})\tilde{\mathcal{P}},$$

est un espace d'interpolation optimal pour Z et nous avons

$$\min(n_1 - 1,\, \deg(\tilde{P})) \leq \deg(\mathcal{P}) \leq \max(n_1 - 1,\, 1 + \deg(\tilde{P}))$$

Ainsi et par récurrence on obtient le résultat suivant qui nous permet de construire de proche en proche un interpolant optimal [32].

Théorème 4.2.4 *Il existe une partition $(Z_1, ..., Z_s)$ de Z, des indices $i_1, ..., i_s \in \{1, 2\}$ et des entiers $h_1, ..., h_s$ tels que*

Z_1 est constitué des noeuds de Z ayant en commun la coordonnée x_{i_1,h_1}, de plus grand poids dans Z,

Z_2 est constitué des noeuds de $Z \backslash Z_1$ ayant en commun la coordonnée x_{i_2,h_2} de plus grand poids dans $Z \backslash Z_1$,

Z_s est constitué des noeuds de $Z \backslash \overset{s-1}{\underset{r=1}{\cup}} Z_r$ ayant en commun la coordonnée x_{i_s,h_s} de plus grand poids dans $Z \backslash \overset{s-1}{\underset{r=1}{\cup}} Z_r$.

Alors et en posant, pour $r = 1, ..., s$

$$j_r = 3 - i_r,$$

et en calculant récursivement les polynômes d'interpolation de Lagrange à une variable $P_1, ..., P_s$ de degré minimal auxquels on associe des polynômes à deux variables $\pi_1, \ldots, \pi_s$ tels qu'on ait

$$\begin{aligned}
P_1(x_{j_1}(z)) &= r_z, \quad \forall z \in Z_1, \\
\pi_1(x_1, x_2) &= P_1(x_{j_1}), \\
P_2(x_{j_2}(z)) &= \frac{r_z - \pi_1(z)}{(x_{i_1}(z) - x_{i_1,h_1})}, \quad \forall z \in Z_2, \qquad (4.2.4) \\
\pi_2(x_1, x_2) &= \pi_1(x_1, x_2) + (x_{i_1} - x_{i_1,h_1}) * P_2(x_{j_2}), \\
&\vdots \\
P_r(x_{j_r}(z)) &= \frac{r_z - \pi_{r-1}(z)}{\prod_{k=1}^{r-1}(x_{i_k}(z) - x_{i_k,h_k})}, \quad \forall z \in Z_r, \quad r = 2, ..., s, \qquad (4.2.5) \\
\pi_r(x_1, x_2) &= \pi_{r-1}(x_1, x_2) + \prod_{k=1}^{r-1}(x_{i_k} - x_{i_k,h_k}) * P_r(x_{j_r}).
\end{aligned}$$

On déduit que le polynôme

$$\pi_s(x_1, x_2) = \sum_{r=1}^{s} \prod_{k=1}^{r-1} (x_{i_k} - x_{i_k,h_k}) * P_r(x_{j_r})$$

est un polynôme d'interpolation de Lagrange associé à (Z, R) *de degré optimal et on a*

$$\deg(\pi_s) \leq \max_{1\leq r\leq s}(n_r + r - 2)$$

où, $n_r = card(Z_r)$.

Pour $k = 1 \dots s$, *notons* $\mathcal{N}_k(Z_k)$ *les bases de Newton en variables* x_{j_k} *associé à* $x_{i_k}(Z_k)$, *alors l'espace* $\mathcal{P} = vect\left(\prod_{q=1}^{k-1}(x_{i_q} - x_{i_q,h_q}) * \mathcal{N}_k(Z_k),\ k \in \{1, \dots s\}\right)$ *est l'espace optimal d'interpolation associé à* Z.

Dans la section suivante nous donnons l'algorithme nommé OLMVPIA2 pour Optimal Recursive Multivariate Polynomial Interpolation Algorithm qui est une implémentation du théorème 4.2.4.

4.2.1 L'Algorithme OLMVPIA à deux variables

Dans cette section nous allons développer l'algorithme nommé OLMVPIA2 qui calcule un polynôme d'interpolation associé à (Z, R) de degré optimal (dans le cas à deux variables). Nous commençons, tout d'abord, par donner un algorithme nommé **C_Max** qui permet de déterminer à partir de Z la coordonnée x_{i_1,h_1} ayant un poids maximal puis de construire le sous-ensemble Z_1 constitué des noeuds ayant cette coordonnée en commun. Cet algorithme réduit également l'ensemble d'interpolation de départ Z en supprimant les éléments de Z_1..

Algorithm **C_Max** : Coordonnée maximale

input : Z,

Initialization : I_1, I_2, Z_1 tree empty lists
for z in Z :
    for i in $\{1, 2\}$:
        add $x_i(z)$ in I_i
    end
end
for i in $\{1, 2\}$:
    select x_i the most repeated value in I_i
    k_i the occurence of x_i in I_i
end
select (i_1, x_{i_1}) from $\{1, 2\} \times \{x_1, x_2\}$ such as $k_{i_1} = \max\{k_j,\ j \in \{1, 2\}\}$
for $z \in Z$:
    if $x_{i_1}(z) = x_{i_1}$:
        add z to Z_1
        remove z from Z
    end
end

return Z_1, i_1

Maintenant, nous donnons un deuxième algorithme nommé **Partition** qui, faisant appel au premier, permet de construire une partition $(Z_1, ..., Z_s)$ de Z et des indices $i_1, ..., i_s$ vérifiant les contraintes d'optimalité du théorème 4.2.4.

Algorithm **Partition** : Partition of Z
input : Z
Initialization : ZP empty list while Z not empty : add **C_max**(Z) to ZP end
return ZP

Remarque 4.2.3 *L'algorithme précédent renvoie la liste ZP formée par les couples (Z_k, i_k) ; $1 \leq k \leq s$.*

Enfin, nous allons écrire l'algorithme OLMVPIA[32] avec deux variables.

Algorithm : **OLMVPIA2**
input : Z, R
Initialization : $Q = 1,\ \pi = 0$ $ZP =$ Paritition(Z) $s =$ lenght(ZP) for $i \in \{1, \ldots, s\}$ $(Z_i, h) = ZP[i]$ $j = 3 - h$ $t = x_h(Z_i[1])$ compute $R_i = \{\frac{r_z - P(z)}{Q(z)} : z \in Z_i\}$ compute $P(x_j)$: the interpolation polynomial in a one variable associated to $(x_j(Z_i), R_i)$ $P = P + Q * P$ $Q = (x_h - t) * Q$ end
return P

Remarque 4.2.4 *Pour calculer $P(x_j)$: le polynôme d'interpolation en une variable associé à $(x_j(Z_i), R_i)$, on peut utiliser des algorithmes d'interpolation classiques (voir la section 1.2) ou encore le nouvel algorithme GRPIA[59].*

4.3 OLMVPIA : généralisation à $p > 2$

Dans cette section, nous allons décrire l'algorithme nommé OLMVPIA pour calculer un polynôme d'interpolation optimal en p variables avec $p > 2$. La généralisation passe par deux étapes similaires au cas à deux variables. On construit la partition de l'ensemble des noeuds Z en fonction des poids des coordonnées, puis on interpole l'ensemble associé à la coordonnée de poids maximum. A ce stade, nous sommes ramenés à la résolution d'un problème d'interpolation optimale avec $p - 1$ variables, nous reprenons alors l'étape précédente et continuons ainsi jusqu'à atteindre le cas de deux variables. Nous construisons ainsi, étape par étape, le polynôme d'interpolation optimal.

Nous commençons d'abord, par donner un algorithme de partition suivant le même principe du théorème 4.2.4, avec Z est ici une partie finie de $\mathbb{K}^p$. La construction de cet algorithme est, comme dans le cas à deux variables, divisée en deux parties.

Algorithm **C_Max** : Maximum coordinate

input : Z,

Initialization : $I_1, \ldots, I_p, Z_1$: $p+1$ empty lists

for z in Z

    for i in $\{1, \ldots p\}$

        add $x_i(z)$ in I_i

    end

end

for i in $\{1, \ldots p\}$

    select x_i the most repeated value in I_i

    k_i the occurence of x_i in I_i

end

select (i_1, x_{i_1}) from $\{1, \ldots p\} \times \{x_1, \ldots x_p\}$ such that $k_{i_1} = \max\{k_j,\ j \in \{1, \ldots p\}\}$

for $z \in Z$:

    if $x_{i_1}(z) = x_{i_1}$

        add z to Z_1

        remove z from Z

end

return Z_1, i_1

On donne ensuite l'algorithme **Partition**, qui suit le même schéma que celle du cas à deux variables.

Algorithm **Partition** : Partition of Z
input : Z,
Initialization : ZP : empty list while Z not empty add **C_max**(Z) to ZP end
return ZP

Maintenant, on construit OLMVPIA (cas général). On commence par interpoler Z_1 en les valeurs associées, à ce niveau le problème revient à interpoler dans une dimension plus petite $(p-1)$, donc on recommence avec notre algorithme OLMVPIA et ainsi de suite jusqu'à ce qu'on arrive au cas à deux variables où on exploite OLMVPIA2, et on recommence avec $Z_i,\ i \in \{2, \ldots, s\}$ suivant le principe du théorème 4.2.4.

Algorithm :**OLMVPIA**

input : Z, R

initialisation : $P = 0$, $Q = 1$

$p = \text{lenght}(Z)$

if $p = 2$

    return **OLMVPIA2**

else

    $Zp =$ Parition(Z)

    $s = \text{length}(Zp)$

    for $k \in \{1, \ldots, s\}$

        $(Z_k, i) = ZP[k]$

        $t = x_i(Z_k[1])$ # common coordinate

        compute $R_k = \{\dfrac{r_z - P(z)}{Q(z)} : z \in Z_k\}$

        compute $Z_k^{p-1} = \{z$ deprived of i-th coordinate $: z \in Z_k\}$

        $Pt =$ **OLMVPIA**(Z_k^{p-1}, R_h)# here the interpolant does not depend on the indeterminate x

        $P = P + Q * Pt$

        $Q = (x_i - t) * Q$

end

return P

Remarque 4.3.1 *L'algorithme OLMVPIA est en effet récursif : on détermine un interpolant avec p variables en passant par l'interpolation avec $p - 1$ variables, le cas terminal de l'algorithme étant $p = 2$..*

4.4 Exemples

Exemple 1 : détaillé

Afin d'illustrer le fonctionnement de notre algorithme, l'exemple ci-dessus, donnera les étapes de construction détaillée de l'interpolant optimal. Reprenons l'ensemble d'interpolation précédent

$$Z = \{(0,1),(0,-1),(0,2),(1,1),(1,-1),(1,2),(2,1),(1,3)\}$$

et associons-lui les valeurs d'interpolation apparaissant dans le tableau suivant

$x_{2,h} \backslash\ x_{1,h}$	0	1	2	$f(x_{2,h})$
1	0	1	3	3
-1	3	4		2
2	-6	6		2
3		7		1
$f(x_{1,h})$	3	4	1	

On en déduit que l'abscisse 1 est la coordonnée de poids maximal égal à 4 et on a

$$Z_1 = \{(1,1),(1,-1),(1,2),(1,3)\},$$
$$R_1 = \{1,4,6,7\}.$$

Le polynôme d'interpolation π_1 associé à (Z_1, R_1) est

$$\pi_1(x,y) = P_1(y) = 1 - \frac{3}{2}(y-1) + \frac{13}{6}(y-1)(y+1) - \frac{25}{24}(y-1)(y+1)(y-2).$$

L'abscisse 0 est la seconde coordonnée de poids maximal avec

$$Z_2 = \{(0,1),(0,-1),(0,2)\},$$

on lui associe les valeurs d'interpolation calculées selon la formule (4.2.4)

$$\widetilde{r}_z = \frac{r_z - \pi_1(z)}{(x_1(z) - 1)},$$

ainsi

$$\begin{aligned}
\widetilde{r}_{(0,1)} &= \pi_1(0,1) = 1, \\
\widetilde{r}_{(0,-1)} &= -\left(3 - \pi_1(0,-1)\right) = 1, \\
\widetilde{r}_{(0,2)} &= -\left(-6 - \pi_1(0,2)\right) = 12.
\end{aligned}$$

Le polynôme d'interpolation de Lagrange P_2 associé est

$$P_2(x_2) = \frac{11}{3}y^2 - \frac{8}{3}.$$

On obtient le polynôme π_2 interpolant $Z_1 \cup Z_2$

$$\begin{aligned}
\pi_2(x,y) &= \pi_1(x,y) + (x-1)P_2(y) \\
&= 1 - \frac{3}{2}(y-1) + \frac{13}{6}(y-1)(y+1) - \frac{25}{24}(y-1)(y+1)(y-2) + \\
&+ (x-1)(\frac{11}{3}y^2 - \frac{8}{3}).
\end{aligned}$$

Enfin, il reste

$$Z_3 = \{(2,1)\},$$

on peut choisir l'ordonnée 1 comme celle de poids maximal. On associe à $(2,1)$ la valeur d'interpolation calculée selon la formule (4.2.5)

$$\begin{aligned}
\widetilde{r}_{(2,1)} &= \frac{r_{(2,1)} - \pi_2(2,1)}{Q(2,1)} = \frac{1}{2}, \\
Q(x,y) &= x(x-1).
\end{aligned}$$

Le polynôme d'interpolation de Lagrange associé est donc constant

$$P_3 = \frac{1}{2},$$

et par suite le polynôme d'interpolation de Lagrange associé à (Z, R) de degré optimal est

$$\begin{aligned}\pi(x,y) &= \pi_2(x,y) + \frac{1}{2}x(x-1) \\ &= \frac{1}{2}x^2 + \frac{11}{3}xy^2 - \frac{19}{6}x - \frac{25}{24}y^3 + \frac{7}{12}y^2 - \frac{11}{24}y + \frac{11}{12}.\end{aligned}$$

qui est de degré total égal à 3.

Remarque 4.4.1 *Avec le même exemple ci-dessus, il a fallu exécuter* 9 *fois l'algorithme LRMVPIA aléatoire [31] pour obtenir un polynôme interpolant de degré optimal.*

Exemple 2

Prenons l'exemple [30, 31], figure -2.5 on a

$$\begin{aligned}Z&=\{0,1,2\} \times \{0,1,-\frac{1}{2},\frac{1}{2}\}, \\ R &= (1,0,-2,-1,1,\frac{1}{2},-1,1,0,\frac{3}{2},\frac{7}{3},-4)\end{aligned}$$

OLMVPIA2 nous a donné la même solution que [30, 31], à savoir

$$\begin{aligned}\pi = 1 - \frac{1}{2}x - \frac{1}{2}x^2 + 3y + \frac{71}{12}xy - \frac{21}{4}x^2y + \\ - 3y^2 + \frac{107}{6}xy^2 - \frac{49}{6}x^2y^2 - 2y^3 - 20xy^3 + \frac{38}{3}x^2y^3.\end{aligned}$$

Ceci est justifié par le fait que dans ce cas particulier "lower set", comme il est indiqué dans [7, 39, 40, 66, 70], $\Pi_{(n_2,n_2)}$ est l'espace approprié d'interpolation.

Exemple 3

Prenons un autre exemple, figure -3.2), avec

$$Z = ((1,-1),(1,0),(1,1),(1,2),(1,3),(1,4),(5,4),(6,4),(7,4),(8,4),(9,4)),$$

et

$$R = (34, 34, -19, -16, -4, -18, -1, 24, 18, -27, -4).$$

Nous obtenons grâce à notre algorithme la même solution optimale que celle donnée par LRMVPIA aléatoire, avec plusieurs essais

$$\pi = \frac{79}{140}x^5 - \frac{359}{24}x^4 + \frac{3103}{21}x^3 - \frac{15961}{24}x^2 + \frac{548803}{420}x + \frac{7}{5}y^5 - \frac{27}{2}y^4 + \frac{229}{6}y^3 - 13y^2 - \frac{991}{15}y - 741.$$

Remarque 4.4.2 *Avec le même exemple ci-dessus, il a fallu exécuter des dizaines de fois l'algorithme LRMVPIA aléatoire [31] pour obtenir un polynôme interpolant de degré optimal.*

Exemple 4, $p = 3$

On considère, comme dans [30, 31], la grille, -2.6 de $\mathbb{R}^3$

$$Z = \{0, 1\} \times \{2, 0\} \times \{1, -\frac{1}{2}, \frac{7}{3}\},$$

et les valeurs de l'interpolation associées

$$R = (1, 0, -2, -1, 1, \frac{1}{2}, -1, 1, \frac{22}{7}, 0, \frac{9}{2}, -3).$$

L'application du OLMVPIA donne, d'une façon naturelle, la même solution obtenue dans [30, 31].

$$\pi = \frac{-943}{408} + \frac{1091}{408}x + \frac{8647}{5712}y - \frac{8899}{5712}y + \frac{139}{408}xz + \frac{3887}{5712}yz - \frac{671}{408}z - \frac{1283}{5712}xyz + \frac{133}{68}z^2 - \frac{137}{68}xz^2 - \frac{661}{952}yz^2 + \frac{745}{952}xyz^2.$$

Exemple 5

Prenons un autre exemple tiré de [31], figure -3.3.

$$Z = ((0,0,0),(0,0,1),(0,0,2),(0,0,3),(0,1,3),(0,2,3),(0,3,3),$$
$$(0,4,3),(1,4,3),(2,4,3),(3,4,3),(4,4,3)).$$

et les valeurs d'interpolation

$$R = (0, 4, \frac{1}{3}, 1, -1, 7, \frac{4}{3}, 2, 3, \frac{1}{2}, -2, 5).$$

L'application de OLMVPIA nous donne

$$\pi = \frac{x^4}{4} - \frac{11x^3}{12} - \frac{3x^2}{4} + \frac{29x}{12} + \frac{131y^4}{72} - \frac{535y^3}{36} + \frac{2653y^2}{72} - \frac{929y}{36} + 2z^3 - \frac{59z^2}{6} + \frac{71z}{6}$$

Remarque 4.4.3 *Pour le même exemple, l'algorithme LRMVPIA aléatoire [31] nous a donné après des dizaines d'essais la solution optimale suivante*

$$P_{lrmvpia} = \frac{1}{4}x^4 - \frac{11}{12}x^3 - \frac{3}{4}x^2 + \frac{29}{12}x + \frac{131}{72}y^4 + \frac{1}{8}y^3z - \frac{1097}{72}y^3 - \frac{7}{8}y^2z + \frac{1421}{36}y^2 \frac{1}{2}yz^3 + 3yz^2 - \frac{15}{4}yz - \frac{505}{18}y + 2z^3 - \frac{59}{6}z^2 + \frac{71}{6}z.$$

On peut remarquer qu'il s'agit d'un polynôme différent de celui obtenu par OLMVPIA mais les deux ont le même degré total 4.

4.4.1 Conclusion

L'algorithme du polynôme interpolant optimal : OLMVPIA présente plusieurs avantages dont l'exploitation des algorithmes d'interpolation polynomiale en une seule variable. Un inconvénient quand même est que si on rajoute un noeud ayant des coordonnées en commun avec les noeuds de l'ancien ensemble d'interpolation, il va falloir recommencer tous les calculs.

4.5 Annexe : Code Python du OLMVPIA

Dans ce paragraphe, on donne le code Python utilisé dans l'implémentation du OLMVPIA.

Commençons avec les modules scientifiques du langage Python employés et des fonctions auxiliaires utilisées

```
# imporation des modules
from fractions import Fraction
import numpy as np
import numpy.random as rd
from sympy import *
init_printing()

# fonctions utiles
def projection(Z):
    p = len(Z[0])
    return [list(map(lambda z: z[i], Z)) for i in range(p)]

def vecteur_indet(p):  # vecteurs des inconnues X1...Xp
    X = ['X%d' % (i + 1) for i in range(p)]
    for i in range(p):
        X[i] = symbols('X%d' % (i + 1))
    return X


def intialiser(X):  # les polynomes P1...Pp
    p = len(X)
    P = [0 for i in range(p)]
    for i in range(p):
        P[i] = 1 + 0 * X[i]
    return P

# definition de la fonction des valeurs d'interpolation : z --> r_{z}
def fonct_valeur_inter(Z,R):
    n = len(Z)
    v_inter = {Z[i]: R[i] for i in range(n)}
    return lambda z : v_inter[z]

def projection(Z):
    p = len(Z[0])
    return [list(map(lambda z: z[i], Z)) for i in range(p)]
```

Ensuite, on donne le code des deux algorithmes C_Max et Partition

```
#les fonction C_Max et partition

def frequence(L): ### les z_{i}  avec leurs frequences ordonnnees
    n = len(L)
    key_o = lambda x : x[1]
    Liste_freq = [ (L[i],L.count(L[i])) for  i in range(n) if L[i] not in L[ :i]]
    Liste_freq.sort(key= key_o)
    return Liste_freq

def C_Max(Z):
    p = len(Z[0])
    proj =  projection(Z)
    index_max = 0 # l'idnice de coordonne maximal
    cor_max = [] # qui va contenir la coordonnee la plus repetee
    nbr_max = 0
    for i in range(p):
        avec_frequence = frequence(proj[i])
        cord, nbr = avec_frequence[-1]
        if nbr > nbr_max:
            cor_max, index_max = cord, i
            nbr_max = nbr
    liste_max = []
    Zt = []
    for z in Z:
        if z[index_max] == cor_max:
            liste_max.append(z)
        else :
            Zt.append(z)
    return liste_max, index_max, Zt,

### Maintenant on va definir une fonction qui va nous permettre de partitioner le Z selon les coordonnees les plus répetees.

def parition(Z):
    t = len(Z)
    Z_part = []
    while t > 0:
        liste_max, index_max,  Z = C_Max(Z)
        Z_part.append([liste_max, index_max])
        t = len(Z)
    return Z_part
```

Ci-dessous l'algorithme de Newton avec les différences divisées de l'interpolation en une seule variable choisie, vu l'importance qu'il présente en temps et stabilité.

```
'''
On commence par donner un algorithme d'interpolation en une seule
variable, vu les avantages que présente l'algorithme de Newton
avec les différences divisées
'''
# l'algorithme de différence divisée, en utilisant un seul tableau.
def diff_divise(x, y):
    d = [yi for yi in y]
    n = len(x)
    for j in range(1,n):
        for i in range(n-1, j-1,-1):
            d[i] = (d[i] - d[i-1])/(x[i] - x[i-j])
    return d

# Pour calculer le polynome d'interpolation dans la base de Newton,nous avons␣
 ↪utilisé le schema d'Horner.
def interp_N(x, y, Xi):
    yt = []
    n = len(x)
    d =diff_divise(x,y)
    P = 0 + 0 * Xi
    for k in range(n-1, -1, -1):
        P = d[k] + (Xi -x[k])*P
    return expand(P)
```

OLMVPIA à deux variable

```
### OLMVPIA à deux variables

def OLMVPIA2(Z,R, X = []):
    struc_Z = parition(Z)
    s = len(struc_Z)
    p = len(Z[0])
    if len(X) == 0:
        X = vecteur_indet(p)
    P = 0 + 0 * X[0]
    Q = 1 + 0 * X[0]
    Rv = fonct_valeur_inter(Z,R)
    for h in range(s):
        Zh, i_cord = struc_Z[h]
        x = []
        y = []
        # cette boucle pour calculer les nouvelles valeurs de l'interpolation
        for z in Zh:
            a = Rv(z) - P.subs(((X[j], xj) for j, xj in zip(range(p), z)))
            b = Q.subs(((X[j], xj) for j, xj in zip(range(p), z)))
            r_z = a / b
            y.append(r_z)
            x.append(z[1 - i_cord])
        # interpolation en une seule variable
        Xi= X[1 - i_cord]
        P_k = interp_N(x, y, Xi)
        P = P + Q * P_k
        Q = Q*(X[i_cord] - z[i_cord])
    return expand(P)
```

Enfin le code du OLMVPIA dans le cas général avec $p \geq 2$ un entier quelconque

```
#### OLMVPIA : generalisation
def OLMVPIA(Z,R, X=[]):
    p = len(Z[0])
    if p == 2 :

        return OLMVPIA2(Z,R, X)
    else :
        struc_Z = parition(Z)

        s = len(struc_Z)
        if len(X) == 0:

            X = vecteur_indet(p)
        P = 0 + 0 * X[0]
        Q = 1 + 0 * X[0]
        Rv = fonct_valeur_inter(Z,R)
        for h in range(s):
            Zh, i_cord = struc_Z[h]
            Z_p_1 = [z[ : i_cord] + z[ i_cord + 1 : ] for z in Zh]
            R = []
            for z in Zh:
                a = Rv(z) - P.subs(((X[j], xj) for j, xj in zip(range(p), z)))
                b = Q.subs(((X[j], xj) for j, xj in zip(range(p), z)))
                r_z = a / b
                R.append(r_z)
            X_p = X[ : i_cord] + X[i_cord +  1 : ]
            Pp = OLMVPIA(Z_p_1,R,X_p)
            P = P + Q*Pp
            z = Zh[0]
            Q = Q*(X[i_cord] - z[i_cord])
    return expand(P)
```

Conclusion générale et perspectives

Dans le présent travail, nous nous sommes intéressés à la conception de nouvelles algorithmes de résolution de problèmes d'interpolation de Lagrange à plusieurs variables dans une configuration de noeuds quelconques.

Le chapitre 1 était un exposé des résultats classiques et utiles employés dans les différentes phases de cette étude, comme le complèment de Schur, l'interpolation polynomiale à une variable et les outils d'analyse des algorithmes.

Dans le chapitre 2, on a proposé un nouvel algorithme intitulé RMVPIA qui résout le problème de l'interpolation de Lagrange à plusieurs variables dans le cas où l'ensemble de l'interpolation est une grille. L'algorithme RMVPIA calcule aussi de manière récursive le polynôme interpolant solution unique du problème. Il a été construit à partir de l'écriture matricielle du problème posé et en appliquant les propriétés du complément de Schur pour résoudre des systèmes d'équations linéaires.

Dans le chapitre 3, on a décrit deux versions de l'algorithme RLVMPIA, qui résout le problème dans le cas général ou l'ensemble des noeuds est quelconque. L'idée consiste à généraliser l'approche de la méthode de Newton utilisée en une seule variable au cadre de plusieurs variables. Ainsi le RLMVPIA avec ordre et RLMVPIA aléatoire ont été construits et testés sur des exemples. On a montré que le LRMVPIA avec ordre est une généralisation de l'algorithme RMVPIA. On a vu aussi que dans le cas où l'ensemble est particulier "lower set", il permet d'aboutir au même résultat, comme

tous les algorithmes de construction de la solution. Dans le but de trouver un bon espace d'interpolation, nous avons proposé le RLMVPIA aléatoire. C'est un premier essai que nous avons mis en place vers la recherche de la solution optimale. L'approche est de type Newton utilisant une construction aléatoire. Ainsi, pour trouver le sous espace d'interpolation optimal, il faut exécuter l'algorithme plusieurs fois.

Dans le dernier chapitre et pour des raisons de performance algorithmique, nous nous sommes intéressés à construire directement et d'une façon déterministe, un espace d'interpolation, dans lequel le problème considéré admet une et une seule solution de degré optimal. Nous avons conçu alors le OLMVPIA qui repose sur l'idée de classer les noeuds selon un concept de poids des coordonnées.

Notre but actuel est de concentrer nos efforts de recherche dans ce domaine afin de pouvoir généraliser le travail et d'étudier le problème de l'interpolation polynomiale de Birkhoff à plusieurs variables dans le cadre général d'une configuration quelconque d'un espace de dimension finie et de chercher d'éventuelles applications dans différents domaines des mathématiques appliquées.

Nous comptons, dans l'avenir, exploiter nos algorithmes dans les domaines de la construction des courbes et des surfaces assistée par ordinateur et dans le domaine des éléments finis. Il y a aussi la possibilité d'appliquer ces algorithmes pour résoudre des problèmes d'analyse des données statistiques et de la cryptographie.

Bibliographie

[1] A. Aitken, *Determinants and Matrices*, Oliver and Boyd, Edinburgh (1956).

[2] G. Birkhoff, *General Mean Value and Reminder Theorems with Applications To Mechanical Differentiation and Quadrature*, AMS.org/Journal-terms-of-use. (1906) 107-136.

[3] G. Birkhoff, *The algebra of multivariate interpolation*, In C.V. Coffman and G.J. Fix, editors, Constructive Approaches to Mathematical Models. Academic Press.Inc. (1979) 345-3633.

[4] W. Borchardt, Über eine Interpolationsformel für eine Art symmetrischer Funktionen und deren Anwendung, Abh. d. Preuβ. Akad. d. Wiss. (1860) 1-20.

[5] C. de Boor, A. Ron, *On multivariate interpolation*, Const. Approx. 6 (1990) 287-302.

[6] Boor, C. de, Ron, A. (1991) : *On polynomial ideals of finite codimension with applications to box spline theory*. J. Math. Anal. and Appl. 158,168-193.

[7] C. De Boor A. Ron, *Computational aspects of polynomial interpolation in several variables*, Math. Comp., 58 (1992), pp. 705-727, https ://-doi.org/10.2307/2153210.

[8] . C. De Boor, On the error in multivariate polynomial interpolation, Appl. Numer. Math. 10 (1992) 297-305.

[9] Boor, C. de (1994) : *Gauss elimination by segments and multivariate polynomial interpolation*. In : Zahar, R.V.M. (ed) Approximation and Computation : A Festschrift in Honor of Walter Gautschi, pp. 87-96, Birkhäuser

[10] Boor, C. de (1995) : *A multivariate divided difference*. In : Chui, C.K., Schumaker, L.L. (eds) Approximation Theory VIII, Vol. 1 : *Approximation and Interpolation*, pp. $87-96$, World Scientific Publishing Co.

[11] C. BREZINSKI, *The Mühlbach-Neville-Aitken algorithm and some extensions*, BIT 20. 4(1980) 444-451.

[12] C. BREZINSKI, *Recursive interpolation, extrapolation and projection*, J. Comput. Appl. Math. 9(1983) 369-376.

[13] C. BREZINSKI, *Other manifestations of the Schur complement*, Linear Algebra Appl. 111(1988) 231-247.

[14] C. BREZINSKI, *History of continued fractions and Padé approximants*, Springer Series in Computational Mathematics, vol. 12. Springer-Verlag. Berlin (1991).

[15] C. BREZINSKI, *The generalisations of Newton's interpolation formula due to Mühlbach and Andoyer*, Electron. Trans. Numer. Anal. 2(1994) 130-137.

[16] C. BREZINSKI, *Historical Perspective on Interpolation, Approximation and Quadrature*, HANDBOOK OF NUMERICAL ANALYSIS, VOL. III (1994).

[17] C. BREZINSKI, M. REDIVO ZAGLIA, *A Schur complement approach to a general extrapolation algorithm*, Linear Algebra Appl. 368(2003) 279-301.

[18] J-P. CALVI, *Lectures on multivariate polynomial interpolation,* Hanoi, Summer (2005).

[19] J. M. CARNICER, M. GASCA, *Planar Configurations with Simple Lagrange Interpolation Formulae*, Mathematical Methods in CAGD. (2000) 1-8.

[20] J. Carnicer, T. Sauer, *Leibniz rules for multivariate divided differences*, J. Approx. Theory, 181 (2014), pp. 43-53, https ://doi.org/10.1016/j.jat.2014.02.002.

[21] R. COTES, *De methodo differentiali Newtonia*, appendice de Harmonia mensurarum, Cantabrigiae (1722). Traduction allemande par Kowalewski. (1917) 12-25.

[22] R. W. COTTLE, *Manifestations of the Schur complement*, Linear Algebra Appl. 8(1974) 189-211.

[23] J. CHAI, NA. LEI, Y. LI, P. XIA, *The proper interpolation space for multivariate Birkhoff interpolation*, J.C.A.M. 235(2011) 3207-3214.

[24] J-P. DEMAILLY, *Analyse numérique et équations différentielles*, EDP Sciences. (2006).

[25] R.A. DEVORE, G.G. LORENTZ, *Constructive Approximation*, Springer-Verlag, Berlin Heidelberg New-York(1993).

[26] L. DUMAS, *Modélisation à l'oral de l'agrégation Calcul scientifique*, ellipses. 1999.

[27] Encyklopäadie der mathematischen Wissenschaften, Teubner, Leipzig, pp. 1900 - 1904.

[28] P. ERDOS, P. TURAN, *On interpolation I. Quadrature-and mean-convergence in the Lagrange interpolation*, Ann of Math. 38(1937) 146-147.

[29] P. ERDOS, P. VERTESI, *On the almost everywher divergence of Lagrange interpolary polynomials for arbitrary system of nodes*, Acta Math. Acad. Sci. Hungar. 36(1980) 71-89.

[30] M. ERRACHID, A. ESSANHAJI, A. MESSAOUDI, *RMVPIA : A new algorithm for computing the Lagrange multivariate polynomial interpolation*, Numerical Algorithms Journal. (2020).

[31] A. ESSANHAJI, M. ERRACHID, *Lagrange Multivariate Polynomial Interpolation : A Random Algorithmic Approach*, Journal of Applied Mathematics, vol. 2022, Article ID 8227086, 8 pages, 2022. https ://doi.org/10.1155/2022/8227086

[32] A. ESSANHAJI, M. ERRACHID, OLMVPIA, *Optimal recursive algorithm for computing the Lagrange multivariate polynomial interpolation*, (submitted)

[33] G. FABER, *Über die interpolatorische Darstellung stetiger Funtionen*, Jahresbericht der deutschen Mathematiker-Vereingung. 23(1914) 192-210.

[34] L. FEJÉR, *über interpolation*, Götting. Nachr. (1916) 66-91.

[35] L. FEJÉR, *On the caracterization of some remarkable systems of points of interpolation by means of conjugate points*, Amer. Math. Monthly. 41(1934) 1-14.

[36] M. GASCA, J.I. MAEZTU, *On Lagrange and Hermite Interpolation in* $\mathbb{R}^k$, Numer. Math. 39(1982) 1-14.

[37] M. GASCA, A. LOPEZ-CARMONA, *A general recurrence interpolation formula and its applications to multivariate interpolation*, J. Approx. Theory. 34(1982) 361-374.

[38] M. GASCA, G. MÜHLBACH, *Generalized Schur-Complements and A Test for Total Positivity*, Applied. Numerical. Mathematics. 3(1987) 215-232.

[39] M. GASCA, T. SAUER, *On the history of multivariate polynomial interpolation*, JCAM. (2000) 23-35.

[40] M. GASCA, T. SAUER, *Polynomial interpolation in several variables*, Advances in Computational Mathematics 12 (2000) 377 ?410.

[41] C. GOLDSTEIN, *Hermite and Lipschitz : A Correspondence and Its Echoes*, https ://hal.archives-ouvertes.fr/hal-02331151. (2018)

[42] CH. GUILPIN, *Manuel de Calcul Numérique Appliqué*, EDP Sciences. 1999.

[43] CH. HERMITE, *Sur un nouveau développement en série de fonctions*, C.R. Acad. Sci. Paris. 58(1864) 266-273.

[44] CH. HERMITE, *Sur la formule d'interpolation de Lagrange*, J ; reine angew. Math. 84(1878) 70-79.

[45] E. ISAACSON, H.B. KELLER , *Analysis of Numerical Methods*, Dover, 1994.

[46] K. JBILOU, A. MESSAOUDI, *Matrix recursive interpolation algorithm for block linear systems. Direct methods*, Linear Algebra and its Applications. 294(1999) 137-154.

[47] S. KARLIN, J.M. KARON, On Hermite-Birkhoff Interpolation, J. Approx. Theory. 6(1972) 90-114.

[48] V. KOMORNIK, *Précis d'analyse réelle Topologie-Calcul différentiel-Méthodes d'approximation* ellipses. (2001).

[49] L. Kronecker, Über einige Interpolationsformeln für ganze Funktionen mehrerer Variabeln. Lecture at the academy of sciences, December 21, 1865, in : H. Hensel (Ed.), L. Kroneckers Werke, Vol. I, Teubner, Stuttgart, 1895, pp. 133-141. (reprinted by Chelsea, New York, 1968).

[50] J.L. LAGRANGE, *Leçons élémentaires sur les mathématiques*, Cours à l'école Normale de Paris. (76)VII (1795) 284-287.

[51] NA. LEI, Y. TENG, R. YU-XUE, *A fast algorithm for multivariate Hermite interpolation*, ppl. Math. J. Chinese Univ. 29(4)(2014) 438-454.

[52] G. G. LORENTZ, *Birkhoff Interpolation and the Problem of Free Matrices*, J. Approx. Theory. 6(1972) 283-290.

[53] G. G. LORENTZ, *The Work of P. Turàn on Interpolation and Approximation*, J. Approx. Theory. 29(1980) 6-10.

[54] R. A. LORENTZ, *Multivariate Birkhoff interpolation* in : Lecture Notes in Mathematics vol. 1516. Springer-Verlag. (1992).

[55] R. A. LORENTZ, *Multivariate Hermite interpolation by algebraic polynomials : A survey*, JCAM. (2000) 167-201.

[56] A. MESSAOUDI, *Recursive interpolation algorithm : a formalism for solving systems of linear equations I. Direct methods*, J. Comput. Appl. Math. 76(1996) 13-30.

[57] A. MESSAOUDI, *Recursive interpolation algorithm : a formalism for solving systems of linear equations, II. Iterative methods*, J. Comput. Appl. Math. 76(1996) 31-53.

[58] A. MESSAOUDI, R. SADAKA, H. SADOK, *New algorithm for computing the Hermite interpolation polynomial*, Numer. Algorithms. 77(4)(2018) 1069-1092.

[59] A. Messaoudi, M. Errachid, K. Jbilou, H. Sadok, *GRPIA : a new algorithm for computing interpolation polynomials*, Numerical Algorithms Journal. 80(2019) 253-278.

[60] A. Messaoudi, H. Sadok, *Recursive polynomial interpolation algorithm (RPIA)*, Numerical Algorithms Journal. 76(2017) 675-694.

[61] G. Mühlbach, *The general Neville-Aitken algorithm and some applications*, Numer Math. 31(1978) 97-110.

[62] G. Mühlbach, *The general recurrence relation for divided differences and the general Newton-interpolation-algorithm with applications to trigonometric interpolation*, Numer. Math. 32(4)(1979) 393-408.

[63] G. Mühlbach, *An Algorithmic Approach to Hermite-Birkhoff-lnterpolation*, Numer. Math. 37(1981) 339-347.

[64] S. Narumi, Some formulas in the theory of interpolation of many independent variables, Tohoku Math. J. 18 (1920) 309-321.

[65] L. Neder, Interpolationsformeln für Funktionene mehrerer Argumente, Skandinavisk Aktuarietidskrift (1926) 59.

[66] R. D. Neidinger, *Multivariate Polynomial Interpolation in Newton Forms*, Siam Review. 61(2019) 361-381.

[67] E.H. Neville, *Iterative interpolation*, J. Indian. Math. Soc. 20(1934) 87-120.

[68] I. Newton, *Methodus Differentialis*, manuscrit de (1676), publié dans Analysis per quantitatum series, fluxiones, ac differentias par W. Jones, London (1711).

[69] I. Newton, *Philosophiae Naturalis Principia Mathematica*, Printed by Joseph Streater by order of the Royal Society. London (1687).

[70] Nira Dyn, Michael S. Floater, *Multivariate polynomial interpolation on lower sets*, Journal of Approximation Theory 177 (2014) 34-42.

[71] D. V. Ouellette, *Schur complements and statistics*, Linear Algebra Appl. 36(1981) 187-295.

[72] F. PALACIOS, J.L. DIAZ, P. RUBIO,*Order regularity of two-node Birkhoff interpolation with lacunary polynomials*, Appl. Math. lett. 22(3)(2009) 386-389.

[73] J.M. PENA, T. SAUER, *On the multivariate Horner scheme* SIAM J. Numer. Anal. 37(4)(2006) 1186-1197.

[74] K. Pearson, On the construction of tables and on interpolation, Vol. 2, Cambridge University Press, Cambridge, 1920.

[75] PH. PICART, *Cours de calcul formel Algorithmes fondamentaux*, ellipses. (1999).

[76] J. RUBIO, J.L. DIAZ, P. RUBIO,*On the solvability of the Birkhoff interpolation problem*, J. Approx. Theory. 124(1)(2003) 109-114.

[77] C. RUNGE,*über empirische Functionen und die Interpolation zwischen äquidistanten Ordinaten*, Z. für Math. Phys. 46(1901) 224-243.

[78] T. SAUER, *Computationel aspects of multivariate polynomial interpolation*, Advances Comput. Math. (3)(1995) 219-238.

[79] T. SAUER, YUAN XU, *On Multivariate Lagrange Interpolation*, Math. Comp. (64)(1995) 1147-1170.

[80] T. SAUER, *Polynomial interpolation of minimal degree*, Numer. Math., 78(1997), pp. 59-85, https ://doi.org/10.1007/s002110050304.

[81] T. SAUER, *Lagrange interpolation on subgrids of tensor product grids*, Math. Comp., 73(2004), pp. 181-190, https ://doi.org/10.1090/S0025-5718-03-01557-6.

[82] I. SCHUR, *Potenzreihn im innern des einheitskreises*, J. Reine. Angew. Math. 147(1917) 205-232.

[83] Y.G. SHI, *Theory of Birkhoff interpolation*, Nova Science. (2003).

[84] A. SPITZBART, *Ageneralization of Hermite's interpolation formula*, Amer. Math. Monthly. 67(1960) 42-46.

[85] I.F. Steffensen, Interpolation, Chelsea, New York, 1927 (2nd Edition, 1950).

[86] P.L. TCHEBYCHEV, *Sur l'interpolation dans le cas d'un grand nombre de données fournies par les observations*, Mémoires de l'Acad. Imp. des sciences de Saint-Pétersbourg. (7)1(1859) 1-81.

[87] R. THEODOR, *Initiation à l'analyse numérique*, Masson. (1994).

[88] P. TURAN, *Some open problems in approximation theory*, Math. Lapok. 25 (1974) 21-75.

[89] P. TURAN, *On some open problems of approximation theory*, J. Approx. thory. 29(1980) 23-85.

[90] H. WARNER, *Remarks on Newton type multivariate interpolation for subsets of grids*, Computing. 25(1980) 181-191.

[91] WERNER, H. *Remarks on Newton type multivariate interpolation for subsets of grids*. Computing 25, 181-191 (1980). https ://doi.org/10.1007/BF02259644