USTL
FLANDRES ARTOIS

LIFL
U.A 369 du C.N.R.S

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre : 876

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

BOUSSEMART Frédéric

LA SIMULATION GRAPHIQUE INTERACTIVE DES SYSTEMES DYNAMIQUES NON LINEAIRES : Conception et Réalisation en Scratchpad

Thèse soutenue le **17 Février 1992** devant la commission d'Examen

Membres du jury :

M. MERIAUX	Président
P. CHENIN	Rapporteur
D. PINCHON	Rapporteur
J.C. FIOROT	Examinateur
G. HEGRON	Examinateur
G. JACOB	Directeur de thèse
F. LAMNABHI-LAGARRIGUE	Examinateur

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS, A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. MIGEON Michel	EUDIL
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphériques
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
M. CATTEAU Jean Pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnement
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

Remerciements

Je tiens à remercier en premier lieu Monsieur Michel Mériaux, Professeur à l'Université de Lille I et directeur du LIFL. Qu'il ait accepté de présider ce jury est pour moi une joie et un honneur.

Je remercie Monsieur Patrick Chenin, Maître de conférence au LMC-IMAG à l'Université de Grenoble I, d'avoir accepté de rapporter sur ce travail. Ses questions et ses commentaires m'ont permis d'observer mon travail d'un oeil nouveau.

Monsieur Didier Pinchon, Chargé de recherche au CNRS, Ingénieur de recherche chez IBM, a accepté de rapporter sur ce travail. Ses remarques judicieuses m'ont permis d'en étoffer le contenu et je lui en sais gré.

Je tiens également à remercier Monsieur Jean-Charles Fiorot, professeur à l'ENSIMEV, Université de Valenciennes. Une partie de cette thèse s'appuie sur ses travaux. Il a de plus accepté de me consacrer du temps, d'apporter des critiques constructives et de participer à ce jury.

Je tiens à exprimer ma reconnaissance à Monsieur Gérard Hégron, Directeur de recherche à l'INRIA-IRISA à Rennes, pour l'honneur qu'il me fait en acceptant de se pencher sur ce travail et en participant à ce jury.

Je suis reconnaissant à Madame Françoise Lamnabhi-Lagarrigue, Chargée de recherche au CNRS au LSS-ESE à Gif sur Yvette pour s'être intéressée à ce travail et pour avoir accepté de participer à ce jury.

Par sa gentillesse et sa compétence, Monsieur Gérard Jacob, Professeur à l'Université de Lille I, a su m'aider et me diriger tout en me laissant une grande liberté. Je tiens tout particulièrement à l'assurer de ma profonde gratitude.

Je tiens aussi à remercier mes camarades de l'équipe SNCF et en particulier Hoang Ngoc Minh qui a accepté de me consacrer beaucoup de son temps et qui a toujours su m'encourager dans les moments difficiles.

Les membres du département d'informatique de l'IUT de Lille, m'ont permis d'enseigner dans un cadre agréable tout en préparant cette thèse. Ils se sont montrés arrangeants en toute circonstance et je les en remercie vivement.

Je remercie également Monsieur Henri Glanc qui a assuré le tirage de cette thèse et des versions préliminaires avec soin et rapidité.

Je tiens enfin à remercier les ingénieurs du LIFL et en particulier Gilles Carin qui a installé le PC-RT et a toujours su se rendre disponible pour les nombreux services que je lui ai demandé.

Pour Florence.
Antoine,
Théo.

Table des matières

Liste des figures

0

Introduction

Ce travail s'inscrit dans le cadre du projet de développement d'outils de résolution des problèmes d'automatique non linéaire de l'équipe *Séries Non commutatives et Calcul Formel* (SNCF) du *Laboratoire d'Informatique Fondamentale de Lille* (LIFL). Notre but est de développer des *outils de simulation* pour l'étude des systèmes dynamiques non linéaires. Il s'effectue dans le cadre du calcul exact pour étendre les possibilités du *système de calcul formel* Scratchpad(et dans un proche avenir, *Axiom*).

Pouquoi faire un simulateur?

Notre motivation vient de l'inadéquation des moyens de simulation actuellement disponibles sur les points suivants :

- le paramétrage des entrées,
- les modifications interactives des entrées,
- l'exploitation de calculs exacts, rendus possibles par le calcul formel et nécessaires pour l'étude des propriétés structurelles des systèmes.

Avant de présenter le cadre scientifique dans lequel nous nous plaçons, indiquons la démarche qu'il faudrait suivre sans le développement des outils que nous présentons.

Pour cela, prenons l'exemple de l'étude du problème du *motion planning* (détaillée dans le chapitre 2) : le but est de *déterminer les commandes* (ici, nous utilisons l'algorithme de G. Jacob, [31]) qui amènent un mobile d'un point à un autre (ces deux points supposés connus). Nous désirons alors *visualiser sa trajectoire.*

Sans outil spécifique, le traitement du problème de bout en bout pourrait se décomposer, par exemple, de la manière suivante :

1. Travail du Calcul formel :
   - Entrée des données du problème.
   - Calcul des entrées du système.
   - Calcul de la représentation d'état.
2. Préparation à la simulation.

- Transcrire en Fortran les résultats du calcul formel (avec conversion des rationnels en flottants).
- Ecriture des déclarations du programme Fortran.
- Envoi sur le *Cyber*
- Compilation du programme.

3. Simulation.

- Utilisation d'une bibliothèque de calcul numérique (on peut prendre par exemple celle fournie par la société *NAG*) pour calculer les points de la trajectoire.
- Envoi des résultats au logiciel *UNIRAS* pour le tracé.

Cette démarche reste bien lourde, notamment à partir de la *Version 2.2* du *15 novembre 1989* qui était la seule à notre disposition.

Dans l'attente de la création d'interfaces efficaces entre Scratchpad (*Axiom*) et les bibliothèques numériques - qui ne manqueront pas d'être développées dans le cadre de la société *NAG* - nous sommes conduits à une autre solution : rendre Scratchpad capable de traiter les problèmes cités plus haut *de bout en bout*, ce qui permet une grande unité de traitement. Ce choix nous a donc conduit à développer de petits algorithmes numériques spécifiques à nos problèmes, qui resteront faciles à utiliser sous une session Scratchpad. On verra que ce choix nous permet aussi de comparer très facilement les résultats des algorithmes numériques et symboliques en les représentant sur un même graphique, à une même échelle et avec la même précision.

Pour cela, il est nécessaire de bien comprendre les mécanismes de Scratchpad, ses potentialités sur le plan du calcul exact comme approché. Nous décrivons dans le premier chapitre la *philosophie* de ce système de calcul formel qui se classe parmi les *langages orientés objet*. Nous développons nos programmes en respectant les règles du *génie logiciel*, ce qui est grandement facilité par l'utilisation d'un langage comme Scratchpad (programmation modulaire, héritage, généricité, surcharge d'opérateur, ...). Nos algorithmes sont alors implantés à *leur plus haut niveau de généralité*, ce qui donne un outil très souple d'étude des systèmes dynamiques : Scratchpad résoud de manière exacte (et générique) tout ce qui précède la simulation et passe ensuite sans difficulté des expressions symboliques aux valeurs numériques jusqu'à offrir l'affichage de la simulation graphique *sans changer d'environnement de travail.*

Avec ce principe de résolution, nous *évitons toute discrétisation* (et donc les problèmes d'instabilité numérique) pour calculer l'expression de la sortie des systèmes dynamiques. Ceci offre deux avantages importants : nous repoussons un peu plus loin le travail du calcul numérique (limité ici à la représentation graphique) et nous fournissons des expressions symboliques des sorties, dépendant éventuellement de paramètres réels.

Nous ajoutons ainsi à Scratchpad des fonctionnalités importantes qui nous semblent nécessaires à une analyse graphique complète du comportement des systèmes dynamiques non linéaires.

Que veut-on simuler?

Les systèmes dynamiques sont une formulation mathématique permettant de modéliser un très grand nombre de problèmes de toutes natures (physiques, électroniques, chimiques,

biomédicales, ...) par une relation entre des entrées $\{a^i \mid i \in I\!N_+\}$ et une sortie y liées par un système d'équations différentielles de la forme :

$$(\Sigma)\begin{cases} \dot{q}(t) &= \sum_{i=0}^{m} a^i(t) A_i(q) \\ y(t) &= h(q(t)) \end{cases}$$

où

- les $a_i(t)$ sont les entrées,
- q(t) est l'état du système à l'instant t,
- h est une fonction dite d'observation.

D'après la *formule fondamentale de M. Fliess* ([22]), le comportement entrée/sortie d'un tel système peut être codé par sa série génératrice, qui est une série formelle en variables non commutatives sur un alphabet de codage fini $Z.$:

$$G = \sum_{w \in Z^*} < G \mid w > w,$$

Ce codage nous amène à *l'étude syntaxique* (monoïde libre, algèbre de Lie libre, séries formelles) développée par l'école de M.P. Schützenberger ([13, 14, 22, 37]). Nous bénéficions ainsi des études systématiques de ces objets et surtout d'algorithmes bien adaptés au calcul formel ([27, 36, 32]) que nous rappelons brièvement au début du chapitre 2.

D'après la *formule fondamentale de M. Fliess*, on obtient l'expression de la sortie $y(t)$ en fonction des entrées en *évaluant* sa série génératrice comme suit :

$$y(t) = \sum_{w \in Z^*} < G \mid w > \mathcal{E}_a(w)$$

Nous rappelons au chapitre 2 le principe de cette *transformation d'évaluation* ([28]).

L'étude, comme la simulation, de tels systèmes nécessite l'emploi de techniques d'approximations ([22, 64, 27, 38, 30, 29]). Nous utilisons ici les suivantes (voir chapitre 2) :

1. La première méthode consiste à simuler les *approximations polynomiales* de la manière suivante :
   Si S est une série génératrice (à priori infinie), on l'approximera par un polynôme P :

   (a) sur une seule lettre :

   $$S = \sum_{n \geq 0} a_n t^n \qquad P = \sum_{n=0}^{k} a_n t^n$$

   (b) sur plusieurs lettres non commutatives :

   $$S = \sum_{w \in Z^*} < S \mid w > w \qquad P = \sum_{longueur(w) \leq k} < S \mid w > w$$

2. La deuxième technique d'approximation des séries génératrices est celle, étudiée à Lille par G. Jacob, C. Hespel et Hoang Ngoc Minh, des *approximants de type Padé non commutatifs*, permettant d'approcher les séries génératrices des systèmes dynamiques par des séries rationnelles de la manière suivante :

   (a) sur une seule lettre :

   $$S = \sum_{n \geq 0} a_n t^n$$

   Pour approcher de manière plus précise la fonction f, on l'approxime par une fraction rationnelle :

   $$\frac{P(t)}{Q(t)} = \sum_{n=0}^{k} a_n t^n + \sum_{n>k} b_n t^n$$

   (b) On étend cette définition aux séries sur plusieurs lettres (ne commutant pas) :

   $$S = \sum_{w \in Z^*} < S \mid w > w$$

   que l'on l'approche par une fraction rationnelle :

   $$R = \sum_{longueur(w) \leq k} < S \mid w > w + \sum_{longueur(w) \geq k} < Q \mid w > w$$

   L'approximation par des fractions rationnelles, bien que précise, s'avère insuffisante pour certains types de problèmes. En effet, si elle permet l'étude de séries formelles très compliquées telles que :

   $$S = z_1 z_0^* + (S \sqcup S) z_0 z_0^*$$

   provenant de la résolution d'équations symboliques du type :

   $$\dot{y} = u + y + y^2$$

   elle reste totalement inadaptée à l'étude de séries aussi simples que $(z_0 z_1)^*$ (sauf pour certaines restrictions de la classe des entrées, [28]).

3. Nous avons donc développé dans un troisième temps une simulation des *approximations structurelles nilpotentes* (voir [30]) des systèmes dynamiques, que nous décrivons dans le chapitre 2.

   Cette technique est d'autre part indispensable à l'étude du guidage exact sur un état cible (ou "motion planning") des systèmes nilpotents, dans la solution proposée par Sussmann ([65]), Jacob ([31]), Guyon, Jacob et Petitot ([26]). Nous rappelons également cette technique au chapitre 2, pour laquelle nous fournissons des simulations.

Comment simuler?

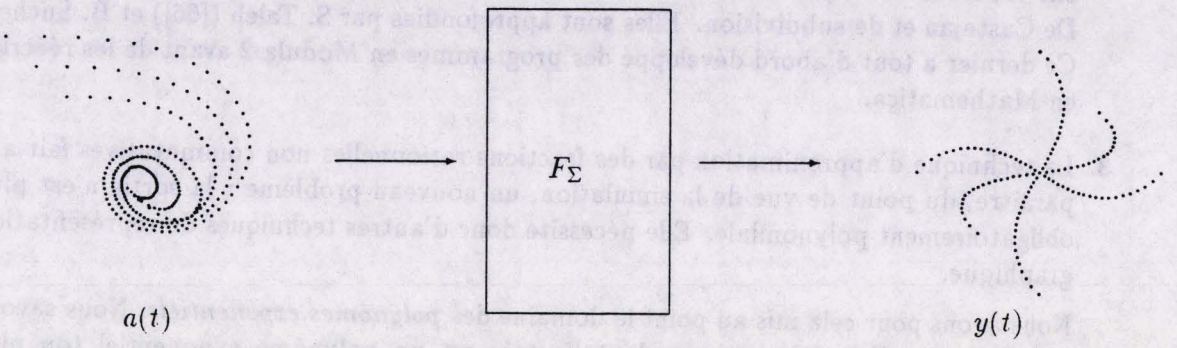
Le développement d'un simulateur graphique doit permettre de *montrer* des phénomènes de périodicité, de découplage, de non-interaction entre entrées et sorties, etc... qu'il est difficile de mettre en évidence sur des expressions symboliques souvent longues de plusieurs pages (voir [28]).

Rendre un simulateur efficace suppose évidemment qu'il soit capable d'"avaler" facilement les données (symboliques ou numériques) qu'on lui présente. Cela suppose également qu'il soit très souple d'utilisation, de manière à permettre des simulations répétées (ce qui facilite une étude approfondie).

Nous désirons pour cela un *simulateur interactif* pour étudier le comportement des systèmes dynamiques, c'est à dire un outil d'observation "rapide" de l'*évolution de la sortie* par des modifications "directes" des entrées.

Nous considérons un système dynamique comme un *transformateur de courbes* :

$$F_\Sigma : a(t) \mapsto y(t)$$



basé sur le codage symbolique de M. Fliess, la transformation d'évaluation et diverses techniques d'approximation. Chacune d'elles porte sur des classes d'objets (les courbes à représenter) différentes, ce qui nécessite autant de stratégies de simulation :

1. Pour la méthode d'approximation polynomiale, nous savons que des entrées polynomiales donnent des sorties du même type.

   Cette particularité nous permet d'utiliser les *courbes de Bézier*, introduites au début des années soixante par P. Bézier et P. De Casteljau (voir [3, 12]) pour des utilisation en C.A.O. (et notamment pour la définition de carrosseries de voitures chez Citroën). Bien que très connue, cette technique de représentation graphique des polynômes n'est pas disponible actuellement sur les systèmes de calculs formels (une implantation réalisée par B. Sucher est en cours sur Mathematica). L'utilisation de courbes de Bézier est actuellement possible sur quelques langages graphiques avec d'importantes limitations. En *PostScript* par exemple, on peut tracer une courbe par la donnée de ses points de contrôle, ceux-ci devant obligatoirement être au nombre de 3. On trouve également une utilisation des courbes de Bézier dans d'importants logiciels graphiques comme *Catia*. ceux-ci utilisent directement le calcul numérique, et perdent ainsi les possibilités que nous offrons de programmation au plus haut degré de généralité, comme d'utilisation de paramètres réels. Cette technique, que nous décrivons au chapitre 3, permet de satisfaire aux critères de souplesse et d'interactivité posés plus haut : nous définissons une courbe par un ensemble de *points de contrôle* représentatifs de son allure générale. L'expression *transformateur de courbes* prend alors tout son sens : nous modifions l'entrée par le simple déplacement d'un ou de plusieurs points de contrôle (avec Scratchpad par un changement de valeur numérique) et observons directement l'effet sur la sortie. Les expressions symboliques correspondantes restant toujours accessibles pour une étude

approfondie. Le tracé effectif est ensuite réalisé par les algorithmes de De Casteljau et de subdivision que nous décrivons également au chapitre 3. Cela présente deux avantages notables : d'une part, pour la même précision graphique, une diminution importante du nombre de valeurs significatives à calculer et d'autre part une représentation synthétique des courbes par leurs points de contrôle : le simulateur peut alors être considéré comme un *transformateur de points de contrôle.*

2. Les courbes de Bézier ne permettent pas de représenter les courbes sur des *intervalles infinis*, nous utilisons alors des *changements de variables homographiques et quadratiques* qui nous amènent aux *courbes rationnelles.* Nous adoptons, pour les représenter, la notion de *vecteurs massiques* qui remplacent les points de contrôle. Ces techniques ont été introduites par J.C. Fiorot et P. Jeannin ([19]) qui étendent les algorithmes de De Castejau et de subdivision. Elles sont approfondies par S. Taleb ([66]) et B. Sucher. Ce dernier a tout d'abord développé des programmes en Modula 2 avant de les réécrire en Mathematica.

3. La technique d'approximation par des fractions rationnelles non commutatives fait apparaître, du point de vue de la simulation, un nouveau problème : la sortie n'est plus obligatoirement polynomiale. Elle nécessite donc d'autres techniques de représentation graphique.

   Nous avons pour cela mis au point le domaine des *polynômes exponentiels.* Nous savons que la sortie d'un tel système, dont l'entrée est un polynôme exponentiel (ou plus simplement un polynôme) est aussi un polynôme exponentiel (cela revient à dire que la sortie, pour une entrée de type polynomial exponentiel, peut être approximée par des polynômes exponentiels). Nous couplons alors la technique de représentation des courbes de Bézier avec un calcul numérique des exponentielles. Ceci nous permet toutefois de garder une certaine maniabilité pour une étude répétée sur différentes valeurs des entrées.

4. La technique d'approximation structurelle nilpotente, enfin, est simulée de manière numérique par les méthodes très utilisées de Runge Kutta et d'Adams (que nous avons implantées en Scratchpad de manière compacte). Par cette technique, nous simulons des systèmes aux entrées de type quelconque. Nous donnons des exemples de cette simulation dans les chapitres 2 et 3. Elle ne permet pas, contrairement aux méthodes citées plus haut, de simulation interactive du comportement des entrée/ sortie, mais elle nous permet ici d'une part de simuler les résultats de calcul du "motion planning" et d'autre part de comparer trois techniques d'approximations et de simulations.

Nous utilisons ensuite les méthodes de calcul numérique pour tester les approximations précédemment décrites, et par conséquent pour les valider. Nous nous assurons ainsi de la fiabilité de notre simultateur et donnons de Scratchpad un large registre de ses possibilités.

Les limites actuelles de nos programmes de simulation sur Scratchpad (du moins pour la version 2.2 de novembre 1989) tiennent essentiellement en trois points :

1. Pour la simulation du comportement de systèmes dynamiques, la limitation du type des entrées à la classe des polynômes exponentiels ne nous permet pas de représenter de manière efficace les entrées trigonométriques. Ce problème est cependant surmontable, la seule limite réellement incontournable est celle des fonctions intégrables formellement.

2. En ce qui concerne le problème du "motion planning", et dans les exemples que nous développons en 2.3.4, le traitement des approximations nilpotentes d'ordre 4 engendre la résolution d'un système de 14 équations à 14 inconnues, ce que nous ne pouvons pas actuellement effectuer formellement (nous utilisons ici les bases de Gröbner ce qui nous limite, du moins sur un PC/RT aux systèmes de 7 équations à 7 inconnues), nous devons donc nous limiter ici aux approximations d'ordre 3. L'utilisation de techniques formelles plus efficaces ou de techniques numériques devrait à l'avenir nous permettre d'avancer plus loin.

3. La dernière difficulté importante vient de l'utilisation des nombres flottant pour la simulation graphique. Nous donnons page 53 un exemple de fraction rationnelle dont nous ne pouvons pas simuler l'évaluation. Notons cependant que l'expression symbolique de cette fraction est longue de plusieurs lignes (avec notamment un mot de 14 lettres), l'expression symbolique de son évaluation tenant sur plusieurs pages. Le calcul de la représentation graphique de cette courbe par la traduction des rationnels en flottants conduit à des accumulations d'erreurs donnant des courbes totalement incohérentes. Cette difficulté a d'ailleurs toutes les chances de résister aux augmentations de précision de la représentation en flottants. Elle pourrait tout aussi bien se produire sur des expressions symboliques de taille plus petite. Une étude spécifique de ce problème reste à mener.

Cependant, les limites de notre simulateur restent très larges, et permettent d'entrevoir son utilisation systématique pour l'étude de systèmes réels dans un avenir très proche. Ces études pourront se réaliser sur une classe élargie du type des entrées et devraient en outre contribuer à donner une idée intuitive des erreurs sémantiques des diverses méthodes d'approximation. L'utilisation d'un matériel plus efficace (*RS 6000*) et d'un logiciel plus au point (*Axiom*) devraient de plus augmenter les performances et permettre des simulations en temps réel. Le calcul formel est actuellement peu utilisé pour la simulation des systèmes non linéaires. Il se prête cependant bien à l'association calcul exact/calcul numérique, aussi nous pensons que c'est le cadre idéal pour la poursuite de nos travaux.

1

Scratchpad et son interface graphique

1.1 Introduction

Le but de ce travail est de donner des outils évolués de simulation de problèmes liés à l'automatique tels que le comportement entrée/sortie des systèmes dynamiques ou le "motion planning".

Le développement de tels outils est important car il offre, en plus des résultats exacts qu'il demeure indispensable d'obtenir, la possibilité d'*observer* et d'*analyser* succintement des phénomènes tels que les points singuliers, le découplage, etc... de manière quasi instantannée. Nous apportons ainsi la capacité de *montrer* à un système qui ne peut que *calculer*. Pour cela nous utilisons le calcul exact d'approximations polynomiales de taille arbitraire. Ainsi, pour approcher une série infinie, nous choisissons des polynômes de taille de plus en plus grande jusqu'à ne plus obtenir de différences en résultats de simulation. Une comparaison est ensuite faite avec une méthode de simulation numérique affin de vérifier les résultats obtenus. Nous effectuons d'autre part l'évaluation de fractions rationnelles dont le résultat symbolique est, dans certains cas que nous étudions, exact. Nous préciserons aussi par des comparaisons chiffrées les limites respectives des méthodes symboliques et numériques.

Pour cela, il est intéressant de disposer d'un système de calcul formel puissant. Nous pouvons alors effectuer, dans un premier temps, les calculs les plus coûteux de manière symbolique pour conserver les résultats exacts - ici c'est essentiellement le calcul de l'*évaluation des séries formelles* - que nous instancions dans un deuxième temps. Nous obtenons alors directement le comportement de systèmes dynamiques pour une classe définie d'entrées polynomiales avec un seul calcul d'évaluation. Tout ceci a été rendu possible par l'utilisation de Scratchpad (*Axiom*)qui est un langage de calcul formel très puissant malgré les limitations de la version dont nous avons pu disposer. Actuellement, la commercialisation de Scratchpad sous le nom d'*Axiom* par la société *NAG* devrait, nous l'espérons, résoudre ces problèmes.

La première version que nous avons utilisée ne contenait aucune possibilité graphique, ce qui nous a amené à développer notre propre interface. Nous avons dans un deuxième temps reçu une version plus récente (datant tout de même de 1989) disposant d'une interface graphique sophistiquée mais ne possédant pas toutes les primitives que nous avions implantées. Nous l'avons donc complétée de nos propres modules, joignant ainsi nos besoins spécifiques à une présentation plus agréable du graphisme. Nous n'avons pas pu disposer de la toute dernière version de Scratchpad, ni de la nouvelle version *Axiom*. Le premier prolongement de ce travail sera, évidemment, son transport sur *Axiom*, et donc la disponibilité de modules de simulation graphique pour l'étude des systèmes dynamiques non linéaires.

Ce travail se situe dans le cadre d'un projet d'équipe plus vaste comprenant l'identification, la réalisation, l'élimination et surtout le "motion planning" et les transformations d'évaluation que nous avons implantés en Scratchpad et qui sont partie prenante du simulateur que nous avons développé.

1.2 Les principaux systèmes de calcul formel

1.2.1 Un bref historique

C'est au début des années cinquante que remontent les premiers essais du calcul formel sur des programmes de dérivation formelle. Mais c'est dans les années soixante qu'apparaissent les premiers grands systèmes possédant un savoir faire véritablement étendu.

On peut alors distinguer *Macsyma* et *Reduce*, premiers systèmes a avoir remporté un succès auprès de la communauté scientifique et encore utilisés de nos jours. *Maple*, de taille plus modeste, peut être installé sur des machines plus petites, ce qui le rend attractif. Et enfin le dernier né, *Mathematica*, dont l'interface homme-machine très agréable lui vaut un succès indéniable.

Il ne faut bien entendu pas oublier Scratchpad, développé au début des années 70 à Yorktown sous le nom de Scratchpad I, puis de Scratchpad II dans les années 80 (avec des modifications notables au niveau du typage), il demeure, 20 ans plus tard, un produit expérimental mais qui sera dans un tout proche avenir commercialisé sous le nom d'*Axiom*.

1.2.2 Présentation des différents systèmes

Nous présentons ici brièvement les systèmes généraux de calcul formel les plus répandus.

Reduce a été développé à partir de 1966. Ecrit à partir d'un Lisp "minimal", il est portable sur de nombreuses machines, ce qui facilite sa diffusion et en fait un des langages de calcul formel les plus répandus. Reduce dispose de deux modes d'utilisation : un mode "algébrique", semblable à Macsyma, et un mode "symbolique", pour l'écriture d'expressions Lisp sans parenthèses et permet ainsi deux niveaux de programmation.

Macsyma, développé à partir de 1968 au MIT est le plus connu de tous. De taille imposante (120 000 lignes de code source), il possède plus de 500 fonctions commandant le système. C'est cependant un système qui commence à vieillir. Les structures de données sont de bas niveau, imposant l'utilisation des primitives Lisp. Les temps d'exécution sont élevés. Et enfin, Macsyma est difficilement portable, rendant sa diffusion et sa maintenance difficiles.

Maple a été développé à partir de 1980 à l'université de Waterloo (Canada). Il est de taille réduite par rapport aux "gros systèmes" que sont Reduce et Macsyma. Son noyau est écrit en *C*, ce qui le rend plus "rapide" que ses prédécesseurs et lui assure une portabilité importante. Il dispose de structures de données évoluées et d'une bibliothèque d'algorithmes écrits directement en Maple, permettant de "savoir ce qui se passe". Tout ceci fait de Maple un bon système de calcul formel.

Mathematica, developpé à partir de 1987, est un langage pour lequel la priorité est l'interface Homme-machine. Il permet une définition et une manipulation aisées des fonctions et dis-

pose d'une interface graphique très élaborée, deux points qui lui valent son succès. Il est écrit à partir d'une version orientée objet de *C* ce qui lui donne des performance de temps satisfaisantes. Il est cependant difficile de savoir comment sont écrits les algorithmes de bas niveaux, ce qui engendre une certaine réserve de la communauté scientifique pour son utilisation en tant que système de calcul formel "majeur".

Scratchpad, dont le développement sous la forme actuelle a débuté en 1981, possède des fonctionnalités bien supérieures aux autres systèmes. C'est d'abord un langage de programmation *fortement typé* permettant de développer des algorithmes à leur plus haut niveau de généralité (voir section 1.3). Il dispose d'une bibliothèque de modules écrits en Scratchpad qui sont la connaissance mathématique du système, structurée de manière à coller au modèle mathématique : ce n'est pas, comme on le trouve en d'autres langage, une simple collection de programmes. On trouve aussi un interpréteur permettant une utilisation interactive des connaissances couplé à un "parser" effectuant l'inférence de type c'est à dire, plus concrètement, la recherche du bon module pour l'exécution d'un opérateur. Développé à partir de Common-Lisp, Scratchpad n'est actuellement disponible que sur matériel IBM. Très longtemps confidentiel, il est de plus en plus utilisé par la communauté scientifique.

1.3 Les principes fondamentaux de Scratchpad

1.3.1 Introduction

Avant toute description de Scratchpad, nous précisons que nous ne disposons actuellement que de la *version 2.2* datant du *15 novembre 1989*. De nombreux changements sont intervenus depuis. Les principes fondammentaux sont probablement inchangés. Néanmoins, de nombreux domaines ont étés modifiés, voire supprimés dans les versions plus récentes. Les informations sur les modules de base de Scratchpad présentés ici sont donc à prendre avec prudence.

Les principes retenus par les concepteurs de Scratchpad sont directement issus de langages orientés objet tels que ML, ce qui le différencie des autres systèmes de calcul formel. L'idée est de coller le plus possible au modèle mathématique en utilisant les notions classiques des langages orientés objet (voir [8]) d'héritage , de généricité, de polymorphisme que nous décrivons ici. On construit ainsi chaque ensemble à partir d'autres plus généraux. Cette construction est illustrée sur la figure 1.2. Scratchpad se distingue néanmoins des langages tels que Smalltalk, car il ne dispose pas de possibilités de communication entre objets. Celles-ci sont remplacées par l'utilisation d'une base de donnée permettant d'effectuer de l'inférence de type et par l'existence d'un *Scope*, qui contient l'ensemble des objets visibles et donc utilisables au cours d'une session. Tout module non présent dans le *Scope* est chargé avant utilisation (voir l'exemple 1.3.1).

Scratchpad est ainsi plus qu'un simple système de calcul formel : c'est en même temps un langage élaboré permettant la création de modules compilés et un système utilisable en mode interprété, ce qu'on peut décrire par la figure 1.1

Cette conception, inédite pour le calcul formel, semble tout à fait adaptée, bien que pour des raisons purement informatiques, Scratchpad se détache parfois de la réalité mathématique (on observe par exemple sur la figure 1.2 page 28 une séparation totale entre les structures commutatives et non commutatives). Scratchpad, dans la version que nous possédons, recèle

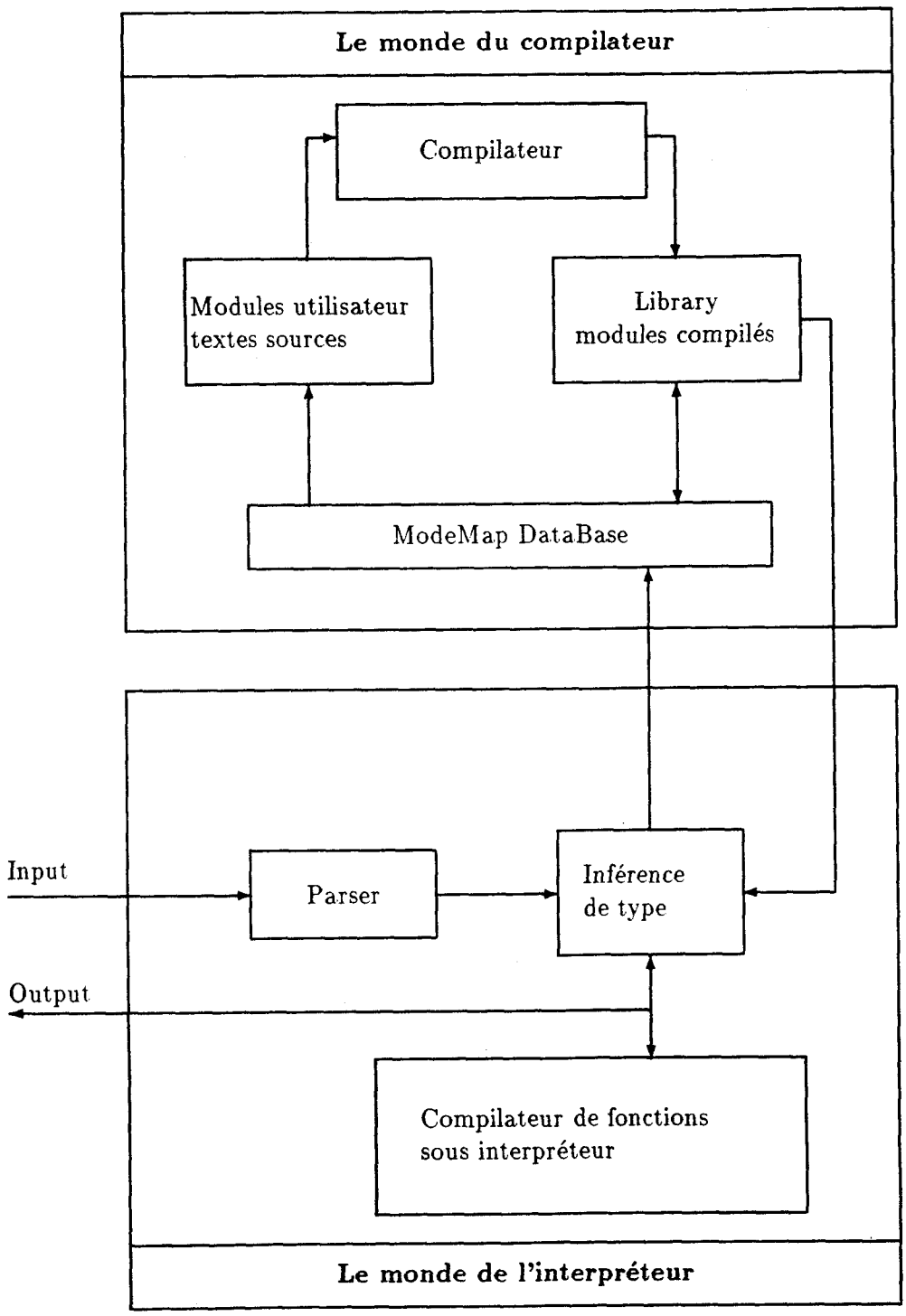


Figure 1.1 : Mode de fonctionnement de Scratchpad

également des insuffisances (c'est encore un logiciel expérimental) qui engendrent parfois de gros problèmes d'utilisation : manque de documentation, compilateur incapable de donner un message d'erreur précis, obligation de quitter l'environnement régulièrement (après quelques heures d'utilisation, Scratchpad ne fonctionne plus correctement).

Scratchpad reste cependant un élément incontournable du calcul formel dont on ne peut actuellement donner les limites. A ceci s'ajoute la cession à *NAG* des droits d'exploitation de Scratchpad qui devient ainsi *Axiom* et dont on peut espérer une finition plus satisfaisante, ainsi que le développement d'interfaces graphiques vers le calcul numérique et la simulation graphique.

A cette description, nous ajoutons ici une liste de fonctionnalités graphiques que nous aurions aimé trouver (pour effectuer nos simulations) et que nous avons dû implanter.

1.3.2 Quelques éléments de syntaxe

Avant d'entrer dans une description plus approfondie de l'univers de Scratchpad, il est important de connaître quelques éléments de sa syntaxe. Ces éléments sont indispensables à une bonne compréhension des différents exemples que nous présentons par la suite.

Il est à noter que Scratchpad permet l'affichage des expressions sous format TEX, par la commande ")set output tex on", que nous avons utilisée tout au long de ce document pour présenter nos résultats de manière plus agréable.

1.3.2.1 Affectations et règles de réécriture

L'affectation d'une variable à une valeur se fait classiquement par la commande ":=". La valeur de la variable ne pourra plus alors être modifiée que par une nouvelle affectation (exception est faite pour les listes, ou l'instruction ":=" correspond à une affectation de pointeurs).

Un autre mode d'affectation consiste à créer, sous interpréteur, des règles de réécriture en utilisant l'instruction "==" : après exécution de $a == b + c$, toute modification de b ou de c engendre un changement de valeur pour a.

```
(1) ->a == b+c
                                                    Type: Void
                                                   Time: 0 sec
(2) ->b:=1

   (2)  1
                                                Type: PositiveInteger
                                               Time: .1 (IN) = .1 sec
(3) ->c:=2

   (3)  2
                                                Type: PositiveInteger
                                                          Time: 0 sec
(4) ->a
```

```
   Compiling body of rule a to compute value of type PI

   (4)  3
                                                        Type: PI
                                    Time: .3 (IN) + .233 (OT) = .533 sec
(5) ->c:=5

   (5)  5
                                                  Type: PositiveInteger
                                                           Time: 0 sec
(6) ->a

   (6)  6
                                                  Type: PositiveInteger
                                                           Time: 0 sec
```

Il est également possible de passer des paramètres à a, on crée alors des fonction exactement comme en mode compilé :

```
(7) ->cube x == x**3
                                                        Type: Void
                                                 Time: .1 (OT) = .1 sec

(8) ->cube a
   Compiling function cube with type PI -> PI

   (8)  216
                                                  Type: PositiveInteger
                                    Time: .367 (IN) + .2 (OT) = .567 sec
```

Enfin, la dernière forme d'affectation couramment utilisée est la commande "==>" qui permet la création de macro-instructions.

1.3.2.2 Conversions et forçages de type

Scratchpad étant un langage fortement typé, il est nécessaire de construire des passerelles permettant les passages d'une donnée, lorsqu'ils sont possibles, d'un type vers un autre.

```
(1) ->a:= 1/2

        1
   (1)  -
        2
                                                  Type: RationalNumber
                                                 Time: .2 (IN) = .2 sec
```

Dans ce cas, on donne à a la valeur $1/2$ et le type "nombre rationnel" : Scratchpad donne par défaut aux entiers 1 et 2 le type "positiveInteger", ceux-ci sont ensuite convertis en entiers relatifs ("Integer"), la division de deux entiers est alors prise par défaut dans le domaine

des nombre rationnels (RationalNumber), le résultat est, bien sûr, un nombre rationnel. La variable *a* est représentée en mémoire par la liste (1,2). Pour toute modification du type de ce résultat, on utilise une commande de conversion, ce qui correspond en Scratchpad à l'instruction "coerce" et que l'on peut exécuter plus simplement en mode interprété avec l'instruction "::" :

```
(2) ->a::SF

   (2)  0.5
                                                    Type: SmallFloat
                                          Time: .1 (IN) = .1 sec
```

Nous n'avons modifié ici ni le type ni la valeur de *a*, nous avons simplemement affiché sa valeur sous la forme d'un nombre réel. Ces deux instructions peuvent s'écrire en une seule commande :

```
(3) ->b:= (1/2)::SF

   (3)  0.5
                                                    Type: SmallFloat
                                        Time: .233 (IN) = .233 sec
```

Ici, la variable *b* est de type "nombre flottant". Nous sommes arrivés à cela en plusieurs étapes successives : inférence de type et exécution du code de l'opérateur "/" pris dans le domaine des nombres rationnels. Execution ensuite du code de la fonction "coerce" implantée dans le domaine des SmallFloat. Cette méthode nécessite donc l'exécution de 2 fonctions Scratchpad. On peut, pour éviter cela, diriger Scratchpad vers le bon code à utiliser en précisant par exemple le type du résultat attendu. On force alors le type de la variable recevant le résultat par la commande ":".

```
(4) ->c:SF :=1/2

   (4)  0.5
                                                    Type: SmallFloat
                                          Time: .3 (IN) = .3 sec
```

On engendre ici un temps plus long d'inférence de type : les valeurs 1 et 2 vont être typées comme des entiers puis comme des SmallFloat, le code utilisé pour l'exécution de la division pourra alors être pris dans le domaine des SmallFloat.

1.3.2.3 Utilisation forcée d'un paquetage

Une dernière méthode enfin permet d'imposer le domaine ou paquetage dans lequel on prend le code à exécuter. On utilise pour cela le symbole "$" suivi du nom du domaine où l'on ira chercher le code :

```
(5) ->1 /$SF 2

   (5)  0.5
```

```
                                                    Type: SmallFloat
                                  Time: .133 (IN) + .1 (EV) = .233 sec

(6) ->1 /$QF(P RN) 2

        1
   (6)  -
        2
                                Type: QuotientField Polynomial RationalNumber
                                                    Time: .5 (IN) = .5 sec
```

Le symbole $ placé derrière l'opérateur ne laisse pas le choix à Scratchpad. Sur les deux exemples donnés ici, les valeurs 1 et 2 sont typées dans le premier cas comme des SmallFloat et dans le deuxième cas comme des QuotientField Polynomial RationalNumber. Les résultats fournis proviennent alors d'algorithmes différents. Les résultats, au niveau du type (c'est à dire de la représentation interne) comme de l'affichage, peuvent alors être très différents.

1.3.2.4 Manipulation de listes

Les listes sont une structure de données couramment utilisée en Scratchpad. Quelques indications sur les principales commandes de manipulation sont donc nécessaires à une bonne compréhension des exemples comme des programmes fournis en annexe. Le type liste est paramétré par le type de ses éléments. On peut ainsi manipuler des listes d'entiers, de polynômes, de fractions rationnelles, etc...

Commençons par la création : pour former une liste d'objets on utilise la primitive "construct", que l'on peut remplacer en mode interprété par deux crochets.

```
(1) ->liste := [1,2,3,4]

   (1)  [1,2,3,4]
                                          Type: List PositiveInteger
                                                    Time: 0 sec
```

Le premier élément porte l'indice 0. on pourra connaître la longueur de la liste par le symbôle # :

```
(2) ->#liste

   (2)  4
                                                    Type: PositiveInteger
                                  Time: .1 (IN) + .8 (OT) = .9 sec
```

ou la valeur de son plus grand indice :

```
(3) ->maxIndex liste

   (3)  3
                                                    Type: PositiveInteger
```

```
                                        Time: .1 (EV) + .167 (OT) = .267 sec
```

Les manipulations de listes sont réellement performantes lorsqu'elles sont faites de manière récursive (ceci est évidemment dû à la traduction de tout code Scratchpad en Lisp avant compilation). Deux primitives souvent utilisées sont alors *first* et *rest* qui sont respectivement équivalentes aux opérateurs *car* et *cdr* de Lisp. On peut alors écrire une fonction "moins" de la manière suivante :

```
(4) ->moins l ==
        if # l =0 then l
                  else cons(- first l, moins rest l)
                                                          Type: Void
                                                          Time: 0 sec

(5) ->moins liste
   Compiling function moins with type List PositiveInteger -> List
   Integer

   (5)  [- 1,- 2,- 3,- 4]
                                                    Type: List Integer
                                  Time: 1.033 (IN) + 1.1 (OT) = 2.133 sec
```

dont l'effet est de prendre l'opposé des éléments de la liste "l" passée en paramètre.

Scratchpad introduit également quelques opérateurs de manipulation à la syntaxe très concise qui nécessitent quelques éclaircissement :

Le symbole "!" tout d'abord, permet l'application d'une fonction à tous les éléments d'une liste (sans écriture explicite d'une boucle).

```
(6) -> - ! liste

   (6)  [- 1,- 2,- 3,- 4]
                                                    Type: List Integer
                                  Time: .433 (IN) + .067 (EV) = .5 sec
```

On applique ici l'opérateur "−" à tous les élément de `liste`. Les éléments de la liste des résultats sont ici d'un type différent de celui des données : ils sont du type retourné par l'opérateur "−".
Nous aurions pu, pour exécuter la même opération, utiliser une commande de balayage de la liste où la commande d'exécution d'une boucle apparaît explicitement :

```
(7) ->[-e for e in liste]

   (7)  [- 1,- 2,- 3,- 4]
                                                    Type: List Integer
                                                          Time: 0 sec
```

On peut aussi utiliser le symbole "/" précédé d'un opérateur binaire. Scratchpad va alors agir comme si on avait placé cet opérateur entre chacun des éléments de la liste passée en

paramètre.

```
(8) -> +/liste

   (8)  10
                                                    Type: PositiveInteger
                                                Time: .133 (IN) = .133 sec
```

On additionne ici tous les éléments de la liste `liste` entre eux.

1.3.2.5 La syntaxe du compilateur

Les éléments de syntaxe donnés précédemment sont utilisables pour la partie compilation autant que pour l'utilisation de Scratchpad en mode interprété. Ce que nous donnons ici correspond à des instructions plus généralement utilisées dans les programmes.

La création de fonctions compilées nécessite en premier lieu la donnée d'une *signature*, c'est à dire des ensembles de départ et d'arrivée qu'on exprime sous la forme : $f(T_1, T_2, \ldots, T_n) - > A$ où les T_i, ainsi que A, sont des domaines ou des catégories. A peut aussi être la réunion de plusieurs types : on l'exprime alors par $Union(A_1, A_2, \ldots, A_m)$. Le type du résultat fourni par f pourra ainsi être l'un quelconque des A_i.

Lors de l'écriture d'un *domaine*, on donne dans la *partie publique* l'ensemble des signatures des fonctions utilisables sous interpréteur. On réserve ensuite la *partie privée* à leur implantation. Tout domaine est ainsi exprimé sous la forme :

Nom_de_Domaine(paramètres): <Partie Publique> == <Partie Privée>

Pour clarifier l'écriture des modules, on utilise généralement la clause *where*. Nous décrivons son effet sur un exemple exécuté en mode interprété :

```
(1) ->a := x + y where (x:= 1; y:= x+1)

   (1)  3
                                                    Type: PositiveInteger
                                          Time: .267 (IN) + .1 (EV) = .367 sec
```

On effectue ici les affectations des variables x et y avant celle de a

En mode compilé, on utilise *where* pour séparer nettement l'écriture des parties publiques et privées. On obtient ainsi des expressions de la forme :

```
)abb domain VM VecteurMassique
  VecteurMassique(F:Field) : pub == priv where

    L ==> List

    pub == VectorSpace(F) with

      pp : (L F,F) -> $
```

```
        .
        .
priv == add

  V := L F
  PP := Record(cf : V,pds : F)
  Rep := Union(V,PP)

  vect(l : L F) == autoCoerce(l)
        .
        .
        .
```

L'exécution de la commande "pub == priv" est ainsi différée après l'exécution des instructions "pub ==" et "priv ==".

Nous voyons d'autre part sur cet exemple que le domaine des vecteurs massiques hérite des attributs et déclarations de fonctions stipulés dans la catégorie *VectorSpace*.

Nous notons ici un autre aspect particulier de la syntaxe : en Scratchpad, les structures de bloc sont matérialisées par l'indentation.

Notons enfin que la couche Lisp sur laquelle est construit Scratchpad influe beaucoup sur le style de programmation : Il est toujours intéressant de construire des programmes récursifs quand cela est possible. Nous utilisons alors fréquemment l'instruction d'échappement "=>" qui permet de sortir prématurément d'une fonction. Nous pouvons par exemple réécrire la fonction de parcours d'une liste donnée plus haut de la manière suivante :

```
moins l ==
  #l =0 => l
  cons(-first l,moins rest l)
```

Lorsque la condition "taille de l = 0" est vérifiée, l'instruction d'échappement est exécutée et la fonction est terminée. Dans le cas inverse, l'instruction suivante est exécutée (avec un appel récursif à "moins"). Remarquons également une particularité de la syntaxe de Scratchpad : lorsqu'une fonction ne prend qu'un seul argument, il n'est pas nécessaire d'utiliser de parenthèses. l'expression " moins rest l" est équivalente à "moins(rest(l))".

1.3.3 La programmation modulaire

En Scratchpad, chaque entité (entiers, listes, polynômes,...) est un objet, programmé et compilé séparément. Chaque nouveau module devient ainsi un apport, un ensemble de fonctionnalités dont tout utilisateur peut disposer. Tous ces modules sont chargés soit sur demande expresse de l'utilisateur (par la commande "`)load`"), soit parce qu'ils sont nécessaires à l'exécution d'une fonction. La taille de la bibliothèque est telle que ce mode de fonctionnement est incontournable. Cela engendre un inconvénient : la première exécution d'une opération entraine le chargement en mémoire centrale de tous les modules nécessaires à son aboutissement et le temps d'exécution en est considérablement augmenté. On obtient, par exemple, pour la première exécution de $cos(x)$ le chargement d'une quarantaine de modules.

Exemple 1.3.1 *Session Scratchpad*

```
(1) ->cos(x)
   (1)  cos(x)
                                        Type: FunctionalExpression Integer
                Time: 3.567 (IN) + 2.833 (EV) + 154.733 (OT) = 161.133 sec

(2) ->cos(x)

   (2)  cos(x)
                                        Type: FunctionalExpression Integer
                                                    Time: .2 (IN) = .2 sec
```

Le résultat fourni par Scratchpad n'est pas surprenant : la variable `x` *n'a pas de valeur numérique et l'expression* `cos(x)` *ne peut pas être simplifiée.*
La différence notable de temps de calcul entre la première et la deuxième exécution de cette même commande, due au chargement des modules (154.733 secondes) dans le Scope. Le temps d'inférence de type passe de 3.567 sec. à .2 sec. et le temps d'évaluation, c'est à dire de calcul proprement dit, passe de 2.833 sec. à une quantité négligeable (c'est à dire inférieure à 0.0005 sec) car Scratchpad possède une mémoire tampon : lors de la deuxième exécution de `cos(x)`*, il n'est pas tenté de simplification.*
La lenteur impressionnante des premières exécutions est inévitable. Tout utilisateur doit s'y habituer.

Les temps de calcul donnés par la suite ne sont jamais ceux (non significatifs) de la première exécution.

1.3.4 Catégories, domaines et paquetages

Scratchpad est un langage fortement typé. Le type de toute variable ou constante est le *domaine* auquel elle appartient. Un domaine est la donnée d'une partie publique, donnant les signatures et attributs des fonctions exportées, et d'une *partie privée*, qui comprend la représentation interne des données et l'implantation des fonctions, avec la possibilité d'implanter des fonctions locales encapsulées.

Le domaine est lui même l'élément d'une *catégorie* (qu'on appelle aussi sa *catégorie de base*) qui spécifie les signatures, c'est à dire l'ensemble des opérations, que pourront réaliser les domaines ainsi que les *attributs*, c'est à dire des propriétés de ces opérations (commutativité, distributivité,...).

Chaque catégorie possède des signatures et attributs hérités, ainsi que d'autres qui lui sont spécifiques.

Exemple 1.3.2 *La catégorie des monoïdes.*
Ce qui suit est l'aide en ligne fournie par Scratchpad.

```
->)show Monoid
 Monoid is a category constructor.
```

```
Abbreviation for Monoid is MONOID
This constructor is exposed in this frame.
Issue )edit xcatdef.spad to see algebra source code for MONOID

---------------------------- Operations ----------------------------

?*? : ($,$) -> $                    ?**? : ($,NonNegativeInteger) -> $
?=? : ($,$) -> Boolean              1 : () -> $
coerce : $ -> Expression
```

Le symbole **$** *joue un rôle particulier dans la définition des signatures : un domaine appartenant à la catégorie des monoïdes possède automatiquement les opérations données plus haut, le symbole* **$** *représentant un élément du domaine en question.*

En observant le programme Scratchpad de la catégorie des monoïdes, on constate l'intérêt de la notion d'héritage : seules les déclarations de l'unité et de la puissance (son code étant par ailleurs imposé) ont été nécessaires.

```
Monoid(): Category == SemiGroup with
    --operations
      1: constant ->  $                    ++ multiplicative identity
      "**": ($,NonNegativeInteger) -> $    ++ exponentiation
    add
      RS := RepeatedSquaring($)
      x:$ ** n:NonNegativeInteger ==
         n = 0 => 1
         expt(x,n:PositiveInteger)$RS
```

Ceci met en évidence l'intérêt de la programmation orientée objet pour la modélisation des structures mathématiques : une bonne connaissance des modules de Scratchpad permet d'en écrire de nouveaux en quelques lignes.

Dans l'exemple précédent, on note le rôle particulier de la fonction *coerce* : elle permet la conversion d'un type dans un autre. Tout élément de type A peut être converti dans le type B s'il existe un chemin allant de A en B par des fonctions *coerce*. Ceci revient, du point de vue informatique, à effectuer un changement de représentation interne. Ce chemin peut être imposé par l'utilisateur ou laissé à Scratchpad qui effectue de l'*inférence de type* en utilisant une base de données :

Exemple 1.3.3 *Exécutons* $1 + 1/2$. *Par défaut, le nombre 1 est un entier positif,* $1/2$ *est un nombre rationnel. Scratchpad ne dispose pas de l'opération* $+$ *avec la signature (entier positif, nombre rationnel), il va convertir 1 en suivant le chemin suivant :*

$$\textit{Entier Positif} \rightarrow \textit{Entier} \rightarrow \textit{Nombre rationnel}$$

et utiliser ensuite le code implanté pour l'addition de nombres rationnels.

Sur cet exemple, une remarque s'impose : en mathématiques, un nombre entier n est considéré comme un rationnel par identification à la fraction n/1, de même, avec Scratchpad, pour faire du nombre 3 un rationnel, il faut utiliser la fonction `coerce: I -> RN` qui permettra de représenter 3 par 3/1. Du point de vue informatique, les algorithmes appliqués aux entiers

seront différents de ceux appliqués aux nombres rationnels.

Enfin, dans la version dont nous disposons, la fonction `coerce: $ -> Expression` permet de programmer l'affichage à l'écran du résultat. Dans la version *Axiom*, le type *Expression* disparait, remplcé par les type *InputForm* et *OutputForm*.

Les domaines sont typés par des catégories. Ainsi on peut dire que le domaine des entiers appartient à la catégorie des anneaux, tout comme les domaines des matrices carrées ou des polynômes en une variable. Cela signifie que tous ces domaines ont en commun les propriétés et les opérations définies dans la catégorie des anneaux (opposé d'un élément, addition, élément neutre, etc...) dont les implantations sont, bien entendu, spécifiques à chacun. Ce typage des domaines permet d'écrire les algorithmes à leur plus haut niveau de généralité. On peut citer l'exemple du domaine des polynômes dont l'implantation est réalisée pour des coefficients dans un anneau quelconque. On peut ainsi travailler sur des polynômes à coefficients entiers, rationnels, matriciels, polynomiaux, etc...

L'utilisation des attributs est un des points forts de Scratchpad, elle permet de choisir un domaine plutôt qu'un autre, et donc l'algorithme le plus efficace, pour l'exécution d'une opération.

Exemple 1.3.4 *Les attributs*
L'aide en ligne de Scratchpad permet d'afficher les attributs d'une catégorie ou d'un domaine. Nous voyons ici l'ensemble des attributs du domaine des entiers :

```
(1) ->)sh Integer )attribut
 Integer is a domain constructor.
 Abbreviation for Integer is I
 This constructor is exposed in this frame.
 Issue )edit basicd2.spad to see algebra source code for I

------------------------------ Attributes ------------------------------

  canonicalUnitNormal                central
  noZeroDivs                         unitsKnown
  commutative(*)                     orderPreserve(*)
  orderPreserve(+)                   total(<)
```

Ces attributs sont ceux des diverses catégories auxquelles appartient le domaine des entiers.

Les signatures et attributs, c'est à dire la catégorie du domaine, forment la partie publique, visible pour l'utilisateur. L'implantation et la représentation interne forment la partie privée, encapsulée dans le domaine.

Les *paquetages* sont une forme affaiblie de domaines. Sans représentation interne, ils ne forment pas des types mais simplement des regroupements d'opérations qu'on ne veut pas implanter dans un domaine particulier. *Dans les versions récentes de Scratchpad, l'implantation d'opérations dans les paquetages a été supprimée.*

1.3.5 Héritage

Le mécanisme d'héritage est essentiel en Scratchpad. Il permet l'écriture de modules en utilisant les propriétés, signatures ou implantations des autres. On construit ainsi une nouvelle catégorie par héritage des signatures et attributs d'une ou de plusieurs catégories plus générales et en lui ajoutant éventuellement des signatures et attributs qui lui seront propres. On utilise à ce niveau un *héritage multiple* : Tout domaine peut hériter de plusieurs catégories (par utilisation de "Join").

De même, la construction d'un domaine pourra se faire par héritage d'un autre domaine dont on utilisera la représentation interne et certaines implantations. On utilise alors l'*héritage simple* : un héritage multiple à ce niveau engendrerait des conflits (on pourrait par exemple hériter de deux représentations internes différentes, ce qui n'a pas de sens).

On peut ainsi hériter aussi bien des parties publiques que privées. Ceci permet d'alléger considérablement l'écriture des domaines mais oblige le programmeur à une connaissance approfondie de la bibliothèque Scratchpad.

Cette structure explique le nombre parfois très important de modules chargés lors de la première exécution d'une fonction : Le domaine sollicité ordonne le chargement de tous les modules nécessaires au fonctionnement de toutes ses opérations, ceux ci font de même jusqu'à ce que tous soient opérationnels. L'héritage est un principe intéressant dont il faut se servir. Il constitue une des difficultés de la programmation en Scratchpad : choisir les ancètres les plus appropriés afin de minimiser la programmation et de la rendre la plus efficace possible.

Exemple 1.3.5 *Le graphe d'héritage de la catégorie des algèbres.*

On voit sur la figure 1.2 l'architecture mathématique implantée dans Scratchpad pour construire les algèbres sur un anneau quelconque.

Notons la différence faite entre ensembles commutatifs et non commutatifs avec une remarque sur la notation des opérateurs en Scratchpad, qui impose l'utilisation du "+" pour les opérations commutatives et du "" pour les opérations non commutatives. Ainsi, un groupe abélien n'est pas un descendant de la catégorie des groupes : le premier dispose du "+" et le second du "*".*

1.3.6 Polymorphisme

Comme tous les langages orientés objets, Scratchpad dispose de la notion de polymorphisme : une même valeur peut appartenir à différents types (elle peut être représentée en mémoire de diverses manières), un même opérateur peut s'appliquer à différents types (le type des données indiquera l'algorithme à utiliser). Cette notion n'est pas neuve. Ce qui caractérise l'approche orientée objet est son utilisation systématique : en Scratchpad, l'entité 0 peut être un entier, un polynôme, un nombre complexe, une matrice,... L'opérateur "*" s'applique aux nombres rationnels, aux fonctions trigonométriques, aux listes,...

On distingue alors en Scratchpad deux types de polymorphisme ([4]) : celui cité précédemment, engendrant une *surcharge d'opérateur*, et le *polymorphisme paramétré* (ou généricité), illustré sur l'exemple 1.3.6 et dont le choix du code appliqué n'apparait qu'à l'exécution.

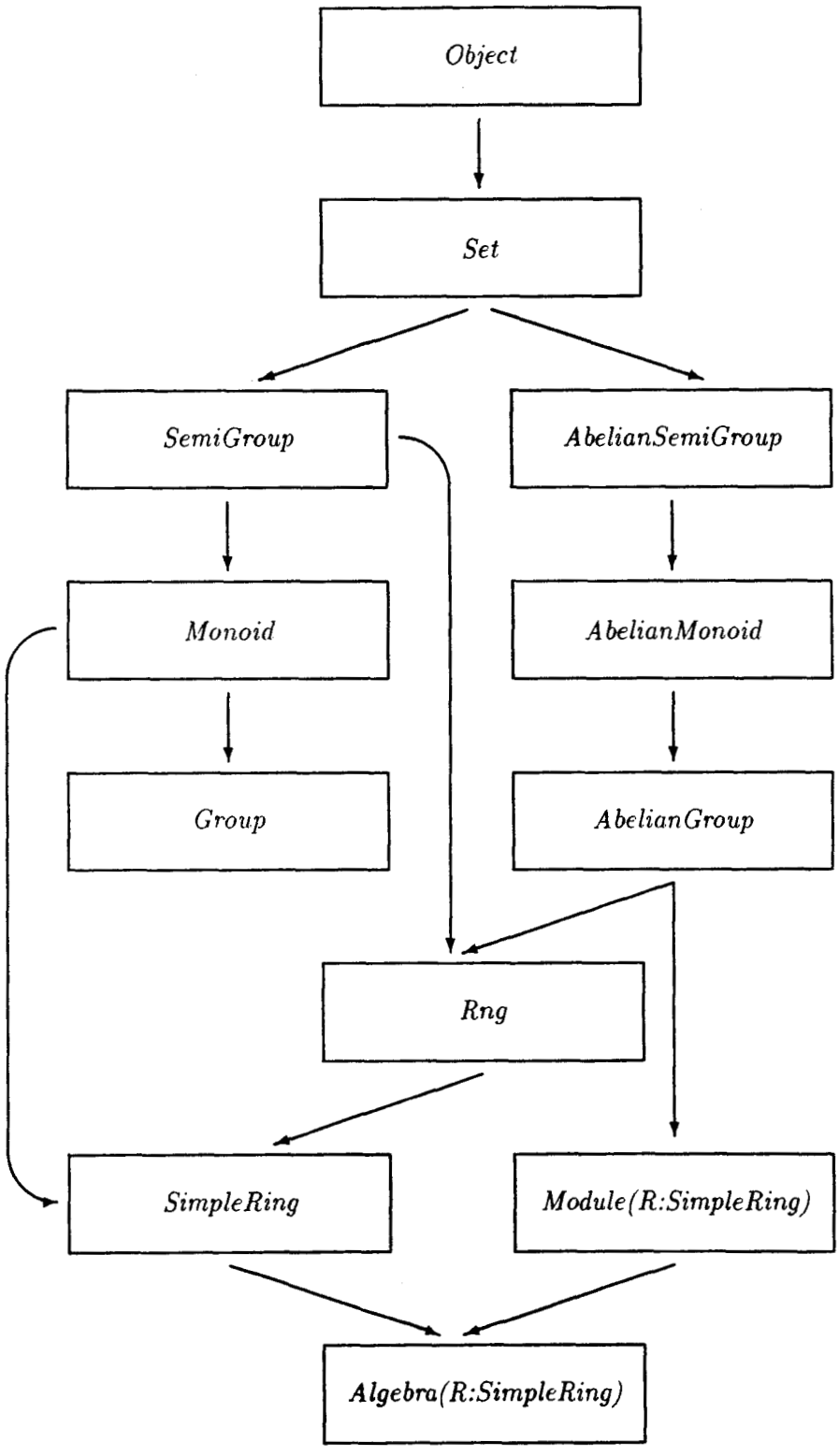


Figure 1.2 : Graphe d'héritage de la catégorie des Algèbres

1.3.7 Généricité

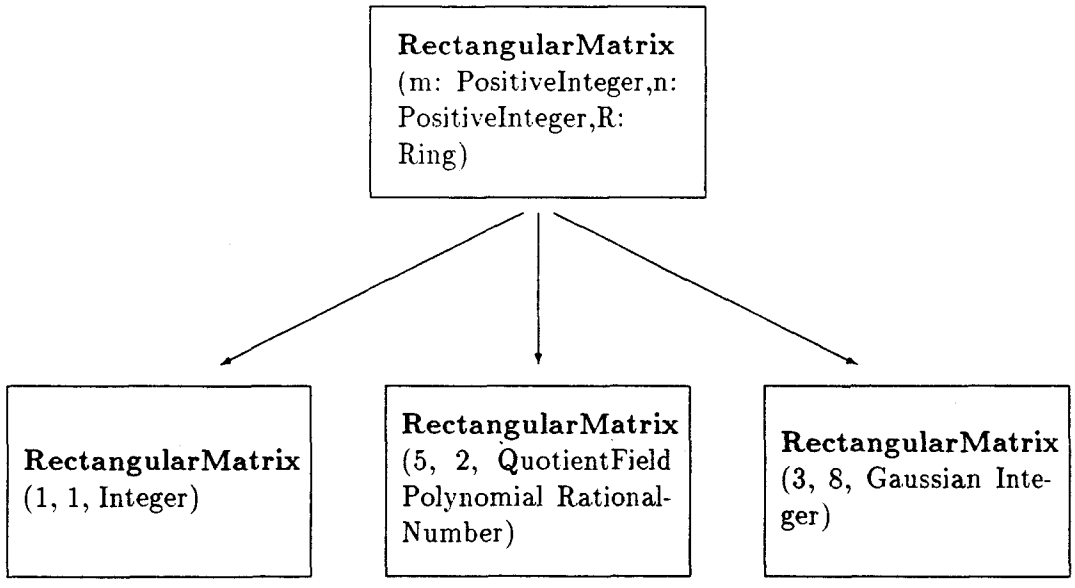
Cette fonctionnalité de Scratchpad est tout à fait intéressante car elle permet l'écriture de programmes en toute généralité. Le principe de la généricité est de donner en paramètres d'un domaine ou d'une catégorie d'autres domaines ou catégories sur lesquels ses éléments sont construits.
Citons en exemple le constructeur de domaine des *matrices rectangulaires*. Dans l'implantation, du moins dans la version dont nous disposons actuellement, on précise simplement que les éléments d'une matrice rectangulaire $m \times n$ sont pris dans un anneau R : *RectangularMatrix(m: PositiveInteger,n: PositiveInteger,R: Ring)*. On utilisera alors ce constructeur pour créer différents domaines de matrices rectangulaires.

```
(1) ->MatriceRect1 := RectangularMatrix(2,3,RN)

   (1)  RectangularMatrix(2,3,RationalNumber)
                                                        Type: Domain
                                    Time: .1 (IN) + .167 (OT) = .267 sec

(2) ->MatriceRect2 := RectangularMatrix(9,4,UP(t,I))

   (2)  RectangularMatrix(9,4,UnivariatePoly(t,Integer))
                                                        Type: Domain
                                      Time: .1 (IN) + .1 (OT) = .2 sec
```



Observons la somme de deux matrices rectangulaires : on additionne les éléments de l'anneau en utilisant la somme implantée dans l'anneau spécifié en paramètre.

Exemple 1.3.6 *Implantation de l'addition de deux matrices* `x` *et* `y` *:*
La signature : `? + ? : ($,$) -> $`
Le texte source :

```
x+y == [[x.i.j+y.i.j for j in 0..maxcol x] for i in 0..maxrow x]
```

Ce programme est appliqué quelque soit l'anneau R dans lequel sont pris les éléments `x.i.j` *et* `y.i.j`*, et c'est l'addition implantée dans R qui sera effectuée.*

Notons que le paramétrage peut être beaucoup plus subtil : les domaines générés par un même constructeur ne disposent pas automatiquement des mêmes fonctions. Nous en donnons ici une illustration.

Exemple 1.3.7 *Nous donnons ici le dédut de l'implantation du constructeur de domaines PolRing :*

```
PolRing(R:Ring,E:OrderedAbelianMonoid): T == C
 where
  T == GeneralPolynomial(R,E) with
    --operations
       if R has IntegralDomain then "exquo": ($,R) -> Union($, "failed")
       "*": $ -> NonNegativeInteger

    --assertions
       if R has unitsKnown then unitsKnown
       if R has commutative("*") then commutative("*")
       if R has IntegralDomain then IntegralDomain
       if R has canonicalUnitNormal then canonicalUnitNormal
  C == FreeModule(R,E) add
```

On voit ici que la fonction "exquo" ne sera utilisable pour le domaine PolRing(R1,E1) que si l'anneau R1 passé en paramètre est intègre. De même, les domaines générés à partir de PolRing n'auront pas tous les mêmes attributs, également conditionnés par les propriétés de l'anneau. :

```
(1) ->PR1 := PolRing(RN,I)

   (1)  PolRing(RationalNumber,Integer)
                                                        Type: Domain
                                                         Time: 0 sec


(2) ->PR2 := PolRing(SM(4,RN),I)

   (2)  PolRing(SquareMatrix(4,RationalNumber),Integer)
                                                        Type: Domain
                                                         Time: 0 sec


(3) ->PR1 has commutative("*")

   (3)  true
```

```
                                                    Type: Boolean
                                        Time: .1 (OT) = .1 sec

(4) ->PR2 has commutative("*")

   (4)  false
                                                    Type: Boolean
                                                      Time: 0 sec
```

Cela suppose alors que tous les modules soient écrits avec beaucoup de soin : l'oubli d'un attribut ou le mauvais choix des ancètres dans l'écriture d'un domaine le rendra inaccessible à d'autres.

La programmation en Scratchpad demande donc une grande discipline et le respect strict de règles de génie logiciel.

1.3.8 Une illustration : Les polynômes non commutatifs en Scratchpad

Les polynômes non commutatifs sont une illustation intéressante de la programmation en Scratchpad : ils mettent en oeuvre l'ensemble des mécanismes décrits plus haut. Leur calcul est très rapide et permet des simulations répétées et intensives.

Nous utilisons ces modules, implantés par M. Petitot ([53]), pour coder les séries génératrices polynomiales de manière à effectuer récursivement la transformation d'évaluation (nous utilisons en particulier de manière directe le domaine *XRecursivePolynomials*).

Le graphe d'héritage des polynômes non commutatifs est donné par la figure 1.3. On observe ici deux modes différents de représentations internes ($XDPOLY$ et $XRPOLY$) impliquant deux implantations différentes des mêmes algorithmes et des performances largement supérieures pour la représentation récursive. On remarque l'importance de la généricité qui permet de programmer les algorithmes à leur plus haut niveau de généralité.

1.4 Intérêt et réalisation d'une première interface graphique

1.4.1 L'interprétation des résultats

L'utilisation du calcul formel permet l'obtention de résultats symboliques et de calculs exacts, ce qui est indispensable pour l'étude des systèmes dynamiques (voir [28, 31, 38, 52]). Néanmoins, l'étude de systèmes réels et donc de taille raisonnable génère des résultats tenant généralement sur plusieurs pages de listing et l'interprétation de leur signification mathématique ou physique devient un problème insurmontable sans outil approprié (voir [38, 28]).

La simulation graphique est donc une aide précieuse, qui devrait permettre par exemple de mettre directement en évidence des phénomènes de découplage, de non interaction entre entrées et sorties, de singularités.

Les logiciels de calcul formel dont nous disposons, Scratchpad et Macsyma permettent l'ob-

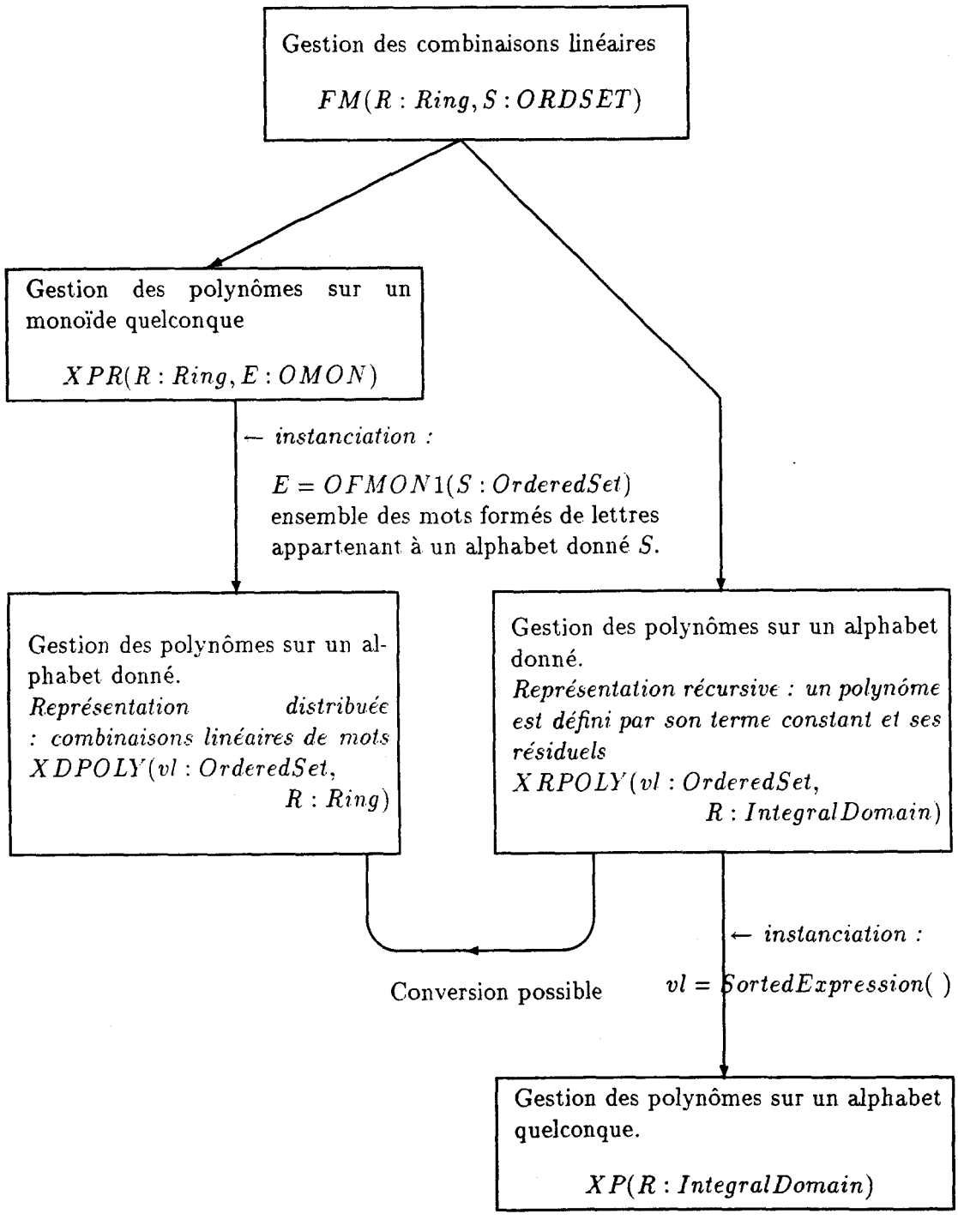


Figure 1.3 : Architecture de l'implantation des polynômes non commutatifs

tention de résultats symboliques de taille généralement importante, et dont nous n'avons à priori aucune idée du comportement. Prenons par exemple l'équation différentielle $\dot{y} = u + y^2$ (décrite en 2.1.4.4), on la code par la série sur deux lettres $S = z_1 + (S \sqcup S)z_0$. Nous pouvons calculer des approximations :

$$\begin{aligned} S_1 &= z_1 \\ S_k &= S_1 + (S_{k-1} \sqcup S_{k-1})z_0 \end{aligned}$$

L'évaluation de la série S_3 pour une entrée polynomiale de degré 2 donne un polynôme de degré 21. Un tel résultat brut est difficilement exploitable "à la main" : combien de fois s'annulle-t-il? quel est sont comportement à l'infini? est-il l'approximation polynomiale d'une fonction périodique? voilà un petit échantillon de questions auxquelles on voudrait pouvoir répondre en un instant, de manière à savoir vers quels aspects intéressants on peut orienter les calculs.

Nous avons donc décidé de créer un outil graphique permettant par exemple pour l'étude de la série donnée ci-dessus de se faire une d'idée de l'ordre d'approximation qui est satisfaisant, c'est à dire, par exemple, à partir duquel les courbes de deux approximations seront, ou presque, confondues en fonction de l'entrée u et de l'intervalle de temps sur lequel on l'étudie. On vérifie alors, en utilisant une méthode numérique, la validité de l'approximation. Nous aurions pu, pour cela, utiliser des logiciels graphiques déjà existants, il aurait alors fallu, étant donnés les moyens actuellement à notre disposition, prendre les résultats symboliques, les transformer en un format lisible par ces logiciels, transférer les données, et enfin, tracer la courbe. Nous perdons alors un temps considérable, et la connaissance de l'approximation acceptable devient extrèmement fastidieuse, sachant qu'on ne connait pas l'erreur théorique de toutes les méthodes d'approximation.

Ce n'est cependant pas le seul intérêt de la simulation : elle permet d'observer directement l'existence des phénomènes (stabilité,singularités, etc.) et, dans tous les cas, de donner à l'utilisateur une idée précise du comportement de la sortie afin d'en faciliter ou d'en orienter une analyse plus approfondie.

1.4.2 La création d'une interface

Nous plaçant dans l'optique du calcul exact, c'est à Scratchpad que nous avons demandé l'essentiel des calculs. Pour simuler le comportement entrée/sortie polynomial, nous avons décidé de l'utilisation de courbes de Bézier dont les principes et l'efficacité sont décrits dans le chapitre 3. Scratchpad génère ainsi des fichiers de coordonnées de points, permettant également la constitution d'une bibliothèque de courbes que nous affichons dans une fenêtre X-Window par un programme C. Les calculs de Scratchpad sur les nombres rationnels permettent une précision quasi infinie, assurant l'exactitude du tracé par la suppression de toute propagation d'erreur comme c'est le cas lors de calculs sur les nombres flottants.

Nous avons utilisé les propriétés de généricité ([49, 51]) de Scratchpad pour implanter les algorithmes de tracé de courbes fonctionnant aussi bien sur des nombres rationnels que sur des flottants de taille quelconque. Nous donnons ainsi à l'utilisateur le choix entre un tracé rapide, de l'ordre de quelques secondes, et un tracé exact, jusqu'à 100 fois plus long. Cette différence énorme des temps s'explique par le calcul de PGCD qu'effectue Scratchpad pour simplifier chaque nouveau nombre rationnel qu'il détermine. Nous avons également défini un format d'échange des données entre Scratchpad et le programme C (donné en annexe) effectuant le tracé.

1.5 Nouvelle version de Scratchpad et son interface

1.5.1 Le graphique sous Scratchpad

Nous avons reçu en décembre 1990 une nouvelle version de Scratchpad possédant des primitives graphiques intégrées. Les principes sont les mêmes que ceux que nous avions adoptés pour ce qui est de la communication : création d'un fichier temporaire et possibilité de sauvegarde de l'image. Cet outil créé dans les laboratoire d'IBM par une équipe de spécialistes du graphisme est plus raffiné que le nôtre : possibilité de zooms, choix des couleurs, etc. L'utilisation de notre première interface a donc pris moins d'importance.

Cependant, et bien que les primitives graphiques soient nombreuses, nous n'avons pas trouvé l'implantation des courbes de Bézier, aussi avons nous ajouté nos propres implantations à celles déjà en place. De plus, nous ne possédons pas de logiciel capable de reproduire sur imprimante l'image sauvegardée au format de Scratchpad (sous forme d'un répertoire contenant 3 fichiers de données). Il est à noter que la fonction Scratchpad de restauration de l'image à l'écran n'est pas non plus implantée dans la version que nous possédons.

Nous avons donc également développé dans le paquetage *TexDraw* les primitives fournissant un fichier source *Latex* donc aisément insérable dans un texte (voir entre autres les exemples 3.10 page 109 et 3.11 page 111 du chapitre 3) donnant une représentation point par point des courbes. Nous avons ensuite écrit les primitives donnant un fichier de points utilisable par un programme *Postscript* (décrit en annexe) permettant de reproduire la courbe habillée (axes et échelle) soit à l'écran, soit sur imprimante. Notons que ce principe permet également une insertion facile dans un texte, ce qui est une fonctionnalité agréable du simulateur.

1.5.2 Les fonctions à implanter

Nos besoins en graphisme sont de différentes natures.
Tout d'abord la visualisation de courbes polynomiales avec un maximum d'informations, ce que les courbes de Bézier sont en mesure de nous fournir. Nous avons donc développé dans les paquetages *BCurve* et *RationalCurve* les primitives de calcul des points des courbes de Bézier polynomiales et rationnelles dont les résultats sont repris par les paquetages *BezierDraw2D* et *RationalDraw2D* pour une visualisation à l'écran. Nous avons également développé le paquetage *GraphicXEvaluationPackage* qui permet la visualisation à l'écran des courbes d'entrée et de sortie d'un système dynamique par l'utilisation d'une seule commande. Nous donnons sur la figure 1.4 page 36 les relations entre les différents modules ammenant à la visualisation des courbes. Nous donnons ci-dessous la fonction des différents modules décrits sur cette figure.

UnivariateBernsteinPolynomial : Gestion des polynômes exprimés en base de Bernstein. Nous en donnons une description détaillée en 3.2.2 page 80

BCurve : Création de courbes de Bézier à partir d'une liste de polynômes exprimés en base de Bernstein ou en base canonique.

XRecursivePolynomial : Polynômes non commutatifs représentés de manière récursive. (voir M. Petitot,[56])

XEvaluationPackage1 : Evaluation d'une série, donnée ici par un polynôme non commutatif, pour des entrées, données ici par des polynômes en base de Bernstein ou, plus directement,

des courbes de Bézier.

GraphicXEvaluationPackage : paquetage de simulation graphique, visualisant à l'écran les courbes d'entrée et de sortie pour une série donnée.

BezierDraw2D : Calcul effectif par subdivision des points nécessaires au tracé de courbes de Bézier.

VecteurMassique : gestion de vecteurs massiques, exprimés sous forme de vecteurs ou de points pondérés. Nous en donnons une description détaillée en 3.4.1 page 96

RationalCurve : Création de courbes rationnelles à partir de vecteurs massiques ou de courbes de Bézier.

Rationaldraw2D : Calcul par subdivision des points nécessaires au tracé de courbes rationnelles.

ParametrizationPackage : changement de paramètres homographiques et quadratiques pour l'expression de courbes rationnelles sur des intervalles de temps infinis.

TwoDimensionalViewport : Paquetage Scratchpad pour la visualisation de courbes à l'écran.

La généralisation du type d'entrées aux polynômes exponentiels nous a amené à développer le domaine spécifique *PolynomialExponential* dont nous décrivons l'interaction avec les autres modules sur la figure 1.5. Les bonnes propriétés de celui-ci ainsi que le respect (facilité par Scratchpad) des règles du génie logiciel nous ont permis de l'inclure sans difficulté aux modules déjà présents.

Enfin, pour pouvoir comparer directement sur un même graphique les résultats des algorithmes du simulateur avec ceux utilisés en calcul numérique, nous avons implanté dans le paquetage *NumericResolutionofEquaDif* les algorithmes classiques d'intégration de *Runge-Kutta* et *Adams* fournissant des résultats aux formats utilisables par nos fonctions graphiques. Ceci n'exclut évidemment pas la possibilité d'appels de fonctions numériques performantes comme celles de la bibliothèque *NAG* par exemple. La création en Scratchpad de fonctions numériques spécifiques nous a simplement permis, en l'abscence de toute interface facile à utiliser ou à construire, de simuler par des méthodes différentes le comportement des systèmes dynamiques sans changer d'environnement de travail.

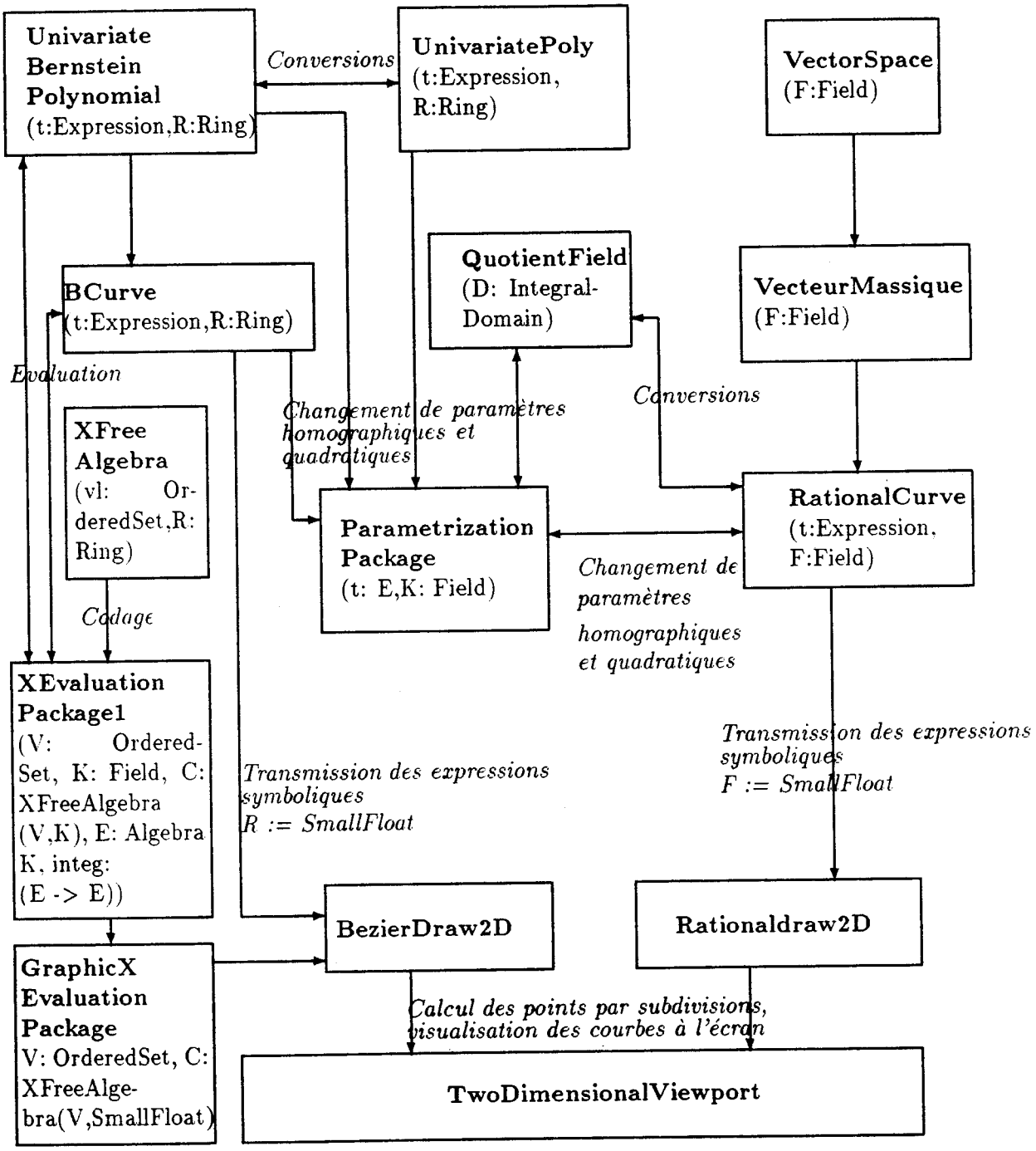


Figure 1.4 : Combinaison des modules pour la simulation graphique

UnivariateBernsteinPolynomial : Gestion des polynômes exprimés en base de Bernstein.
TwoDimensionalViewport : Domaine de Scratchpad pour la visualisation de courbes à l'écran.
XEvaluationPackage1 : Evaluation d'une série, donnée ici par un polynôme non commutatif, pour des entrées, données ici par des polynômes en base de Bernstein ou, plus directement, des courbes de Bézier.
PolynomialExponential : Gestion des polynômes exponentiels de la forme: $\sum_{k=0}^{n} P_k e^{kt}$. Nous en donnons des exemples d'utilisation en 2.2.3.1 page 59 et une description plus détaillée en 3.3 page 92
PolExpoCoercionPackage : Paquetage de conversion des polynômes utilisés dans le domaine *PolynomialExponential*, conversions de la base canonique vers la base de Bernstein et inversement.
FunctionalExpression : Domaine de Scratchpad sur les expressions fonctionnelles, convertibles en fonctions par utilisation du domaine **FunctionSpaceToCompiledFunction**, que l'on peut alors transmettre au paquetage **NumericalResolutionofEquaDif** pour la simulation numérique.

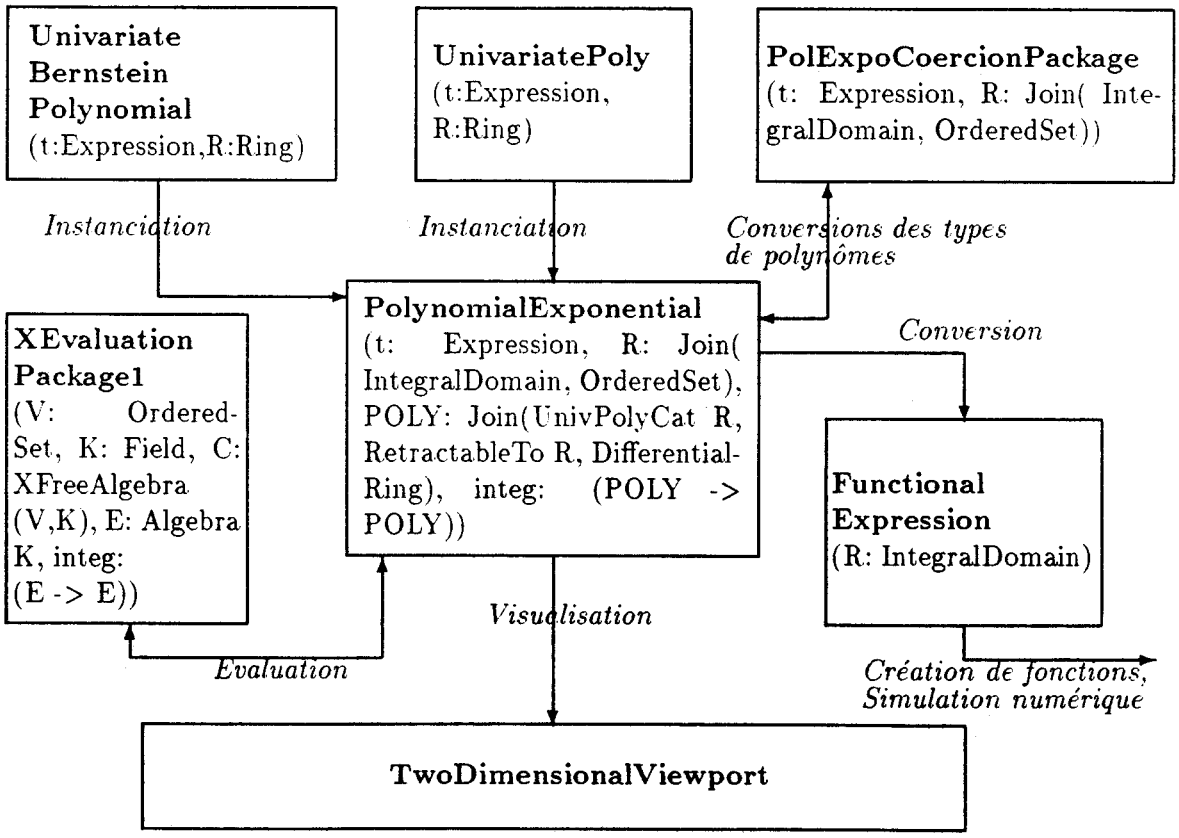


Figure 1.5 : Interactions entres polynômes et polynômes-exponentiels

2

Les systèmes dynamiques

Introduction

Dans ce chapitre, nous rappelons tout d'abord l'ensemble des outils algébriques et combinatoires nécessaires au codage des systèmes dynamiques par des séries formelles en variables non commutatives. Nous donnons également les domaines implantés en Scratchpad par M. Petitot ([53]) et sur lesquels nous basons une partie de notre travail.

Nous nous intéressons également au calcul dans l'algèbre de Lie libre avec la construction des bases de Lyndon (on peut pour plus ample information se référer au ouvrages généraux [50, 67, 15, 5]) dont nous donnons les grandes lignes de l'implantation effectuée par M. Petitot ([33, 54, 34]) en Scratchpad. On peut également se référer aux travaux de P.V. Koseleff ([36]).

Nous rappelons ensuite la définition et les propriétés de la transformation d'évaluation (voir [28]), avec le paquetage d'évaluation implanté en Scratchpad qui lui correspond. Son rôle est de calculer la sortie d'un système dynamique, en fonction d'entrées de type polynomial (en base canonique ou en base de Bernstein) ou polynomial exponentiel (à coefficients rationnels, réels ou complexes), donnée par diverses méthodes d'approximation (polynomiale, rationnelle, structurelle nilpotente). Ces résultats d'approximation seront utilisés par notre simulateur, dont nous détaillons les fonctionnalités dans le chapitre suivant.

Enfin, en utilisant la factorisation de l'opérateur de transport, nous abordons le problème du "motion-planning" (voir [31, 65]) dont le but est de calculer les commandes, polynomiales ou constantes par morceaux, pour conduire un mobile d'un point à un autre de l'espace. Nous simulons la trajectoire ainsi obtenue.

2.1 Outils algébriques et combinatoires

2.1.1 Alphabet et mots

Soit $Z = \{z_0, ..., z_n\}$ un ensemble ordonné fini non vide qu'on appelle *alphabet*. Les éléments de Z sont appelés les *lettres*. Une suite finie de lettres, $u = z_{i_1}...z_{i_k}$ est appelée un *mot*. On note Z^* l'ensemble de tous les mots construits sur l'alphabet Z. Le mot qui ne contient aucune lettre est appelé le mot vide, on le note ε.

Soient $u = z_{i_1}...z_{i_k}$ et $v = z_{j_1}...z_{j_l}$ deux mots de Z^*. On définit le produit (ou concaténation) de u et v par :

$$uv = z_{i_1}...z_{i_k}z_{j_1}...z_{j_l}$$

Ce produit est évidemment non commutatif.
Z^* muni de l'opération de concaténation est un monoïde libre.

La longueur d'un mot w de Z^*, notée $\mid w \mid$, est le nombre de lettres qui le constituent. Ainsi, $\mid \varepsilon \mid = 0$. Pour $u = z_{i_1} ... z_{i_k}$, on a $\mid u \mid = k$.

On note $\mid v \mid_z$ le nombre d'occurences de la lettre z dans le mot v.

On dit que le mot u_1 est *facteur* du mot u_2 si et seulement si :

$$\exists v, w \in Z^* \text{ tels que } u_2 = v u_1 w$$

Si $v = \varepsilon$ alors u_1 est appelé *facteur gauche* (propre si $w \neq \varepsilon$)
Si $w = \varepsilon$ alors u_1 est appelé *facteur droit* (propre si $v \neq \varepsilon$)

Deux mots u_1 et u_2 de Z^* sont dits conjugués si et seulement si :

$$\exists v_1, v_2 \in Z^* \text{ tels que } u_1 = v_1 v_2 \text{ et } u_2 = v_2 v_1$$

2.1.2 Ordre lexicographique sur Z^*

On définit un ordre total sur Z^* de la manière suivante :
$\forall u_1, u_2 \in Z^*$, $u_1 < u_2$ si et seulement si :

$$\exists w \neq \varepsilon \quad \text{tel que} \quad u_1 w = u_2$$

ou

$$\exists v_1, v_2, v_3 \in Z^*, \exists x, y \in Z \quad \text{tels que} \quad u_1 = v_1 x v_2,\ u_2 = v_1 y v_3 \text{ et } x < y$$

Ceci nous donne l'ordre lexicographique.

On définit également sur Z^* l'*ordre lexicographique inverse* :
Soient u_1 et u_2 deux mots de Z^*, $u_1 < u_2$ si et seulement si :

$$\exists v \in Z^* \text{ tel que } u_2 = v u_1$$

ou

$$\exists x, y \in Z,\ v_1, v_2, w \in Z^* \text{ tels que } u_1 = v_1 x w,\ u_2 = v_2 x w \text{ avec } x < y$$

2.1.3 Mots de Lyndon et leur miroir

Definition 2.1.1 *Un mot u de Z^* est un mot de Lyndon s'il vérifie l'une des deux conditions équivalentes suivantes :*

1. *il est strictement plus petit que tous ses conjugués propres,*
2. *il est strictement plus petit que tous ses facteurs droits propres.*

Cette définition s'entend pour l'ordre lexicographique.

On note L l'ensemble des mots de Lyndon sur l'alphabet Z.

Les mots de Lyndon sont implantés dans le domaine *LyndonWord2*

```
LyndonWord2 VarSet: OrderedSet  is a domain constructor
Abbreviation for LyndonWord2 is LWORD2

------------------------------ Operations ------------------------------
?<? : ($,$) -> Boolean                  ?=? : ($,$) -> Boolean
coerce : $ -> OutputForm                coerce : VarSet -> $
coerce : $ -> Expression                left : $ -> $
length : $ -> PositiveInteger           max : ($,$) -> $
min : ($,$) -> $                        retract : $ -> VarSet
retractable? : $ -> Boolean             right : $ -> Union($,"failed")
Lyndon? : OrderedFreeMonoid VarSet -> Boolean
LyndonIfCan : OrderedFreeMonoid VarSet -> Union($,"failed")
LyndonWordsBySize : (List VarSet,PositiveInteger) -> Vector List $
LyndonWordsList : (List VarSet,PositiveInteger) -> List $
aFactoredForm : $ ->
        Record(fl: VarSet,rl: List Record(gen: $,exp: NonNegativeInteger))
coerce : $ -> OrderedFreeMonoid VarSet
factor : OrderedFreeMonoid VarSet -> List $
length : ($,VarSet) -> NonNegativeInteger
retractIfCan : $ -> Union(VarSet,"failed")
```

Voici les mots de Lyndon de longueur 3 sur l'alphabet $\{a, b, c\}$

```
LyndonWordsList([a,b,c],3)$lyn
```

(7) $$[a, a^2\, b, a^2\, c, a\, b, a\, b^2, a\, b\, c, a\, c, a\, c\, b, a\, c^2, b, b^2\, c, b\, c, b\, c^2, c]$$

```
                                Type: List LyndonWord2 OrderedVarlist [c,b,a]
                                      Time: .1 (IN) + .1 (EV) = .2 sec
```

De la même manière, on définit les mots de Lyndon miroir :

Definition 2.1.2 *Un mot w de Z^* est un* **mot de Lyndon miroir** *si et seulement si il est strictement plus petit que tous ses facteurs gauches propres pour l'ordre lexicographique inverse.*

Un mot w est un mot de Lyndon miroir si et seulement si son image miroir est un mot de Lyndon. On note $\widetilde{L}$ l'ensemble des mots de Lyndon miroir sur l'alphabet Z.

Théorème 2.1.1 *Théorème de factorisation de Lyndon (voir [50]).*
Tout mot $w \in Z^*$ peut être écrit de manière unique comme suit :

$$w = l_1^{i_1} l_2^{i_2} \ldots l_n^{i_n} \qquad (n \geq 0)$$

où $l_i \in L$ (resp. $\widetilde{L}$) et $l_1 > l_2 > \cdots > l_n$ (resp. $l_1 < l_2 < \cdots < l_n$).

2.1.3.1 Factorisation standard

Soit $u \in L$.
Soit m son plus long facteur droit propre dans L.
Si $u = lm$ alors $l \in L$ et $l < lm < m$.
Le couple $\sigma(u) = (l, m)$ est appelé la *factorisation standard* de u.

$$u \in L \ ssi \ : \begin{cases} u \in Z \\ u = lm \quad avec \quad l, m \in L \ et \ l < m \end{cases}$$

2.1.3.2 Factorisation standard miroir

Tout mot de Lyndon miroir, $u \in \widetilde{L}$, peut également être factorisé de manière unique :

$$u \in \widetilde{L} \ ssi \ : \begin{cases} u \in Z \\ u = ml \ avec \quad l, m \in \widetilde{L} \ et \ l < m \end{cases}$$

On appelle le couple (m, l), noté $\tilde{\sigma}(m, l)$ la *factorisation standard miroir* de u.

2.1.4 Séries et polynômes non commutatifs

On appelle série formelle à coefficients dans l'anneau commutatif unitaire A en les indéterminées associatives z (non commutatives si $card(Z) \geq 2$) toute application S de Z^* dans A. L'image par S d'un mot w de Z^* sera notée $< S \mid w >$, et appelée coefficient de w dans S.

$$\begin{array}{rccc} S: & Z^* & \longrightarrow & A \\ & w & \longmapsto & < S \mid w > \end{array}$$

On note la série S :

$$S = \sum_{w \in Z^*} < S \mid w > w$$

Une série dont le terme constant est nul ($< S \mid \varepsilon >= 0$) est dite *propre*.

L'ensemble des séries formelles sur Z à coefficients dans A est noté $A \ll Z \gg$.

On définit le *support* d'une série S :

$$supp(S) = \{w \in Z^* \mid < S \mid w > \neq 0\}$$

On définit l'*ordre* d'une série S, qu'on note $\omega(S)$ par :

$$\omega(S) = \begin{cases} +\infty & si \quad S = 0 \\ inf\{\mid w \mid, w \in supp(S)\} & si \quad S \neq 0 \end{cases}$$

2.1.4.1 Somme de séries formelles

La *somme* de deux séries formelles S et T de $A \ll Z \gg$ est la série $S + T$ définie par :

$$\forall w \in Z^* \quad < S + T \mid w >=< S \mid w > + < T \mid w >$$

La somme est *associative* et *commutative*.

2.1.4.2 Produit de Cauchy

Le *produit de Cauchy* de deux séries formelles S et T, noté $S \cdot T$, est défini par :

$$\forall w \in Z^* \quad < T \cdot S \mid w >= \sum_{u,v \in Z^*; uv=w} < T \mid u >< S \mid v >$$

Ce produit est non commutatif.
$(A \ll Z \gg, +, \cdot)$ est un *anneau non commutatif.*

2.1.4.3 Produit par un scalaire

Soit $S \in A \ll Z \gg$ une série, soit $\alpha \in A$. On définit la série $\alpha.S$ par :

$$\forall w \in Z^* \quad < \alpha.S \mid w >= \alpha. < S \mid w >$$

L'ensemble des séries formelles muni de ces 3 opérations est une A-algèbre.

2.1.4.4 Produit de mélange

Le produit de mélange de deux séries non commutatives est défini comme suit :

$$S \sqcup T = \sum_{u,v \in Z^*} < S \mid u >< T \mid v > u \sqcup v$$

avec :

1. $u \sqcup \varepsilon = \varepsilon \sqcup u = u$
2. $\forall u, v \in Z^*, \forall x, y \in Z \quad (ux) \sqcup (vy) = [(ux) \sqcup v]y + [u \sqcup (vy)]x$

$(A \ll Z \gg, +, \sqcup, .)$ est une algèbre commutative.

2.1.4.5 Séries échangeables

Une série H est dite *échangeable* si et seulement si :

$$(\forall z \in Z, \mid u \mid_z = \mid v \mid_z) \Rightarrow < H \mid u >= < H \mid v >$$

Il s'en suit que H peut s'écrire sous la forme :

$$H = \sum_{i_0,\ldots,i_n \geq 0} c_{i_0,\ldots,i_n} z_0^{i_0} \sqcup \ldots \sqcup z_n^{i_n}$$

2.1.4.6 Polynômes non commutatifs

Un *polynôme non commutatif* est une série formelle non commutative de support fini. L'ensemble des polynômes, noté $A < Z >$, est une sous-algèbre de $A \ll Z \gg$.

On définit le degré d'un polynôme $P \in A < Z >$ par :

$$deg(P) = \begin{cases} -\infty & si \quad P = 0 \\ sup\{\mid w \mid, w \in supp(P)\} & si \quad P \neq 0 \end{cases}$$

Toute série formelle S de $A \ll Z \gg$ peut être considérée comme une forme linéaire sur $A < Z >$:

$$\forall P \in A < Z >, < S \mid P >= \sum_{w \in Z^*} < P \mid w >< S \mid w >$$

Ainsi, $A \ll Z \gg$ est identifiée au dual de $A < Z >$.

2.1.4.7 Calcul de résiduels

Definition 2.1.3 *Soit S une série formelle de $A \ll Z \gg$. Soit u un mot de Z^*. On appelle résiduel à gauche (resp. à droite) de S par u la série formelle $u \triangleleft S$ (resp. $S \triangleright u$) de $\ll Z \gg$ définie par :*

$$\forall w \in Z^*, \quad < u \triangleleft S \mid w >=< S \mid wu >$$
$$(resp. \quad < S \triangleright u \mid w >=< S \mid uw >)$$

Pour toute série $S \in A \ll Z \gg$ on a :

$$\varepsilon \triangleleft S = S \triangleright \varepsilon = S$$
$$\forall u \in Z^* \quad supp(u \triangleleft S) = \{w \in Z^* \mid wu \in supp(S)\}$$
$$supp(S \triangleright u) = \{w \in Z^* \mid uw \in supp(S)\}$$

Pour toute série formelle S, nous avons :

$$S \triangleright z = \sum_{w \in Z^*} < S \mid w > w \triangleright z$$
$$z \triangleleft S = \sum_{w \in Z^*} < S \mid w > z \triangleleft w$$

où :

$$w \triangleright z = \begin{cases} v & si \ w = zv \\ 0 & sinon \end{cases}$$

$$z \triangleleft w = \begin{cases} v & si \ w = vz \\ 0 & sinon \end{cases}$$

Toute série peut alors être retrouvée à partir de son terme constant et de la liste de ses résiduels, à gauche ou à droite, de la manière suivante :

Lemme 2.1.1 *Lemme de reconstruction*

$$S =< S \mid \varepsilon > + \sum_{z \in Z} z(S \triangleright z)$$
$$=< S \mid \varepsilon > + \sum_{z \in Z} (z \triangleleft S)z$$

Ce lemme permet de programmer les polynômes non commutatifs en utilisant une représentation "à la Hörner" (voir [53, 56]).

En mémoire, la représentation d'un polynôme non commutatif sera la liste formée de son terme constant et de ses résiduels par les éléments de Z. Chaque élément étant lui-même un polynôme non commutatif, on pourra réitérer le processus.

Exemple 2.1.1 *Soit* $Z = \{z_0, z_1\}$
Soit $P = 1 + z_0^2 + 2z_0z_1 + z_1z_0 + z_1$
$= 1 + z_0(z_0 + 2z_1) + z_1(1 + z_0)$
P est représenté par la liste:

$$(1, (z_0, P \triangleright z_0), (z_1, P \triangleright z_1))$$

avec

$$P \triangleright z_0 = ((z_0, 1), (z_1, 2))$$
$$P \triangleright z_1 = (1, (z_0, 1))$$

On peut aussi retrouver l'expression du produit de mélange de deux polynômes en utilisant le calcul des résiduels :

Lemme 2.1.2 *Lemme de dérivation*
Les résiduels, à gauche et à droite, par une lettre de Z sont des dérivations pour le produit de mélange. C'est-à-dire, pour toute lettre $z \in Z$, *pour toutes séries* $S, T \in A \ll Z \gg$, *on a :*

$$z \triangleleft (S \sqcup T) = [(z \triangleleft S) \sqcup T] + [S \sqcup (z \triangleleft T)]$$
$$(S \sqcup T) \triangleright z = [(S \triangleright z) \sqcup T] + [S \sqcup (T \triangleright z)]$$

Chaque polynôme est représenté de la manière décrite précédemment, par des listes de listes d'éléments. Chacun des nouveaux calculs de produits de mélange s'effectue sur des polynômes de taille plus petite. On est donc assuré de terminer le calcul. On retrouve ensuite l'expression de $P \sqcup Q$ par utilisation du lemme de reconstruction donné dans le paragraphe précédent.

Exemple 2.1.2 *Soit* $Z = \{z_0, z_1\}$
Soient $P = z_0z_1$ *et* $Q = z_1z_0$

$$\begin{aligned}(P \sqcup Q) \triangleright z_0 &= (P \triangleright z_0) \sqcup Q + P \sqcup (Q \triangleright z_0) \\ &= z_1 \sqcup Q + 0\end{aligned}$$

$$\begin{aligned}(P \sqcup Q) \triangleright z_1 &= (P \triangleright z_1) \sqcup Q + P \sqcup (Q \triangleright z_1) \\ &= 0 + P \sqcup z_0\end{aligned}$$

En terme de listes, $P \sqcup Q$ *sera représenté par :*

$$((z_0, z_1 \sqcup Q), (z_1, P \sqcup z_0))$$

On effectue récursivement les deux produits de mélange obtenus :

$$\begin{aligned}(P \sqcup z_0) \triangleright z_0 &= (P \triangleright z_0) \sqcup z_0 + P \sqcup (z_0 \triangleright z_0) \\ &= z_1 \sqcup z_0 + P \\ &= 2z_0z_1 + z_1z_0\end{aligned}$$

$$\begin{aligned}(P \sqcup z_0) \triangleright z_1 &= (P \triangleright z_1) \sqcup z_0 + P \sqcup (z_0 \triangleright z_1) \\ &= 0 \\ (z_1 \sqcup Q) \triangleright z_0 &= (z_1 \triangleright z_0) \sqcup Q + z_1 \sqcup (Q \triangleright z_0) \\ &= 0 \\ (z_1 \sqcup Q) \triangleright z_1 &= (z_1 \triangleright z_1) \sqcup Q + z_1 \sqcup (Q \triangleright z_1) \\ &= Q + z_1 \sqcup z_0 \\ &= 2z_1 z_0 + z_0 z_1\end{aligned}$$

Ce qui donne la liste :

$$((z_0, (z_1, Q + z_1 \sqcup z_0)), (z_1, (z_0, z_1 \sqcup z_0 + P)))$$

On obtient finalement pour le produit de mélange suivant

$$\begin{aligned}(P \sqcup Q) &= z_0[(P \sqcup Q) \triangleright z_0] + z_1[(P \sqcup Q) \triangleright z_1] \\ &= 2z_0 z_1^2 z_0 + z_1 z_0 z_1 z_0 + z_0 z_1 z_0 z_1 + 2z_1 z_0^2 z_1\end{aligned}$$

la représentation en terme de listes:

$$\Big((z_0,\ (z_1, (z_0, (z_1, 1)), (z_1, (z_0, 2)))\ )\ ,\ (z_1,\ (z_0, (z_1, (z_0, 1)), (z_0, (z_1, 2)))\ )\Big)$$

Nous avons ici un algorithme efficace *particulièrement adapté* à une représentation récursive.

2.1.5 Polynômes de Lie

On défini le *crochet de Lie* de 2 polynômes P et $Q \in A < Z >$ par :

$$[P, Q] = PQ - QP$$

On vérifie aisément que le *crochet de Lie* vérifie les trois axiomes suivants :

1. l'opération "Crochet de Lie" est bilinéaire :

$$\begin{aligned}[\lambda_1 P_1 + \lambda_2 P_2, Q] &= \lambda_1 [P_1, Q] + \lambda_2 [P_2, Q] \\ [P, \mu_1 Q_1 + \mu_2 Q_2] &= \mu_1 [P, Q_1] + \mu_2 [P, Q_2]\end{aligned}$$

2. l'opération "Crochet de Lie" est anticommutative :

$$[P, Q] = -[Q, P]$$

3. l'opération "Crochet de Lie" vérifie l'identité de Jacobi :

$$[P, [Q, R]] + [Q, [R, P]] + [R, [P, Q]] = 0$$

On note $Lie < Z >$ le plus petit sous A-module qui contient les lettres de Z et qui est stable pour le crochet de Lie.
$Lie < Z >$ est l'algèbre de Lie libre engendrée par Z.
Un élément de $Lie < Z >$ est appelé un *polynôme de Lie.*

Par conséquent, un polynôme de Lie est soit une lettre de Z (munie d'un coefficient non nul), soit le crochet de Lie de deux polynômes de Lie.
Tout polynôme de Lie est propre.

2.1.6 Base de Poincaré-Birkhoff-Witt

Théorème 2.1.2 *Théorème de Poincaré-Birkhoff-Witt ([28]).*
Soit $B = (P_j)_{j\geq 1}$ une base totalement ordonnée d'une l'algèbre de Lie L. Alors la famille des polynômes de la forme

$$\{P_{j_1}^{i_1} P_{j_2}^{i_2} \dots P_{j_n}^{i_n} \text{ avec } n \geq 0,\ i_1, i_2, \dots, i_n \geq 0\}$$

où

$$j_1 > j_2 > \dots > j_n$$

est une base de l'algèbre enveloppante de L appelée base de Poincaré-Birkhoff-Witt associée à la base B et notée $PBW.B$.

2.1.7 Base duale de $PBW.B$

L'espace vectoriel des séries formelles est naturellement isomorphe au dual de l'espace vectoriel des polynômes non commutatifs. On peut exprimer cette dualité sous la forme suivante :

$$\begin{array}{rcl} A \ll Z \gg \times A < Z > & \longrightarrow & A \\ (S, P) & \longmapsto & < S \mid P >= \sum_{w \in Z^*} < S \mid w >< P \mid w > \end{array}$$

On associe à la base $PBW.B$ de $A < Z >$ sa "base duale" : $\{S_Q\}_{Q \in PBW.B}$.
Chaque série S_{Q_i} est définie de la manière suivante :

$$< S_{Q_i} \mid Q_j >= \delta_{Q_i}^{Q_j} \qquad \forall Q_j \in PBW.B$$

On a alors :

$$\forall P \in A < Z > \qquad P = \sum_{Q \in PBW.B} < S_Q \mid P > Q \tag{2.1}$$

Théorème 2.1.3 *[57] Soit $Q = P_{j_1}^{i_1} P_{j_2}^{i_2} \cdots P_{j_n}^{i_n}$ un polynôme de la base $PBW.B$, alors :*

$$S_Q = \frac{1}{i_1! i_2! \cdots i_n!} S_{P_{j_1}}^{\text{ш} i_1} \text{ш} S_{P_{j_2}}^{\text{ш} i_2} \text{ш} \cdots \text{ш} S_{P_{j_n}}^{\text{ш} i_n}$$

2.1.8 Base de Lyndon

A partir des mots de Lyndon, on construit une base de l'algèbre des polynômes de Lie, *Lie* $< Z >$, appelée *base de Lyndon* ou *base de Chen-Fox-Lyndon* par la formule récurente suivante :

$$\begin{cases} c(z) = z & si\ z \in Z, \\ c(w) = [c(l), c(m)] & si\ w \in L \text{ et } \sigma(w) = (l, m). \end{cases}$$

où $[c(l), c(m)]$ est le crochet de Lie de $c(l)$ et de $c(m)$ et $\sigma(w)$ est la factorisation standard de w.

Lemme 2.1.3 *A tout mot de Lyndon u, on associe le polynôme de Lie Q_u de la manière suivante :*

$$\begin{array}{ll} Q_u = u & si\ u \in Z \\ Q_u = [Q_l, Q_m] & si\ \sigma(u) = (l, m) \end{array}$$

$\{Q_u \mid u \in L\}$ est une base de l'espace vectoriel $Lie < Z >$. Nous l'appelons *base de Lyndon.*

De la même manière, on défini la base de Lyndon miroir :

$$\begin{cases} c(z) = z & si\ z \in Z, \\ c(w) = [c(l), c(m)] & si\ w \in \widetilde{L}\ et\ \tilde{\sigma}(w) = (l, m). \end{cases}$$

Lemme 2.1.4 *A tout mot de Lyndon miroir u, on associe le polynôme de Lie Q_u de la manière suivante :*

$$\begin{aligned} Q_u &= u && si\ u \in Z \\ Q_u &= [Q_m, Q_l] && si\ \tilde{\sigma}(u) = (m, l) \end{aligned}$$

$\{Q_u \mid u \in \widetilde{L}\}$ est alors appelé *base de Lyndon miroir.*

Exemple 2.1.3 *Soit $Z = \{z_0, z_1\}$: Enumération par longueur.*

Longueur	*Mots de Lyndon*	*Base de Lyndon*
1	z_0	z_0
	z_1	z_1
2	$z_0 z_1$	$[z_0, z_1]$
3	$z_0^2 z_1$	$[z_0, [z_0, z_1]]$
	$z_0 z_1^2$	$[[z_0, z_1], z_1]$
4	$z_0^3 z_1$	$[z_0, [z_0, [z_0, z_1]]$
	$z_0^2 z_1^2$	$[z_0, [[z_0, z_1], z_1]$
	$z_0 z_1^3$	$[[[z_0, z_1], z_1], z_1]$
5	$z_0^4 z_1$	$[z_0, [z_0, [z_0, [z_0, z_1]]]$
	$z_0^3 z_1^2$	$[z_0, [z_0, [[z_0, z_1], z_1]]$
	$z_0^2 z_1 z_0 z_1$	$[[z_0, [z_0, z_1]], [z_0, z_1]]$
	$z_0^2 z_1^3$	$[z_0, [[[z_0, z_1], z_1], z_1]]$
	$z_0 z_1 z_0 z_1^2$	$[[z_0, z_1], [[z_0, z_1], z_1]]$
	$z_0 z_1^4$	$[[[[z_0, z_1], z_1], z_1], z_1]$

Longueur	*Mots de Lyndon miroir*	*Base de Lyndon miroir*
1	z_1	z_1
	z_0	z_0
2	$z_1 z_0$	$[z_1, z_0]$
3	$z_1^2 z_0$	$[z_1, [z_1, z_0]]$
	$z_1 z_0^2$	$[[z_1, z_0], z_0]$
4	$z_1^3 z_0$	$[z_1, [z_1, [z_1, z_0]]$
	$z_1^2 z_0^2$	$[z_1, [[z_1, z_0], z_0]$
	$z_1 z_0^3$	$[[[z_1, z_0], z_0], z_0]$
5	$z_1^4 z_0$	$[z_1, [z_1, [z_1, [z_1, z_0]]]$
	$z_1^3 z_0^2$	$[z_1, [z_1, [[z_1, z_0], z_0]]$
	$z_1^2 z_0 z_1 z_0$	$[[z_1, [z_1, z_0]], [z_1, z_0]]$
	$z_1^2 z_0^3$	$[z_1, [[[z_1, z_0], z_0], z_0]]$
	$z_1 z_0 z_1 z_0^2$	$[[z_1, z_0], [[z_1, z_0], z_0]]$
	$z_1 z_0^4$	$[[[[z_1, z_0], z_0], z_0], z_0]$

2.1.8.1 Construction de la base duale

La famille $\{P_w \mid w \in Z^*\}$ est *exactement* la base $PBW.L$ de $K < Z >$ associée à la base de Lyndon de $Lie < Z >$ par le théorème de PBW (2.1.2). On construit la base duale de la base de $PBW.L$ $\{S_w \mid w \in Z^*\}$, à l'aide du théorème suivant :

Théorème 2.1.4 Calcul récursif des polynômes duaux([46]).
Le polynôme dual S_w peut être obtenu récursivement des deux manières suivantes :

$$S_w = \begin{cases} \varepsilon & si \quad w = \varepsilon \\ zS_v & si \quad w = zv \in L \\ \frac{1}{i_1!i_2!\ldots i_n!} S_{l_1}^{\sqcup i_1} \sqcup S_{l_2}^{\sqcup i_2} \sqcup \ldots \sqcup S_{l_n}^{\sqcup i_n} & si \quad w = l_1^{i_1} l_2^{i_2} \ldots l_n^{i_n} \text{ avec } \begin{matrix} l_k \in L \\ l_1 > l_2 > \ldots > l_n \end{matrix} \end{cases}$$

et

$$\widetilde{S}_w = \begin{cases} \varepsilon & si \quad w = \varepsilon \\ \widetilde{S}_v z & si \quad w = vz \in \widetilde{L} \\ \frac{1}{i_1!i_2!\ldots i_n!} \widetilde{S}_{\widetilde{l}_1}^{\sqcup i_1} \sqcup \widetilde{S}_{\widetilde{l}_2}^{\sqcup i_2} \sqcup \ldots \sqcup \widetilde{S}_{\widetilde{l}_n}^{\sqcup i_n} & si \quad w = \widetilde{l}_1^{i_1} \widetilde{l}_2^{i_2} \ldots \widetilde{l}_n^{i_n} \text{ avec } \begin{matrix} \widetilde{l}_k \in \widetilde{L} \\ \widetilde{l}_1 < \widetilde{l}_2 < \ldots < \widetilde{l}_n \end{matrix} \end{cases}$$

ceci donne une définition récursive des S_w. On notera aussi ([57]) que l'ensemble des S_{l_i} pour $l_i \in L$ (resp. $\widetilde{S}_{\widetilde{l}_i}$ pour $\widetilde{l}_i \in \widetilde{L}$) est une base de transcendance de l'algèbre commutative $A \ll Z \gg$ dotée du produit de mélange.

2.1.9 Factorisation de la série double

On définit un produit sur $K \ll Z \gg \otimes K < Z >$ par :

$$(S_1 \otimes P_1)(S_2 \otimes P_2) = S_1 \sqcup S_1 \otimes P_1 P_2$$

En particulier, d'après l'équation 2.1 de la page 47, si $w \in Z^*$, on obtient :

$$\begin{aligned} \sum_{w \in Z^*} w \otimes w &= \sum_{w \in Z^*} w \otimes \Big(\sum_{Q \in PBW.B} < S_Q \mid w > Q \Big) \\ &= \sum_{Q \in PBW.B} \Big(\sum_{w \in Z^*} < S_Q \mid w > w \Big) \otimes Q \\ &= \sum_{Q \in PBW.B} S_Q \otimes Q \end{aligned}$$

On en déduit alors le *théorème de factorisation* :

Théorème 2.1.5 *([46])*

$$\sum_{w \in Z^*} w \otimes w = \prod_{P \in B} exp(S_P \otimes P)$$

Les éléments de B sont ordonnés décroissants pour $B = L$ et croissants pour $B = \widetilde{L}$.

2.2 Systèmes dynamiques et transformation d'évaluation

2.2.1 Les systèmes dynamiques

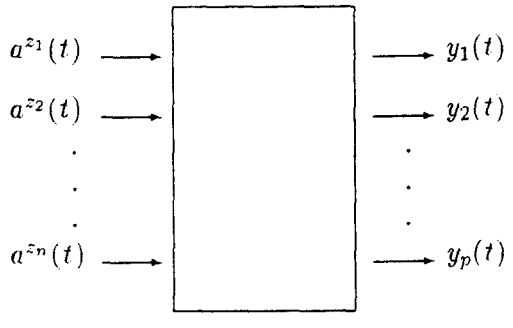
Comme nous l'avons dit dans l'introduction générale, un système dynamique est un système d'équations différentielles de la forme :

$$(\Sigma)\begin{cases} \dot{q}(t) &= \sum_{z\in Z} a^z(t) A_z(q) \\ y(t) &= h(q(t)) \end{cases}$$

avec A_z un champ de vecteurs analytique défini par :

$$A_z = \sum_{k=1}^{n} A_z^k(q)\frac{\partial}{\partial q_k},$$

Mais on peut aussi considérer les systèmes dynamiques comme des "boîte noires"



dont chaque sortie $y_k(t)$ est un signal paramétré dépendant du temps t et des primitives des entrées a^z pour $z \in Z$, c'est à dire :

$$y_k(t) = g_k(t, \int_0^t a^{z_1}(\tau)d\tau, \ldots, \int_0^t a^{z_n}(\tau)d\tau)$$

a^{z_0} est l'application constante : $a^{z_0} = 1$. Pour $k \in \{1, \ldots, p\}$, Les g_k sont des fonctionnelles, c'est à dire des fonctions dont les arguments sont des fonctions dépendant des $\xi_z(t)$:

$$y_k(t) = g_k\,(t, \xi_{z_1}(t), \xi_{z_2}(t), \cdots, \xi_{z_n}(t))$$

avec

$$\xi_z(t) = \int_0^t a^z(\tau)d\tau, \qquad z \in Z$$

Pour coder de telles fonctionnelles, M. Fliess a introduit ([22]) les séries formelles en les indéterminées non commutatives $z \in Z$ appelées séries génératrices (ou séries de Fliess) :

$$S = \sum_{w\in Z^*} < S \mid w > w$$

où chaque lettre de Z représente une opération d'intégration de Stieljes relativement à l'entrée correspondante. En utilisant ce codage, nous sommes ammenés à "oublier" ces opérations pour ne plus manipuler que des expressions construites sur les lettres de l'alphabet Z.

Dans l'exemple suivant, tiré du problème du "motion planning" et repris dans la dernière partie de ce chapitre,

$$\begin{cases} \dot{q_1} &= cos(q_3)u_1 \\ \dot{q_2} &= sin(q_3)u_1 \\ \dot{q_3} &= u_2 \end{cases}$$

Les champs de vecteurs correspondants sont :

$$f_1 = \begin{pmatrix} cos(q_3) \\ sin(q_3) \\ 0 \end{pmatrix} \qquad f_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

les séries génératrices S_1, S_2, S_3 des états q_1, q_2, q_3 seraient :

$$\begin{cases} S_1 &= \sum_{n\geq 0}(-1)^n z_2^{2n} z_1 \\ S_2 &= \sum_{n\geq 0}(-1)^n z_2^{2n+1} z_1 \\ S_3 &= z_2 \end{cases}$$

où z_1 et z_2 codent les entrées u_1 et u_2 qui sont ici soit des fonctions constantes par morceaux, soit des polynômes.

La transformation d'une expression donnée par ses champs de vecteurs en équations d'état est implantée dans le domaine des *Polysystèmes*.

Certains systèmes dynamiques peuvent également coder une équation différentielle. Prenons par exemple l'équation de Düffing (que nous traitons plus complètement en 2.2.4.2) :

$$\ddot{y} + \dot{y} + y + y^3 = a^{z_1}$$

En posant $q_1(t) = y(t)$ et $q_2(t) = \dot{y}(t)$, on trouve la représentation d'état suivante :

$$\begin{cases} \dot{q_1}(t) &= q_2(t) \\ \dot{q_2}(t) &= a^{z_1}(t) - q_1^3(t) - q_1(t) - q_2(t) \\ y(t) &= h(q(t)) = q_1(t) \end{cases}$$

Nous avons donc les deux champs de vecteurs :

$$\begin{cases} A_{z_0} = q_2(t)\dfrac{\partial}{\partial q_1} - (q_1^3(t) + q_1(t) + q_2(t))\dfrac{\partial}{\partial q_2} \\ A_{z_1} = \dfrac{\partial}{\partial q_2} \end{cases}$$

La série génératrice codant la solution de l'équation de Düffing vérifie l'équation symbolique suivante (voir [38, 28]) :

$$S(z_0^2 + z_0 + 1) = z_1 + S^{\omega^3} z_0$$

La détermination des solutions d'une telle équation nécessite des méthodes d'approximation développées dans [38, 28] par exemple.

Nous rappelons ici deux techniques couramment utilisées pour calculer les approximations des séries génératrices des systèmes dynamiques et en particulier ceux provenant des équations différentielles en régime forcé (nous utilisons ici les même notations que celles de [28]) :
M.Fliess et H.J.Sussman ont montré qu'on peut approximer uniformément le comportement entrée/sortie de tout système dynamique par un système polynomial (c'est à dire de série génératrice finie), et plus finement, par un système bilinéaire (c'est à dire de série génératrice rationnelle) (voir [22, 23, 24, 64, 27]).

Les séries génératrices ont permis, notamment, de généraliser le calcul symbolique de Heaviside aux systèmes non linéaires, grâce aux indéterminées non commutatives.

Cette vue a été initiée par M. Fliess, M. Lamnabhi et F. Lamnabhi-Lagarrigue ([25]). Des développements peuvent être trouvés dans [40], permettant dans certains cas de calculer la réponse, exacte ou approchée, d'un système dynamique non linéaire, grâce au calcul formel (voir aussi [2, 39, 41, 42, 43, 45])

C'est une remise en forme de ces techniques, mieux adaptées à un calcul formel rapide et efficace, qui a conduit à introduire la "transformation d'évaluation", qui sera utilisée tout au long de ce travail.

2.2.1.1 Approximation polynomiale

On peut approcher le comportement d'un système dynamique donné par sa série génératrice par des polynômes non commutatifs $\mathcal{G}_k$ obtenus en tronquant $\mathcal{G}$ à un ordre donné :

$$\mathcal{G}_k = \sum_{|w| \leq k} < \mathcal{G} \mid w > w$$

notation
Soit S une série propre, on note S^{*k} l'expression $(S^*)^k$
Soit S une série , on note $S^{\sqcup k}$ l'expression $S \sqcup S \sqcup \ldots \sqcup S$ (k fois).

illustrons cette propriété sur un exemple.

Exemple 2.2.1
Considérons le système décrit par les équations suivantes :

$$\begin{cases} \dot{y} &= u + y^2 \\ y(0) &= 0 \end{cases}$$

La sortie y du système peut être exprimée sous la forme suivante :

$$y = \int_0^t u(\tau)d\tau + \int_0^t y^2(\tau)d\tau$$

Selon la technique de M. Fliess, nous codons cette équation de la manière suivante :

$$\int_0^t u(\tau)d\tau \mapsto z_1$$

$$\int_0^t d(\tau) \mapsto z_0$$

$$y^n \mapsto S^{\sqcup n}$$

La série génératrice de y vérifie alors l'équation symbolique suivante :

$$S = z_1 + S^{\sqcup 2} z_0$$

Nous utilisons les schémas itératifs convergents ([28]) :

$$\begin{cases} S_1 &= z_1 \\ S_k &= S_1 + S_{k-1}^{\sqcup 2} z_0 \end{cases}$$

Nous calculons les premières valeurs de S :

$$\begin{aligned} S_1 &= z_1 \\ S_2 &= z_1 + 2z_1^2 z_0 \\ S_3 &= z_1 + 2z_1^2 z_0 + 8(z_1^2 z_0)^2 z_0 + 48 z_1^4 z_0^3 + 24 z_1^2 (z_1 z_0)^2 z_0 + 12 z_1^3 z_0^2 + 4 z_1^2 z_0 z_1 z_0 \end{aligned}$$

2.2.1.2 Approximation rationnelle

On peut aussi approximer plus finement par des *fractions rationnelles* dont le début coïncide exactement avec celui de la série génératrice :

Exemple 2.2.2 *Considérons l'équation différentielle du premier ordre suivante :*

$$\dot{y} = u + y + y^2$$

qui s'écrit encore

$$\dot{y} - y = u + y^2$$

En utilisant la technique de codage décrite dans l'exemple 2.2.1, on écrit cette équation en terme de séries :

$$\begin{aligned} S - Sz_0 &= z_1 + (S \sqcup S) z_0 \\ S(1 - z_0) &= z_1 + (S \sqcup S) z_0 \end{aligned}$$

On obtient finalement l'expression de la série S :

$$S = z_1 z_0^* + (S \sqcup S) z_0 z_0^*$$

2.2.1.2.1 Calcul des approximations rationnelles

1. $S_1 = z_1 z_0^*$
2. $\begin{aligned} S_2 &= S_1 + (S_1 \sqcup S_1) z_0 z_0^* \\ &= z_1 z_0^* + (z_1 z_0^* \sqcup z_1 z_0^*) z_0 z_0^* \\ &= z_1 z_0^* + 2 z_1 z_0^* z_1 (2 z_0)^* z_0 z_0^* \end{aligned}$
3. $\begin{aligned} S_3 &= S_1 + (S_2 \sqcup S_2) z_0 z_0^* \\ &= S_1 + [(S_1 + T) \sqcup (S_1 + T)] z_0 z_0^* \\ &= S_1 + [S_1 \sqcup S_1 + 2 S_1 \sqcup T + T \sqcup T] z_0 z_0^* \\ &= S_2 + [2 S_1 \sqcup T + T \sqcup T] z_0 z_0^* \end{aligned}$

   avec $T = 2 z_1 z_0^* z_1 (2 z_0)^* z_0 z_0^*$

$$S_1 \text{ш} T = 2z_1 z_0^* z_1 (2z_0)^* [z_0 z_0^* z_1 + 3z_1 (3z_0)^* z_0](2z_0)^*$$
$$T \text{ш} T = 2[24 z_1 z_0^* z_1 (2z_0)^* z_1 (3z_0)^* z_1 (4z_0)^* z_0 + 2(S_1 \text{ш} T) z_1](3z_0)^* z_0 (2z_0)^*$$

$$\begin{aligned} S_3 = {} & z_1 z_0^* + 2(2z_0)^* z_1 z_0^* z_1 z_0 z_0^* + \\ & [(4 z_1 z_0^* z_1 (2z_0)^* [z_0 z_0^* z_1 + 3 z_1 (3z_0)^* z_0](2z_0)^* + \\ & 2[24 z_1 z_0^* z_1 (2z_0)^* z_1 (3z_0)^* z_1 (4z_0)^* z_0 + \\ & 4 z_1 z_0^* z_1 (2z_0)^* [z_0 z_0^* z_1 + 3 z_1 (3z_0)^* z_0](2z_0)^* z_1](3z_0)^* z_0 (2z_0)^*] z_0 z_0^* \end{aligned}$$

On sait (voir [28]) que les séries S_2 et S_3 coïncident exactement avec la série S respectivement jusqu'aux ordres 4 et 6, ainsi l'approximant de S_2 à l'ordre 4 est :

$$z_1 + z_1 z_0 + 2z_1^2 + 2z_1 z_0 z_1 + 4z_1^2 z_0 + 2z_1 z_0^2 z_1 + 8z_1^2 z_0^2 + 4z_1 z_0 z_1 z_0$$

Pour passer du codage symbolique au domaine temporel, on utilise la *transformation d'évaluation*, étudiée récemment dans [28] dont nous décrivons ici les grandes lignes.

2.2.2 Transformation d'évaluation avec noyau

2.2.2.1 Evaluation des séries

L'évaluation de la série formelle S par rapport au noyau f pour l'entrée $a = \{a^{z_0}, a^{z_1}, \ldots a^{z_n}\}$ (sous réserve de convergence, voir [29]) est la fonction :

$$\mathcal{E}_a(f; S) = \sum_{w \in Z^*} < S \mid w > \mathcal{E}_a(f; w)$$

avec l'évaluation de w par rapport au noyau f pour l'entrée a, l'intégrale définie récursivement comme suit :

$$\mathcal{E}_a(f; w)(t) = \begin{cases} f(t) & si \quad w = \varepsilon \\ \int_0^t \mathcal{E}_a(f; v)(\tau) d\xi_z(\tau) & si \quad w = vz \end{cases}$$

En particulier, lorsque $f = 1$, on note $\mathcal{E}_a(1; S) = \mathcal{E}_a(S)$.
Et $\mathcal{E}_a(w)(t)$ est l'intégrale itérée $\int_0^t \delta_a w$.

2.2.2.2 Propriétés fondamentales

1. Soient S et T deux séries et α un scalaire :

$$\mathcal{E}_a(f; S + \alpha T) = \mathcal{E}_a(f; S) + \alpha \mathcal{E}_a(f; T)$$

2. Soient S et T deux séries alors :

$$\mathcal{E}_a(f; ST) = \mathcal{E}_a(\mathcal{E}_a(f; S); T)$$

3. Soient S et T deux séries alors :

$$\mathcal{E}_a(S \text{ш} T) = \mathcal{E}_a(S) \mathcal{E}_a(T)$$

4. Soit une série échangeable $H = \sum_{i_0,\ldots,i_n \geq 0} c_{i_0,\ldots,i_n} z_0^{i_0} \sqcup \ldots \sqcup z_n^{i_n}$ alors :

$$\mathcal{E}_a(H) = \sum_{i_0,\ldots,i_n \geq 0} c_{i_0,\ldots,i_n} \frac{\xi_{z_0}^{i_0} \ldots \xi_{z_n}^{i_n}}{i_0! \ldots i_n!}$$

5. Soit H une série échangeable et $h(\xi(t))$ son évaluation alors :

$$\mathcal{E}_a(f; H) = \int_0^t h(\xi(t) - \xi(\tau)) df(\tau)$$

6. soit $g_k(\xi_z(t))$ l'évaluation de la série échangeable $\left(1 + \sum_{z \in Z} c_z z\right)^{k-1}$, c'est-à-dire :

$$g_k(\xi_z(t)) = \sum_{j=0}^{k-1} \binom{k-1}{j} \sum_{\substack{i_0,\ldots,i_n \geq 0 \\ i_0 + \ldots + i_n = i_0}} \frac{[c_{z_0} t]^{i_0} \ldots [c_{z_n} \xi_{z_n}(t)]^{i_n}}{i_0! \ldots i_n!}$$

alors pour tout entier $k \geq 1$, pour tout nombre complexe $c_z, z \in Z$, on a :

$$\mathcal{E}_a\left[f; \left(\sum_{z \in Z} c_z z\right)^{*k}\right] = \int_0^t g_k(\xi(t) - \xi(\tau)) exp\left[\sum_{z \in Z} c_z [\xi_z(t) - \xi_z(\tau)]\right] df(\tau)$$

Ces propriétés de la transformation d'évaluation permettent, en particulier, de calculer la sortie pour des séries formelles de la forme

$$G_0 z_{i_1} G_1 \ldots z_{i_k} G_k$$

où

$$G_0, \ldots, G_k$$

sont échangeables, et les $z_{i_1}, \ldots, z_{i_k}$ sont des lettres de Z.

2.2.2.3 Evaluation des polynômes et implantation

En particulier, lorsque la série considérée est un polynôme, le résultat d'évaluation existe toujours :
L'évaluation du polynôme $P \in K < Z >$ pour l'entrée $a = (a^{z_0}, a^{z_1}, \ldots, a^{z_n})$ est la fonction définie comme suit :

$$\mathcal{E}_a(f; P) = \sum_{w \in supp(P)} < P \mid w > \mathcal{E}_a(f; w)$$

L'implantation de la transformation d'évaluation est par conséquent directement issue de celle des polynômes non commutatifs. En appliquant le lemme de reconstruction 2.1.1 et les propriétés fondamentales des séries (voir 2.2.2.2), nous avons deux méthodes de calcul de l'évaluation :

1. En utilisant le résiduel à gauche :

$$\begin{aligned}\mathcal{E}_a(f;P) &= \mathcal{E}_a\left(f;< P \mid \varepsilon > + \sum_{z\in Z}(z \triangleleft P)z\right)\\ &= < P \mid \varepsilon > f + \sum_{z\in Z}\int_0^t \mathcal{E}_a(f; z \triangleleft P)d\xi_z\end{aligned}$$

   Le résultat est ici porté par les intégrales.

2. ou en utilisant le résiduel à droite :

$$\begin{aligned}\mathcal{E}_a(f;P) &= \mathcal{E}_a\left(f;< P \mid \varepsilon > + \sum_{z\in Z} z(P \triangleright z)\right)\\ &= < P \mid \varepsilon > f + \sum_{z\in Z}\mathcal{E}_a\left(\mathcal{F};\sum_{z\in Z} P \triangleright z\right)\end{aligned}$$

   avec

$$\mathcal{F} = \mathcal{E}_a(f;z) = \int_0^t f d\xi_z$$

   Dans ce cas, le résultat est porté par le noyau qui est enrichi au fur et à mesure que le polynôme P se réduit. Etant donnée la structure d'implantation choisie pour les polynômes, (voir sur l'exemple 2.1.1), c'est la deuxième méthode qui est utilisée pour l'implantation. Celle-ci, effectuée récursivement permet *par un simple balayage de liste* d'accéder aux résiduels de P, ce qui donne un algorithme **particulièrement efficace** d'évaluation polynomiale.

La transformation d'évaluation est implantée dans le paquetage *XEvaluationPackage1* dont nous donnons ici le code source.

```
)abbrev package XEVAL1 XEvaluationPackage1

XEvaluationPackage1(VarSet,K,CODAGE,ENTREE,integ): Cat == Def where
    VarSet:OrderedSet
    K:Field
    CODAGE: XFreeAlgebra(VarSet, K)
    ENTREE: Algebra(K)
    integ : ENTREE -> ENTREE

    Cat == with
       xeval: (CODAGE, List VarSet, List ENTREE) -> ENTREE
       xeval: (ENTREE,CODAGE, List VarSet, List ENTREE) -> ENTREE

    Def == add
       xeval(noyau, code, lv, le) ==
          r1,r2: ENTREE
          # lv ^= # le => error "longueurs non concordantes"
          constant? code => constant(code) * noyau
          r1 := +/ [xeval(integ(le.i * noyau),resd,lv,le)
                    for i in 0..maxIndex lv | (resd:=lquo(code,lv.i)) ^= 0]
          r2 := constant(code) * noyau
```

```
        r1 + r2

    xeval(code,lv,le) == xeval(1::ENTREE,code,lv,le)
```

On voit ici la mise en pratique du principe de généricité de Scratchpad : nous allons utiliser la même fonction d'évaluation pour *des types d'entrées aussi variés que possible*. Des entrées polynomiales en base canonique pour une expression mathématique habituelle des sorties, des entrées polynomiales en base de Bernstein (voir chapitre 3) pour des sorties du même type permettant un tracé instatané des courbes d'entrée et de sortie, et enfin des entrées polynomiales exponentielles (voir 2.2.3.1) que nous traçons également par une méthode originale (voir chapitre 3).

2.2.2.4 Exemple

Evaluation des séries polynomiales définies dans l'exemple 2.2.1 pour $u(t) = t$. Les opérations (1) à (11), que nous ne donnons pas ici, sont uniquement des instanciations de l'alphabet de codage, qui comprend ici les lettres a et b, du type des polynômes codant la série, ici les polynômes non commutatifs en représentation récursive et du paquetage d'évaluation, auquel il faut donner la fonction d'intégration à utiliser, ici celle implantée dans le domaine des polynômes exponentiels.

```
------------------------------------------------------------------------
--                calcul de la serie polynomiale
------------------------------------------------------------------------
(11) ->p2: Codage := b+2*b*b*a
```

$$(11) \qquad b\,(1 + b\,a\,2)$$

```
       Type: XRecursivePolynomial(OrderedVarlist [a,b],RationalNumber)
                      Time: .767 (IN) + .133 (EV) + .1 (OT) = 1.0 sec

(12) ->p3: Codage := b+sh(p2,p2)*a
```

$$(12) \qquad b\,(1 + b\,(b\,(b\,a\,a\,a\,48 + a\,(b\,a\,a\,24 + a\,12)) + a\,(2 + b\,(b\,a\,a\,8 + a\,4))))$$

```
       Type: XRecursivePolynomial(OrderedVarlist [a,b],RationalNumber)
                                  Time: .433 (IN) + .167 (EV) = .6 sec

------------------------------------------------------------------------
--                evaluation pour l'entree (1,t)
------------------------------------------------------------------------
(13) ->xeval(p3,[a,b],[ua,ub])$Evpkg
```

$$(13) \qquad \left(\frac{1}{4400}\right) t^{11} + \left(\frac{1}{160}\right) t^{8} + \left(\frac{1}{20}\right) t^{5} + \left(\frac{1}{2}\right) t^{2}$$

```
                                        Type: Polynomial[t]RationalNumber
                                      Time: .233 (IN) + .1 (OT) = .333 sec
------------------------------------------------------------------------
--          evaluation pour l'entree (1,t**4-3*t**2-1/2)
------------------------------------------------------------------------
(14) ->ub:=t**4-3*t**2-1/2
```

$$(14) \qquad t^4 - 3\,t^2 - \frac{1}{2}$$

```
                                        Type: Polynomial[t]RationalNumber
                                    Time: 1.067 (IN) + .1 (OT) = 1.167 sec
```

```
(15) ->xeval(p3,[a,b],[ua,ub])$Evpkg
```

$$(15) \qquad \begin{array}{l} \left(\frac{1}{1739375}\right) t^{23} - \left(\frac{4}{259875}\right) t^{21} + \left(\frac{2188}{14812875}\right) t^{19} - \left(\frac{628}{1472625}\right) t^{17} - \left(\frac{643}{330750}\right) t^{15} \\ + \\ \left(\frac{8798}{675675}\right) t^{13} - \left(\frac{8}{17325}\right) t^{11} - \left(\frac{89}{945}\right) t^{9} + \left(\frac{317}{5040}\right) t^{7} + \left(\frac{23}{60}\right) t^{5} - \left(\frac{11}{12}\right) t^{3} - \left(\frac{1}{2}\right) t \end{array}$$

```
                                        Type: Polynomial[t]RationalNumber
                                    Time: .2 (IN) + .5 (EV) + .1 (OT) = .8 sec
```

2.2.2.5 Transformation d'évaluation sur la base duale

Le théorème 2.1.4 fournit une construction de la base de transcendance $\{S_l\}_{l\in\widetilde{L}}$ de l'algèbre de mélange sur $A \ll Z \gg$. Il fournit aussi un algorithme récursif de calcul de la transformation d'évaluation des polynômes $\widetilde{S}_{\widetilde{l}}$: Nous utilisons cette formule pour une implantation récursive en Scratchpad :

$$\mathcal{E}_a(\widetilde{S}_{\widetilde{l}}) \begin{cases} = & 1 & si \quad \widetilde{l} = \varepsilon \\ & \xi_z & si \quad \widetilde{l} = z \\ & \frac{1}{i_1!i_2!\cdots i_n!}\int_0^t \left(\xi_{\widetilde{l}_1}^{i_1}\xi_{\widetilde{l}_2}^{i_2}\cdots\xi_{\widetilde{l}_n}^{i_n}\right) d\xi_z & si \quad \widetilde{l} = \widetilde{l}_1^{i_1}\widetilde{l}_2^{i_2}\cdots\widetilde{l}_n^{i_n} z \end{cases}$$

L'évaluation des S_l est implantée dans le paquetage *EvalOfDualPBWLElt2*

```
EVDPBWL2(VarSet: ORDSET,K: ALGEBRA RationalNumber,INPUT: ALGEBRA K,
                                                integrate: (INPUT -> INPUT))
 is a domain constructor
Abbreviation for EvalOfDualPBWLElt2 is EVDPBWL2

------------------------------ Operations ------------------------------
EvaluationTransform : (INPUT,LWORD2 VarSet,
                                L Record(cod: VarSet,fct: INPUT)) -> INPUT
EvaluationTransform : (LWORD2 VarSet,L Record(cod: VarSet,fct: INPUT))
                                                              -> INPUT
makeInput : (VarSet,INPUT) -> Record(cod: VarSet,fct: INPUT)
```

L'évaluation des S_l, $l \in lynlist = [a, ab, b]$ pour l'entrée constante par morceaux (voir 2.3.2)

$$u_1(t) = \begin{cases} a_1 & pour \quad 0 \leq t < \frac{T}{2} \\ a_2 & pour \quad \frac{T}{2} \leq t < T \end{cases}$$

$$u_2(t) = \begin{cases} b_1 & pour \quad 0 \leq t < \frac{T}{2} \\ b_2 & pour \quad \frac{T}{2} \leq t < T \end{cases}$$

Nous posons $A_1 = a_1\frac{T}{2}$, $A_2 = a_2\frac{T}{2}$ et $B_1 = b_1\frac{T}{2}$, $B_2 = b_2\frac{T}{2}$

```
[ (EvaluationTransform( l, [ u1, u2 ] )$evdom ::vdp).1 for l in lynlist ]
```

$$(21) \qquad [A1 + A2, \frac{1}{2} A1\ B1 + A2\ B1 + \frac{1}{2} A2\ B2, B1 + B2]$$

```
Type: L DistributedMultivariatePolynomial([A1,A2,B1,B2],RationalNumber)
                                Time: .667 (IN) + .233 (EV) + .1 (OT) = 1.0 sec
```

2.2.3 Le type des entrées

Les deux types d'approximations de séries, présentés en 2.2.1, ne nécessitent pas les mêmes types d'entrées.

En effet, les approximations polynomiales conviennent parfaitement à des classes d'entrées polynomiales : l'intégrale d'un polynôme est un polynôme, le calcul des intégrales itérées, c'est à dire de l'évaluation, donnera donc un polynôme. La sortie de l'approximation polynomiale d'une série pour des entrées polynomiales sera alors également polynomiale. Ce résultat permet de représenter les entrées par des polynômes exprimés en base de Bernstein (voir chapitre 3), ce qui permet de simuler entrées et sorties par des courbes de Bézier dont nous décrivons le principe et les avantages dans le chapitre 3.

Les fractions rationnelles, par contre, fournissent des résultats plus complexes : L'utilisation d'entrées polynomiales ne garantit pas une sortie du même type. On trouve en effet, des lettres à la puissance étoile dans l'expression des séries, et leur évaluation donnera une exponentielle.

2.2.3.1 Les entrées polynomiales exponentielles

Commençons par un exemple ([28]).

Exemple 2.2.3 *Soit l'équation différentielle :*

$$\dot{y} + a(t)y = b(t)$$

Pour simplifier les calculs, nous supposons que $y(0) = 0$
En intégrant cette équation, on obtient :

$$y(t) + \int_O^t y(\tau) d\xi_a = \xi_b$$

En codant $\int d\xi_a$ *par* z_a *et* $\int d\xi_b$ *par* z_b *, on obtient en terme de séries génératrices :*

$$\begin{array}{rlcl} & S + Sz_a & = & z_b \\ \Longleftrightarrow & S(1+z_a) & = & z_b \\ \Longleftrightarrow & S & = & z_b z_a^* \end{array}$$

En évaluant S*, nous obtenons :*

$$\mathcal{E}_a(S) = exp(\xi_a(t)) \int_0^t exp\Big(-\xi_a(\tau)\Big) d\xi_b$$

Posons $a(t) = 1$, on obtient des résultats d'évaluation par Scratchpad pour diverses valeurs *polynomiales exponentielles* de $b(t)$. Les commandes (1) à (5) correspondent à l'instanciation des domaines, le type "Entrée" représentant les polynômes-exponentiels à coefficients rationnels. :

```
-----------------------------------------------------------------------
--      instanciation de polynomes exponentiels
-----------------------------------------------------------------------
(5) ->a1:Entree := 2*t*exp(3)$Entree -exp(2)$Entree
```

(5) $$2\,t\,\%e^{3\,t} - \%e^{2\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                                    Time: .667 (IN) + .1 (OT) = .767 sec

(6) ->a2:Entree := 2/5*t*exp(7)$Entree + exp(-5)$Entree -a1
```

(6) $$\left(\frac{2}{5}\right) t\,\%e^{7\,t} - 2\,t\,\%e^{3\,t} + \%e^{2\,t} + \%e^{-5\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                        Time: 1.767 (IN) + .1 (EV) + 1.5 (OT) = 3.367 sec

-----------------------------------------------------------------------
--      evaluation de diverses entrees
-----------------------------------------------------------------------
(7) ->eval(a1)
```

(7) $$\left(t - \frac{1}{2}\right) \%e^{3\,t} - \%e^{2\,t} + \left(\frac{3}{2}\right) \%e^{t}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                                    Time: .967 (IN) + 1.7 (OT) = 2.667 sec

(8) ->eval(a2)
```

$$(8) \quad \left(\left(\frac{1}{15}\right) t - \frac{1}{90}\right) \%e^{7\,t} + \left(-t + \frac{1}{2}\right) \%e^{3\,t} + \%e^{2\,t} - \left(\frac{119}{90}\right) \%e^{t} - \left(\frac{1}{6}\right) \%e^{-5\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                                    Time: .1 (IN) + .6 (OT) = .7 sec

(9) ->eval(a1+a2)
```

$$(9) \quad \left(\left(\frac{1}{15}\right) t - \frac{1}{90}\right) \%e^{7\,t} + \left(\frac{8}{45}\right) \%e^{t} - \left(\frac{1}{6}\right) \%e^{-5\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                                    Time: .167 (IN) + .067 (EV) = .233 sec
```

2.2.3.2 Implantation du domaine des polynômes exponentiels

Nous avons défini le domaine des *polynômes exponentiels* ([28]), de la forme

$$Px(t) = \sum_{i=0}^{n} P_i(t) e^{\alpha_i t}$$

C'est en effet une classe naturelle d'entrées des systèmes dynamiques pour une utilisation en automatique. L'implantation de ce domaine permet d'écrire une *fonction d'intégration spécifique* purement algébrique, plus efficace que celles, plus générales implantées dans la version que nous possédons pour les expressions fonctionnelles (*FonctionalExpressions*) ou les fonction élémentaires (*ElementaryFonction*). Dans la version à venir d'*Axiom*, les domaines *FonctionalExpressions* et *ElementaryFonction* sont remplacés semble-t-il par le domaine *Functions*, ce qui ne remet probablement pas en cause l'intérêt de notre implantation.

Nous utilisons ici une technique d'intégration par parties :

$$\int_0^t \left(\sum_{i=0}^{n} P_i(\tau) e^{\alpha_i \tau}\right) d\tau = \sum_{i=0}^{n} \int_0^t P_i(\tau) e^{\alpha_i \tau} d\tau$$

avec

$$\begin{aligned} \int_0^t P_k(\tau) e^{\alpha_k \tau} d\tau &= P_k(t) \int_0^t e^{\alpha_k \tau} d\tau - \int_0^t P_k'(\tau) e^{\alpha_k \tau} d\tau \\ &= P_k(t) \left[\alpha_k e^{\alpha_k t}\right] - \int_0^t P_k'(\tau) e^{\alpha_k \tau} d\tau \end{aligned}$$

La partie restant à intégrer se calcule par un appel récursif à la fonction d'intégration avec une diminution du degré du polynôme P_k. Nous sommes alors assurés que cet algorithme se termine et que le résultat d'intégration, et donc d'évaluation, existe et reste de type polynôme exponentiel. L'implantation des polynômes exponentiels étant elle-même générique, nous augmentons considérablement la classe des entrées auxquelles le paquetage d'évaluation polynomiale peut s'appliquer, par utilisation notamment d'exponentielles complexes permettant l'évaluation récursive d'entrées trigonométriques.

Exemple 2.2.4 *Une session Scratchpad utilisant les exponentielles complexes (les commandes* (1) *à* (9) *correspondent à l'instanciation des domaines et paquetages : alphabet de codage, polynomes utilisés, fonction d'intégration prise pour l'évaluation).*

```
---------------------------------------------------------------------------
--                        instanciation de la serie generatrice
---------------------------------------------------------------------------
(10) ->sg1: Codage := b*(1+a+a**2+a**3+2*b*a+2*a*b*a+2*b*a**2)
                      + 4*a*b**2*a
```

(10) $$b\,(1+b\,a\,(2+a\,2)+a\,(1+b\,a\,2+a\,(1+a)))+a\,b\,b\,a\,4$$

```
Type: XRecursivePolynomial(OrderedVarlist [a,b],Gaussian RationalNumber)
                Time: 6.5 (IN) + 1.333 (EV) + 11.583 (OT) = 19.417 sec
---------------------------------------------------------------------------
--                        instanciation des entrees
---------------------------------------------------------------------------
(11) ->c1:k := gauss(0,-1)$k
```

(11) $$-\%i$$

```
                                                Type: Gaussian RationalNumber
                                    Time: .367 (IN) + .85 (OT) = 1.217 sec
(12) ->u1:=  (1/2)*(exp(c1)$Entree + exp(-c1)$Entree)
```

(12) $$\left(\frac{1}{2}\right)\,\%e^{-\%i\,t}+\left(\frac{1}{2}\right)\,\%e^{\%i\,t}$$

```
Type: PolynomialExponential(t,Gaussian RationalNumber,
                                    Polynomial[t]Gaussian RationalNumber)
                    Time: 1.833 (IN) + .033 (EV) + 1.133 (OT) = 3.0 sec

(13) ->u2:= 1$Entree
```

(13) $$1$$

```
Type: PolynomialExponential(t,Gaussian RationalNumber,
                                    Polynomial[t]Gaussian RationalNumber)
                                        Time: .1 (IN) + .2 (OT) = .3 sec
---------------------------------------------------------------------------
--                        evaluation
---------------------------------------------------------------------------

(14) ->xeval(sg1,[a,b],[u1,u2])$Evpkg
```

$$
\begin{aligned}
& \left(-\left(\tfrac{1}{8}\right) t^2 + \left(-\tfrac{1}{8} + \left(\tfrac{1}{8}\right) \%i\right) t + \tfrac{7}{16} + \left(\tfrac{3}{16}\right) \%i\right) \%e^{-2\ \%i\ t} \\
+ & \\
& \left(\left(\tfrac{1}{2}\right) \%i\ t^2 + \left(1 + \left(\tfrac{27}{16}\right) \%i\right) t + \tfrac{51}{32} - \left(\tfrac{3}{2}\right) \%i\right) \%e^{-\%i\ t} \\
+ & \\
& \left(-\left(\tfrac{1}{8}\right) t^2 + \left(-\tfrac{1}{8} - \left(\tfrac{1}{8}\right) \%i\right) t + \tfrac{7}{16} - \left(\tfrac{3}{16}\right) \%i\right) \%e^{2\ \%i\ t} \\
+ & \\
& \left(-\left(\tfrac{1}{2}\right) \%i\ t^2 + \left(1 - \left(\tfrac{27}{16}\right) \%i\right) t + \tfrac{51}{32} + \left(\tfrac{3}{2}\right) \%i\right) \%e^{\%i\ t} \\
+ & \\
& \left(\tfrac{3}{2}\right) t - \tfrac{287}{72} + \left(-\left(\tfrac{1}{48}\right) \%i\ t - \tfrac{11}{288}\right) \%e^{-3\ \%i\ t} \\
+ & \\
& \left(\left(\tfrac{1}{48}\right) \%i\ t - \tfrac{11}{288}\right) \%e^{3\ \%i\ t}
\end{aligned}
\tag{14}
$$

```
Type: PolynomialExponential(t,Gaussian RationalNumber,
                            Polynomial[t]Gaussian RationalNumber)
                Time: .233 (IN) + 1.0 (EV) + .067 (OT) = 1.3 sec
```

On remarque sur cet exemple, que l'absence d'un ordre défini sur les nombres complexes entraine l'utilisation d'algorithmes moins performants (1.3 sec d'évaluation) et fournit une expression peu agréable du résultat : l'affichage n'est pas rangé par ordre croissant ou décroissant des puissances. Une expression de `u1` dans un autre ordre aurait donné une expression de la sortie également dans un autre ordre. Le domaine des polynômes exponentiels n'est pas seulement utile à l'évaluation des séries génératrices polynomiales : Nous savons que l'évaluation d'une fraction rationnelle (non commutative) pour une entrée polynomiale exponentielle existe toujours (il n'y a pas de e^{t^k} avec $k > 1$) et reste du même type (voir par exemple [28]). Nous avons par conséquent un type d'entrée *très général*, que nous *simulons graphiquement* dans le cas réel et pour lesquelles nous pouvons évaluer *divers types de séries* avec l'assurance de l'existence de la sortie.

Le graphe d'héritage des polynômes exponentiels est décrit par la figure 2.1. Cette implantation signifie que tout polynôme exponentiel $Px(t)$ sera défini avec :

- $P_i(t)$ des polynômes possédant toutes les primitives de dérivation et à coefficients dans un anneau, pour certaines opérations (addition, soustraction, égalité), l'algorithme dépend directement du type de R : si R est ordonné, on utilisera les algorithmes issus du domaine $FreeModule$, sinon on utilise des algorithmes moins rapides que nous avons implantés mais qui fonctionnent sur n'importe quel type d'anneau.

- α_i des coefficients du même type que ceux des polynômes, ce qui nous assure une totale compatibilité de type lors de l'exécution de certains algorithmes comme celui d'intégration décrit précédemment et pour lequel on multiplie les polynômes P_k par les éléments α_k

L'ensemble des $Px(t)$ étant un anneau différentiel et une algèbre sur les polynômes et sur leurs coefficients, il a les propriétés suffisantes pour être un paramètre du paquetage d'évaluation.

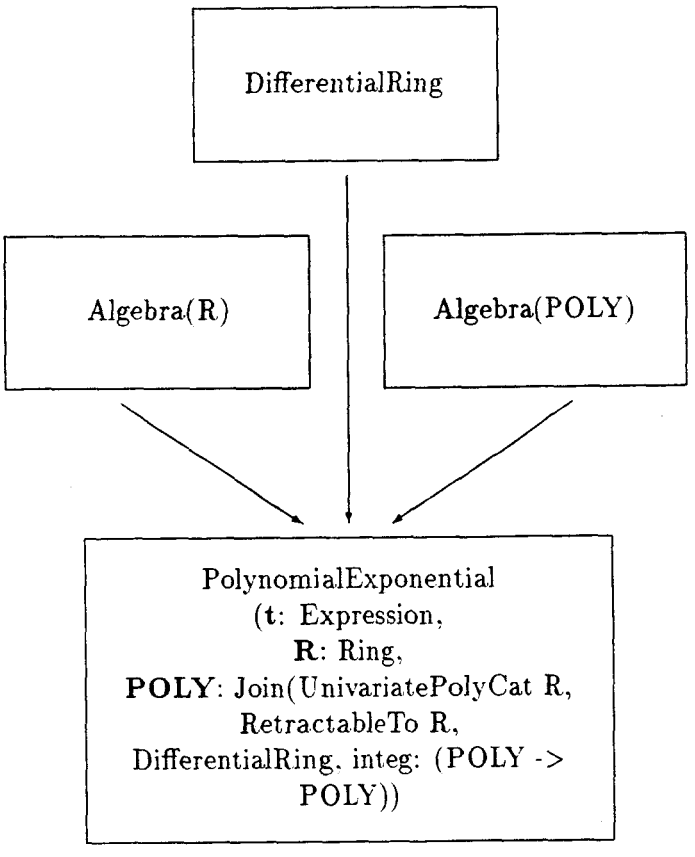


Figure 2.1 : Graphe d'héritage des Polynômes exponentiels

2.2.3.3 Utilisation des polynômes exponentiels

Le début du code :

```
)abb domain POLEXP PolynomialExponential
PolynomialExponential(t:E,R,POLY,integ): pub == priv where

  integ: POLY -> POLY
  R : Ring
  POLY : Join(UnivPolyCat R ,RetractableTo R ,DifferentialRing)

  E        ==>          Expression
  UB       ==>          UnivariateBernsteinPolynomial(t,R)
  FE       ==>          FunctionalExpression R
  SE       ==>          SortedExpressions
  INTDOM   ==>          IntegralDomain

  pub ==  Join(Algebra(POLY),Algebra(R),DifferentialRing) with

    if R has Field then
```

```
    integrate     : $                    -> $
  if R has INTDOM then
    coerce        : $                    -> FE
  coerce          : POLY                 -> $
  exp             : R                    -> $
  leadingCoef     : $                    -> POLY
  listcoef        : $                    -> List POLY
  listOfTerm      : $                    -> List R
  leadingMonomial:$                      -> $
  map             : ((POLY -> POLY),$)   -> $
  reductum        : $                    -> $
  leadingTerm     : $                    -> R
  cte             : $                    -> R
  coef            : ($,R)                -> POLY
  delete          : (R,$)                -> $

 priv == add

------------------------------------------------------------------------
--  Representation interne
------------------------------------------------------------------------

  Term := Record(k:R,c:POLY)
  Rep  := List Term
```

Une session Scratchpad

Les commandes (1) à (3) sont des instanciations des domaines Up (polynômes en une variable) et Entree (polynômes exponentiels).

```
------------------------------------------------------------------------
--                  creation d'un polynome exponentiel
------------------------------------------------------------------------
(4) -> pol:Up := 2/5* t**3 - 3*t**2 + 4*t-8/3
```

$$(4)\qquad \left(\frac{2}{5}\right) t^3 - 3\,t^2 + 4\,t - \frac{8}{3}$$

```
                                   Type: Polynomial[t]RationalNumber
                  Time: 1.7 (IN) + .167 (EV) + .167 (OT) = 2.033 sec

(5) -> px1:Entree := pol*exp(1)$Entree - pol*exp(2)$Entree + 2*pol
```

$$(5)\qquad \left(-\left(\frac{2}{5}\right) t^3 + 3\,t^2 - 4\,t + \frac{8}{3}\right) \%e^{2\,t} + \left(\left(\frac{2}{5}\right) t^3 - 3\,t^2 + 4\,t - \frac{8}{3}\right) \%e^{t} + \left(\frac{4}{5}\right) t^3 - 6\,t^2 + 8\,t - \frac{16}{3}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                  Time: 1.067 (IN) + .2 (EV) + .167 (OT) = 1.433 sec
```

```
(6) -> px2:Entree := 1$Entree
```

(6) 1

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                                                Time: .1 (IN) = .1 sec
```

Les commandes (7) à (12) correspondent à des instanciations de l'alphabet de codage, du type du codage et du paquetage d'évaluation.

```
-------------------------------------------------------------------------
-- evaluation pour des entrees de type polynome exponentiel
-------------------------------------------------------------------------
(13) -> sg: Codage := b+2*b*b*a
```

(13) $b\,(1 + b\,a\,2)$

```
      Type: XRecursivePolynomial(OrderedVarlist [a,b],RationalNumber)
                    Time: .9 (IN) + .1 (EV) + .067 (OT) = 1.067 sec
```

```
(14) -> xeval(sg,[a,b],[px1,px2])$Evpkg
```

(14)
$$\begin{array}{l}\left(-\left(\frac{1}{5}\right)t^5+2\,t^4-6\,t^3+\left(\frac{31}{3}\right)t^2-\left(\frac{31}{3}\right)t+\frac{31}{6}\right)\%e^{2\,t}\\ +\left(\left(\frac{2}{5}\right)t^5-5\,t^4+24\,t^3-\left(\frac{224}{3}\right)t^2+\left(\frac{448}{3}\right)t-\frac{448}{3}\right)\%e^{t}\\ +\left(\frac{2}{15}\right)t^6-\left(\frac{6}{5}\right)t^5+2\,t^4-\left(\frac{16}{9}\right)t^3+t+\frac{865}{6}\end{array}$$

```
Type: PolynomialExponential(t,RationalNumber,Polynomial[t]RationalNumber)
                                    Time: .2 (IN) + .167 (EV) = .367 sec
```

2.2.4 L'opérateur de transport

Nous savons que la technique d'approximation rationnelle des séries formelles n'est pas toujours efficace. En effet, on trouve des exemples, comme la série $S = (z_0 z_1)^*$ que l'on ne sait pas évaluer sans spécifier l'entrée en fonction du temps. Nous utilisons alors les approximations nilpotentes structurelle ([28, 31]).

Cette technique consiste à factoriser "l'opérateur de transport" du système en un produit infini d'exponentielles de champs de vecteurs, autrement dit en une composition à priori infinie de systèmes dynamiques sur une seule entrée (à priori virtuelle).

Soit donc à nouveau le système

$$(\Sigma)\begin{cases}\dot{q}(t) &= \sum_{z\in Z} a^z(t) A_z(q)\\ y(t) &= h(q(t))\end{cases}$$

avec

$$A_z = \sum_{k=1}^{n} A_z^k(q)\frac{\partial}{\partial q_k},$$

notation
Dans la suite, $\mathcal{Y}$ désigne l'homomorphisme d'algèbres de Lie défini comme suit :

$$\forall z \in Z, \quad \mathcal{Y}(z) = A_z$$

Definition 2.2.1 *L'algèbre de Lie du système (Σ) est dite nilpotente d'ordre $k \geq 0$ si*

$$\forall P \in Lie < Z >, \quad deg(P) \geq k, \quad \mathcal{Y}(P) = 0$$

Definition 2.2.2 *La série génératrice associée à la fonction d'observation h relative au système (Σ), en q est la série formelle à coefficients dans $\mathbb{R}$:*

$$\sigma h_q = \sum_{w \in Z^*} \Big(\mathcal{Y}(w) \circ h_q\Big) w$$

D'après la formule fondamentale de M. Fliess, la série σh_q caractérise complètement le comportement entrée/sortie, localement en q, du système dynamique $(\{A_z\}_{z \in Z}, h)$. En évaluant cette série génératrice, on obtient la sortie du système :

$$y(t) = \mathcal{E}_a(\sigma h_q) = \sum_{w \in Z^*} \mathcal{E}_a(w)\mathcal{Y}(w) \circ h_q$$

Cette formule présente alors $y(t)$ comme l'image de la série double $\sum_{w \in Z^*} w \otimes w$ par *l'opérateur structurel* ([28, 31])

$$\mathcal{E}_a \otimes \mathcal{Y}$$

et on a ([28, 30, 31]) :

$$y(t) = \Big[\sum_{w \in Z^*} \mathcal{E}_a(w)\mathcal{Y}(w)\Big] \circ h_q = \Big[\Big(\mathcal{E}_a \otimes \mathcal{Y}\Big)\Big(\sum_{w \in Z^*} w \otimes w\Big)\Big] \circ h_q$$

Definition 2.2.3 *On appelle* opérateur de transport *de l'entrée $a = (a^{z_0}, a^{z_1}, ..., a^{z_n})$ sur l'intervalle de temps $[0, t]$ la série définie par :*

$$\mathcal{H}_a = \sum_{w \in Z^*} \mathcal{E}_a(w)\mathcal{Y}(w)$$

Théorème 2.2.1

$$\begin{aligned} \mathcal{H}_a &= \prod_{l \in L} exp(\xi_l \mathcal{Y}(Q_l)) \quad \textit{ordre décroissant} \\ &= \prod_{\tilde{l} \in \tilde{L}} exp(\xi_{\tilde{l}} \mathcal{Y}(Q_{\tilde{l}})) \quad \textit{ordre croissant} \end{aligned}$$

preuve
D'après le théorème de factorisation de la série double, on a :

$$\begin{aligned} \Big(\mathcal{E}_a \otimes \mathcal{Y}\Big)\Big(\sum_{w \in Z^*} w \otimes w\Big) &= \Big(\mathcal{E}_a \otimes \mathcal{Y}\Big)\Big(\prod_{\tilde{l} \in \tilde{L}} S_l \otimes Q_l\Big) \\ &= \prod_{\tilde{l} \in \tilde{L}} \mathcal{E}_a(S_l)\mathcal{Y}(Q_l) \\ &= \prod_{\tilde{l} \in \tilde{L}} exp(\xi_{\tilde{l}} \mathcal{Y}(Q_{\tilde{l}})) \end{aligned}$$

Il en est de même pour l'autre formule.

On note $\widetilde{L}_k, k \geq 0$, l'ensemble des mots de Lyndon miroirs de longueur inférieure ou égale à k.

2.2.4.1 Approximation de l'opérateur de transport

Definition 2.2.4 *On appelle approximant nilpotent d'ordre k de l'opérateur de transport $\mathcal{H}_a$ du système dynamique (Σ), l'opérateur différentiel :*

$$\mathcal{A}_k(\mathcal{H}_a) = \prod_{l \in \widetilde{L}_k} exp(\xi_l \mathcal{Y}(Q_l))$$

Cette approximation nilpotente sera exacte (c'est-à-dire $\mathcal{A}_k(\mathcal{H}_a) = \mathcal{H}_a$) si l'algèbre de Lie engendrée par les champs de vecteurs $\{A_z\}_{z \in Z}$ est nilpotente d'ordre n avec $n \leq k$.

2.2.4.2 Une application : l'équation de Düffing

Rappelons ici l'équation différentielle de Düffing :

$$\ddot{y} + \dot{y} + y + y^3 = a^{z_1}$$

avec les deux champs de vecteurs :

$$\begin{cases} A_{z_0} = q_2(t)\dfrac{\partial}{\partial q_1} - (q_1^3(t) + q_1(t) + q_2(t))\dfrac{\partial}{\partial q_2} \\ A_{z_1} = \dfrac{\partial}{\partial q_2} \end{cases}$$

2.2.4.2.1 Calcul des approximations nilpotentes

A l'ordre 1, nous obtenons l'expression :

$$y_1(t) = e^{tA_{z_0}} \circ e^{\xi_{z_1}(t)A_{z_1}} \circ h_{|_{q(0)}}$$

avec

$$\begin{aligned} \xi_{z_0}(t) &= t \\ \xi_{z_1}(t) &= \int_0^t a^z(\tau)d\tau \end{aligned}$$

A l'ordre 2, l'expression devient :

$$y_2(t) = e^{tA_{z_0}} \circ e^{\xi_{z_1 z_0}(t)[A_{z_1}, A_{z_0}]} \circ e^{\xi_{z_1}(t)A_{z_1}} \circ h_{|_{q(0)}}$$

avec

$$\begin{aligned} [A_{z_1}, A_{z_0}] &= A_{z_1}A_{z_0} - A_{z_0}A_{z_1} \\ &= \frac{\partial}{\partial q_1} - \frac{\partial}{\partial q_2} \end{aligned}$$

et

$$\xi_{z_1 z_0} = \int_0^t \left(\int_0^\tau a^z(\sigma)d\sigma \right) d\tau$$

Enfin, à l'ordre 3, l'expression devient :

$$y_3(t) = e^{tA_{z_0}} \circ e^{\xi_{z_1 z_0^2}(t)[[A_{z_1},A_{z_0}],A_{z_0}]} \circ e^{\xi_{z_1 z_0}(t)[A_{z_1},A_{z_0}]} \circ e^{\xi_{z_1^2 z_0}(t)[A_{z_1},[A_{z_1},A_{z_0}]]} \circ e^{\xi_{z_1}(t)A_{z_1}} \circ h_{|q(0)}$$

avec

$$[A_{z_1},[A_{z_1},A_{z_0}]] = 0$$
$$[[A_{z_1},A_{z_0}],A_{z_0}] = -\frac{\partial}{\partial q_1} - 3q_1^2\frac{\partial}{\partial q_2}$$

et

$$\xi_{z_1^2 z_0}(t) = \int_0^t \xi_{z_1}^2(\tau)d\tau$$
$$\xi_{z_1 z_0^2}(t) = \int_0^t \xi_{z_1 z_0}(\tau)d\tau$$

Il reste alors à fixer l'entrée $a^z(t)$ et à exprimer le système dynamique par ses équations d'état afin de le simuler graphiquement (on peut voir pour cela les exemples de simulation pages 75 à 78).

2.3 Le problème du "motion planning"

2.3.1 Présentation du problème

Considérons le système:

$$(\Sigma) \qquad \dot{x} = \sum_{i=0}^{n} a^{z_i}(t) f_i(x)$$

Le problème du "motion planning exact" est le suivant :
Etant donnés deux vecteurs d'état p et q, trouver l'entrée $a = (a^{z_1},\ldots,a^{z_n})$ qui conduit exactement de l'état p à l'état q.

G.Lafferriere et H.J. Sussman ont proposé ([47]) un algorithme pour résoudre ce problème sous les conditions suivantes :

1. Le système est sans dérive ($a^{z_0}(t) \equiv 0$).
2. Les f_i, $i = 0\ldots n$ sont des champs de vecteurs analytiques sur $\mathbb{R}^N$.
3. (Σ) est complètement controllable.
4. Les champs de vecteurs sont complets.
5. L'algèbre de Lie de contrôle engendrée par les champs de vecteurs f_i est nilpotente.

L'algorithme se décompose alors en deux étapes:

1. Calculer dans le groupe de Lie du système une transformation $\mathcal{G}$ qui conduit géométriquement de p à q. Ils trouvent alors une trajectoire γ exprimée dans un système dit "étendu" c'est à dire dont le degré peut être supérieur à celui du système de départ. Cette trajectoire est exprimée dans la *base de Hall*[1] ([5]) qui est une base de l'algèbre de Lie que nous n'utilisons pas ici.

[1] On trouve dans Koseleff [36] une étude comparative des bases de Hall et de Lyndon

2. Calculer dans la classe d'entrées données (ici constantes par morceaux) l'expression de l'entrée a pour laquelle le système réalise $\mathcal{G}$ et factoriser par les formules de Campbel-Hausdorff (voir [5]).

Ces travaux ont été repris très récemment par G.Jacob ([31]). Il réalise la première partie de l'algorithme en exprimant la trajectoire γ du système étendu dans la base de Lyndon miroir qui offre certains avantages :

1. des calculs plus simples (voir[31]), certains exemples peuvent même être traités "à la main".
2. une réduction du nombre de "morceaux" des entrées, G. Jacob propose un exemple où l'on passe de 26, pour la solution de Lafferrière et Sussman à 3 morceaux (voir [31])
3. La possibilité de traiter les systèmes *avec dérive* : la partie autonome a^{z_0}, ou dérive, du système force à commander un système "en mouvement".

Nous nous intéressons ici à cette technique de résolution et nous en donnons les simulations graphiques.

2.3.2 Entrées constantes par morceaux

On définit une entrée constante par morceaux $a = (a^{z_1}, \ldots, a^{z_n})$ sur l'intervalle $[0, T]$ par :
Soient $t_1, \ldots, t_k, \ldots$ tels que $0 < t_1 < \ldots < t_k < \ldots < T$ alors :

$$\forall t \in [t_k, t_{k+1}] \quad a^{z_i}(t) = a_i^{k+1} \in I\!R$$

On calcule alors l'expression de ξ_l pour tout $l \in \widetilde{L}$ par :
Soit $l = l_1^{i-1} l_2^{i-2} \ldots l_m^{i-m} z$ et $t_k < t_k + \sigma < t_{k+1}$ alors

$$\xi_l(t_k + \sigma) = \xi_l(t_k) + \frac{1}{i_1! i_2! \ldots i_m!} \int_{t_k}^{t_k+\sigma} \xi_{l_1}^{i_1}(t_k + \tau) \xi_{l_2}^{i_2}(t_k + \tau) \ldots \xi_{l_m}^{i_m}(t_k + \tau) a_z^{(k+1)} d\tau$$

Plus simplement, les coefficients $\xi_l(t_k + 1)$ dépendent uniquement des nombres :

$$A_i^{(h)} = a_i^{(h)}(t_{k+1} - t_k)$$

avec h, nombre de "morceaux" de l'entrée.

2.3.3 Le "motion planning" sans dérive

Rappelons le Schéma d'algorithme du motion planning tel qu'il a été présenté par G.Jacob et C.Guyon dans [26, 31] :

On considère un système (Σ) qui remplit les conditions posées par Lafferriere et Sussman et qui est nilpotent d'ordre k.
Soient $Z = \{z_1, z_2, \ldots, z_n\}$ un alphabet ordonné et $\widetilde{L}_k = \{l_1, l_2, \ldots, l_m\}$ l'ensemble ordonné des mots de Lyndon miroirs sur Z de longueur inférieure ou égale à k.
On note $g_i = \mathcal{Y}(Q_{l_i})$ pour $i \in \{1, 2, \ldots, m\}$

Etape 1
Nous effectuons la première étape comme décrit en 2.3.1 :

- On trace une courbe $\gamma(t)$, $t \in [0,T]$ allant de p ($t=0$) en q ($t=T$).
- On calcule les équations du système étendu : On exprime $\gamma(t)$ comme combinaison linéaire des champs de vecteurs $\mathcal{Y}(Q_l)$, $l \in \tilde{L}_k$. On obtient une "trajectoire d'essai" n'ayant pas de signification physique :

$$\dot{\gamma}(t) = \sum_{1 \leq i \leq m} a^{z_i}(t) g_i\Big(\gamma(t)\Big)$$

- On intègre ce système pour obtenir une expression de q :

$$q = \Big[\prod_{1 \leq i \leq m} exp(c_i g_i) \circ Id\Big]_p$$

avec $c_i = c_i(T)$ des coefficients constants.

Etape 2
On identifie les c_i avec les $\xi_i(T)$. On obtient un système de m équations dont le nombre d'inconnues dépend du nombre de morceaux de l'entrée.

2.3.4 Un exemple de résolution

Nous traitons ici le problème du "monocycle" donné par le système :

$$(\Sigma)\begin{cases} \dot{q_1} &= cos(q_3)u_1 \\ \dot{q_2} &= sin(q_3)u_1 \\ \dot{q_3} &= u_2 \end{cases}$$

où q_1 et q_2 représentent les coordonnées de la roue dans le plan et q_3 son angle.

Approximation nilpotente du système

On procède à une approximation nilpotente d'ordre 2 de (Σ). On calcule pour cela les trois champs de vecteurs :

$$f_0 = cos(q_3)\frac{\partial}{\partial q_1} + sin(q_3)\frac{\partial}{\partial q_2}$$
$$f_1 = \frac{\partial}{\partial q_3}$$
$$f_{10} = [f_1, f_0] = -sin(q_3)\frac{\partial}{\partial q_1} + cos(q_3)\frac{\partial}{\partial q_2}$$

qui sont linéairement indépendants. Le système étendu approximant s'écrit alors :

$$\dot{x}(t) = c_1(t)f_1 + c_{10}(t)f_{10} + c_0(t)f_0$$

Calcul de la trajectoire d'essais

On peut calculer l'entrée $(c_1(t), c_{10}(t), c_0(t))$ (donc pour le système étendu), pour suivre une

trajectoire différentiable arbitraire conduisant de $p = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ à $q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$ à vitesse constante $(\dot{\gamma}(t) = Cte)$.
En effet, choisissons une "trajectoire d'essais" dans l'espace d'état, conduisant de p à q sur l'intervalle de temps $[0, T]$:

$$\gamma(t) = \begin{pmatrix} \gamma_1(t) \\ \gamma_2(t) \\ \gamma_3(t) \end{pmatrix}$$

d'où

$$\dot{\gamma}(t) = \begin{pmatrix} \dot{\gamma}_1(t) \\ \dot{\gamma}_2(t) \\ \dot{\gamma}_3(t) \end{pmatrix}$$

Il suffit de résoudre le système d'équations linéaires :

$$\dot{\gamma}(t) = c_1(t) f_1(\gamma) + c_{10}(t) f_{10}(\gamma) + c_0(t) f_0(\gamma)$$

qui s'écrit encore :

$$\begin{pmatrix} \dot{\gamma}_1(t) \\ \dot{\gamma}_2(t) \\ \dot{\gamma}_3(t) \end{pmatrix} = \begin{pmatrix} cos(\gamma_3) & -sin(\gamma_3) & 0 \\ sin(\gamma_3) & cos(\gamma_3) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0(t) \\ c_{10}(t) \\ c_1(t) \end{pmatrix}$$

Ce qui se résoud par inversion de matrice :

$$\begin{pmatrix} c_0(t) \\ c_{10}(t) \\ c_1(t) \end{pmatrix} = \begin{pmatrix} cos(\gamma_3) & -sin(\gamma_3) & 0 \\ sin(\gamma_3) & cos(\gamma_3) & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \dot{\gamma}_1(t) \\ \dot{\gamma}_2(t) \\ \dot{\gamma}_3(t) \end{pmatrix}$$

c'est-à-dire, finalement :

$$\begin{aligned} c_0(t) &= cos(\gamma_3)\dot{\gamma}_1(t) + sin(\gamma_3)\dot{\gamma}_2(t) \\ c_{10}(t) &= -sin(\gamma_3)\dot{\gamma}_1(t) + cos(\gamma_3)\dot{\gamma}_2(t) \\ c_1(t) &= \dot{\gamma}_3(t) \end{aligned}$$

Codage du système initial.

On code les champs de vecteurs du système initial sur l'alphabet : $Z = \{z_0, z_1\}$.
On génère les mots de Lyndon de longueur inférieure ou égale à l'ordre de nilpotence (ici 2) : on obtient les mots $z_0, z_0 z_1, z_1$

Codage du système étendu.

On effectue un surcodage sur l'alphabet $X = \{X_0, X_1, X_2\}$ en posant :

$$\begin{aligned} X_0 &\mapsto f_0 \\ X_1 &\mapsto f_{01} \\ X_2 &\mapsto z_1 \end{aligned}$$

avec

$$deg(X_0) = 1,\ deg(X_1) = 2,\ deg(X_2) = 1$$

On calcule les mots de Lyndon à l'ordre 2 sur l'alphabet X :

$$X_0 < X_0X_1 < X_0X_2 < X_1 < X_1X_2 < X_2$$

mais $deg(X_0X_1) = 3$ et $deg(X_1X_2) = 3$ sur l'alphabet Z, on ne conserve alors que les mots :

$$X_0 < X_0X_2 < X_1 < X_2$$

notation :
On note $k_l(t)$ les coordonnées du système initial.
On note $h_m(t)$ les coordonnées du système étendu.

Evaluation des S_l pour le système étendu.

Nous avons calculé l'entrée du système étendu :

$$\begin{aligned} B_0(t) &= c_0(t) = cos(\gamma_3)\dot{\gamma}_1(t) + sin(\gamma_3)\dot{\gamma}_2(t) \\ B_1(t) &= c_{10}(t) = -sin(\gamma_3)\dot{\gamma}_1(t) + cos(\gamma_3)\dot{\gamma}_2(t) \\ B_2(t) &= c_1(t) = \dot{\gamma}_3(t) \end{aligned}$$

On en déduit les coordonnées de Lyndon du système étendu :

$$\begin{aligned} h_0 &= \int_0^T B_0(\tau)d\tau &&= \tfrac{1}{q_3}(q_1 sin(q_3) - q_2 cos(q_3) + q_2) \\ h_1 &= \int_0^T B_1(\tau)d\tau &&= \tfrac{-1}{q_3}(q_2 sin(q_3) - q_1 cos(q_3) + q_1) \\ h_2 &= \int_0^T B_2(\tau)d\tau &&= q_3 \\ h_{02} &= \int_0^T H_1(\tau)B_0(\tau)d\tau &&= \tfrac{1}{q_3}\Big((q_1q_3 + q_2)sin(q_3) + (-q_2q_3 + q_1)cos(q_3) - q_1\Big) \end{aligned}$$

Ces coordonnées sont des valeurs numériques obtenues en fixant le point d'arrivée q et la "date d'arrivée" T. La trajectoire d'essai est donc décrite par le produit d'exponentielles suivants :

$$e^{h_0X_0}e^{h_{02}[X_0,X_2]}e^{h_1X_1}e^{h_2X_2}$$

c'est à dire, sur l'alphabet Z :

$$e^{h_0z_0}e^{h_{02}[z_0,z_1]}e^{h_1[z_0,z_1]}e^{h_2z_1}$$

On obtient finalement le développement à l'ordre 2 :

$$e^{h_0z_0}e^{(h_{02}+h_1)[z_0,z_1]}e^{h_2z_1}$$

Evaluation des S_l pour le système initial.

La trajectoire du système initial est décrite de la même façon par un produit d'exponentielles

$$e^{k_0z_0}e^{k_{01}[z_0,z_1]}e^{k_1z_1}$$

qu'il faut alors identifier à celui calculé pour la trajectoire d'essai.
On choisit donc un type d'entrées paramétrées : ici des entrées constantes par morceaux :

$$u_1(t) = \begin{cases} a_1 & pour \quad 0 \leq t < \frac{T}{2} \\ a_2 & pour \quad \frac{T}{2} \leq t < T \end{cases}$$

$$u_2(t) = \begin{cases} b_1 & pour \quad 0 \leq t < \frac{T}{2} \\ b_2 & pour \quad \frac{T}{2} \leq t < T \end{cases}$$

pour lesquelles on obtient les coordonnées de Lyndon suivantes (obtenues par Scratchpad en 2.2.2.5) :

$$k_0 = A_1 + A_2$$
$$k_{01} = \frac{1}{2}A_1B_1 + A_2B_1 + \frac{1}{2}A_2B_2$$
$$k_1 = B_1 + B_2$$

où $A_1 = a_1\frac{T}{2}$, $A_2 = a_2\frac{T}{2}$ et $B_1 = b_1\frac{T}{2}$, $B_2 = b_2\frac{T}{2}$

Identification des entrées

On obtient les entrées en résolvant le système d'équations :

$$A_1 + A_2 = h_0$$
$$\frac{1}{2}A_1B_1 + A_2B_1 + \frac{1}{2}A_2B_2 = h_{01}$$
$$B_1 + B_2 = h_1$$

On résoud formellement (une fois pour toutes) ce système particulier en utilisant les bases de Gröbner. On trouve ici 4 inconnues pour 3 équations. Nous avons donc choisi de fixer la vitesse angulaire ($B_1 = B_2 = \frac{h_1}{2}$) ce qui correspond à un système avec dérive. Il reste alors à remplacer T, h_0, h_{01} et h_1 par leurs valeurs numériques pour obtenir les valeurs des entrées.

Nous donnons sur les figures 2.2, 2.3,2.4,2.5 et 2.6 des exemples de simulation de la trajectoire pour $T = 1$, pour des cibles respectivement égales à

$$\begin{pmatrix} -1 \\ 2 \end{pmatrix}, \begin{pmatrix} 10 \\ 20 \end{pmatrix}, \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 2 \\ -3 \end{pmatrix} \; et \; \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

et un angle d'arrivée égal à $\frac{\pi}{4}$ (le départ est toujours en 0, le symbole $\otimes$ représente le point visé).

Le calcul numérique nous donne, pour l'ensemble de ces simulations, un angle d'arrivée égal à 0.76969045 ($\frac{\pi}{4} = 0.7853982$. Le tableau suivant nous donne les points d'arrivée de la simulation :

point cible	$\begin{pmatrix} -1 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 10 \\ 20 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 2 \\ -3 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 0 \end{pmatrix}$
point atteint	$\begin{pmatrix} -0.93 \\ 1.99 \end{pmatrix}$	$\begin{pmatrix} 7.95 \\ 22.80 \end{pmatrix}$	$\begin{pmatrix} 1.63 \\ 3.48 \end{pmatrix}$	$\begin{pmatrix} 1.93 \\ -2.93 \end{pmatrix}$	$\begin{pmatrix} 1.784 \\ 0.285 \end{pmatrix}$

On voit, sur ce tableau, que l'approximation nilpotente d'ordre 2 donne déjà des résultats proches du but recherché. On remarque cependant que la précision n'est pas la même suivant le point cible fixé : on obtient de bon résultats pour les points cibles $\begin{pmatrix} -1 \\ 2 \end{pmatrix}$ et $\begin{pmatrix} 2 \\ -3 \end{pmatrix}$, alors que la cible $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$, bien que plus proche, n'est pas atteinte de manière aussi précise.

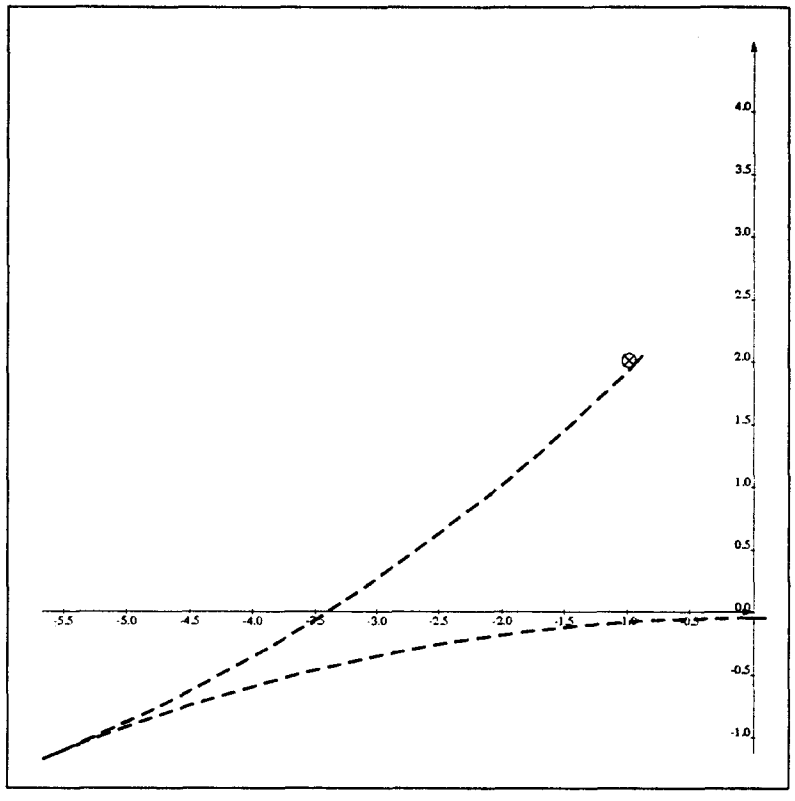


Figure 2.2 : trajectoire de l'unicycle. Arrivée en (-1,2)

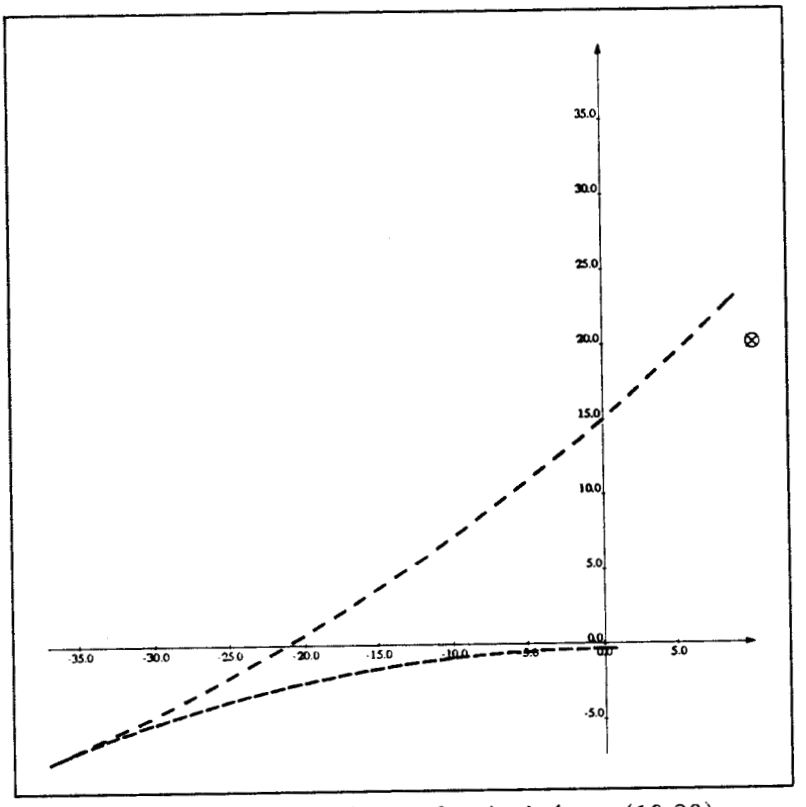


Figure 2.3 : trajectoire de l'unicycle. Arrivée en (10,20)

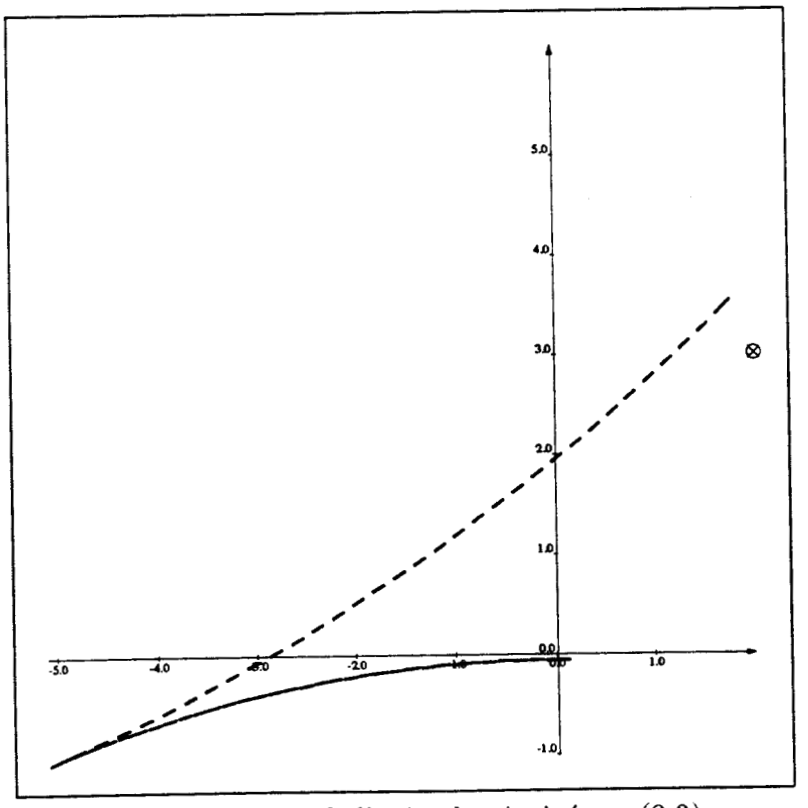


Figure 2.4 : trajectoire de l'unicycle. Arrivée en (2,3)

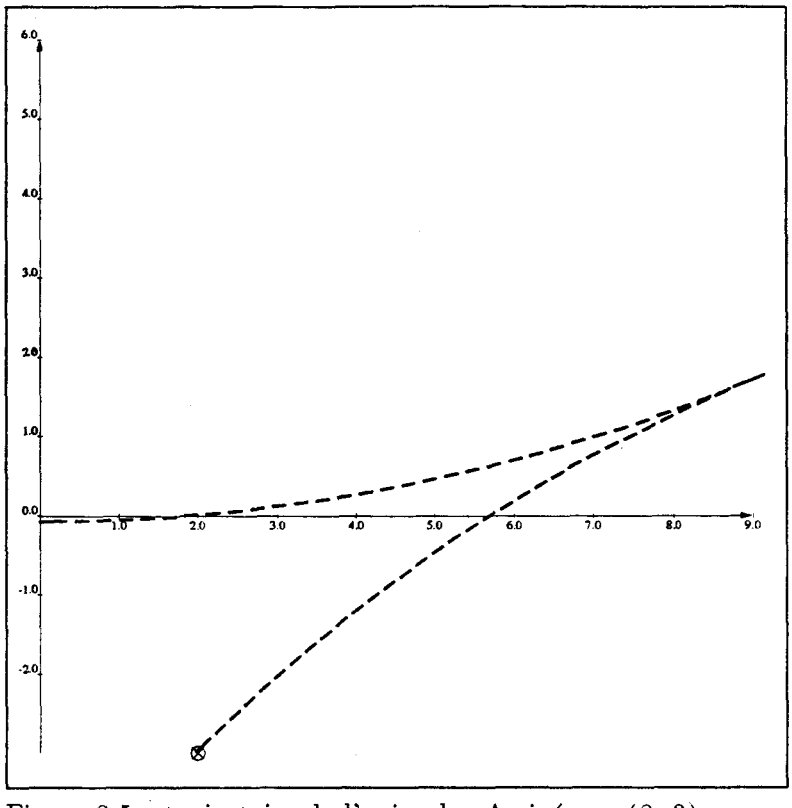


Figure 2.5 : trajectoire de l'unicycle. Arrivée en (2,-3)

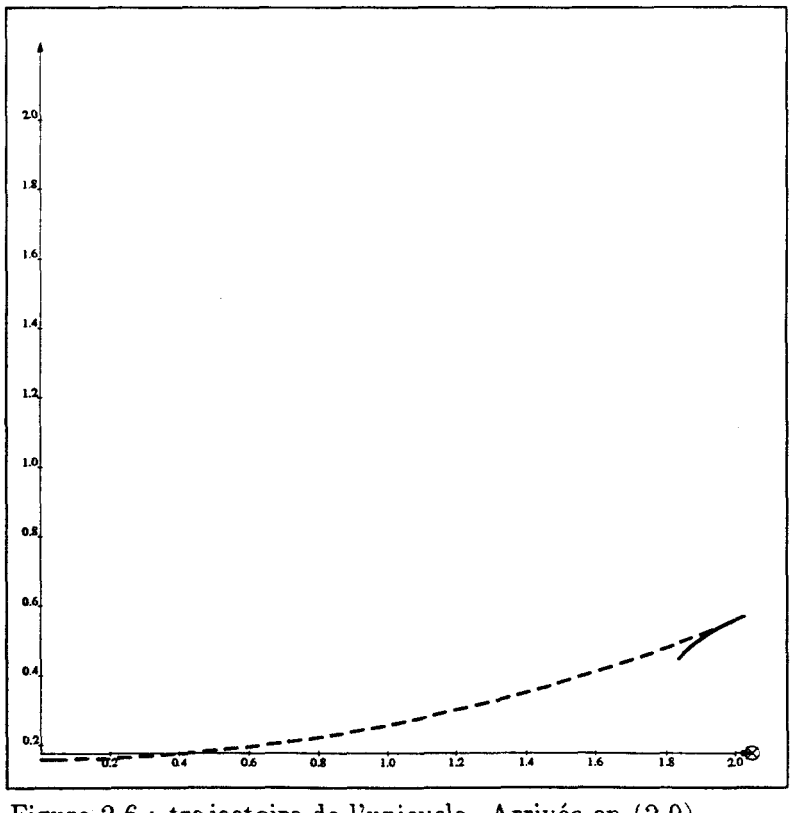


Figure 2.6 : trajectoire de l'unicycle. Arrivée en (2,0)

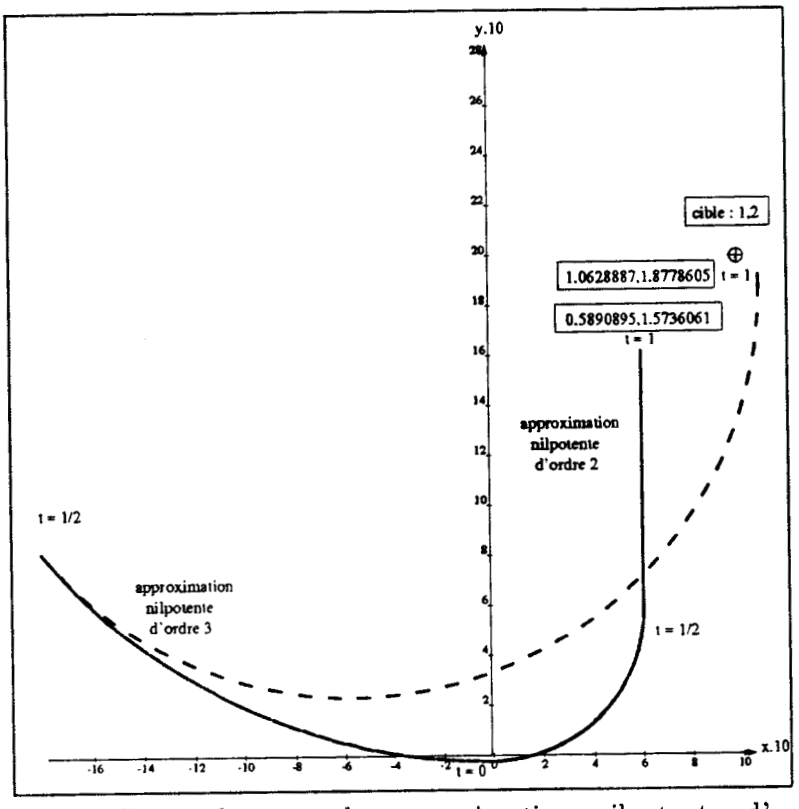


Figure 2.7 : trajectoire de l'unicycle. pour des approximations nilpotentes d'ordres 2 et 3. Entrées constantes par morceaux. Arrivée en (1,2)

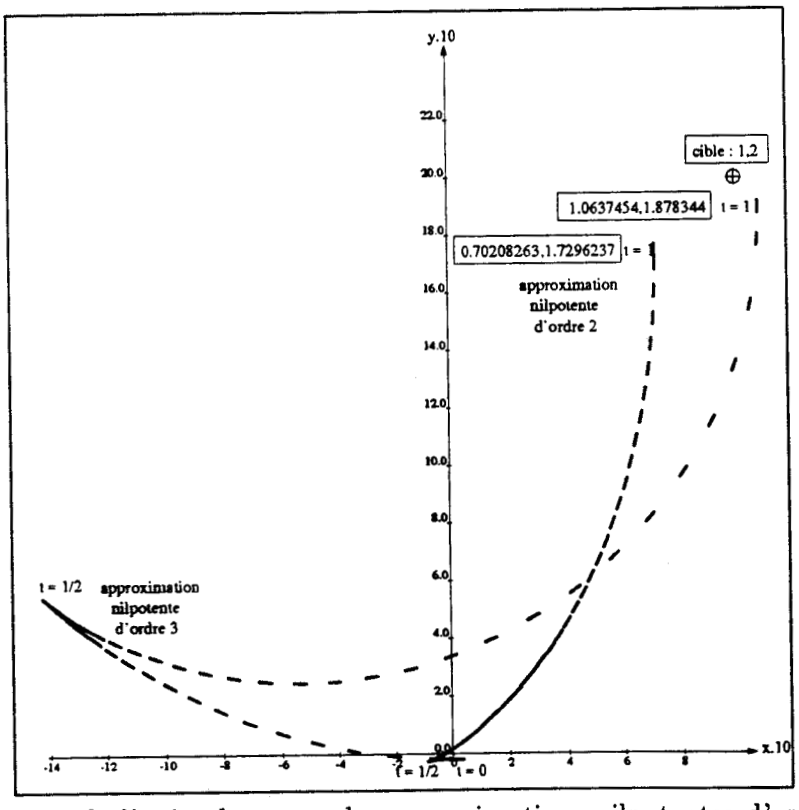


Figure 2.8 : trajectoire de l'unicycle. pour des approximations nilpotentes d'ordres 2 et 3. Entrées polynomiales. Arrivée en (1,2)

3

Le simulateur

3.1 Introduction

Notre but est, dans un premier temps, de développer un outil de représentation du comportement entrée/sortie des systèmes dynamiques représentés par leur série génératrice ([24]), puis d'étendre les possibilités de cet outil à la simulation du "motion planning" ([31, 64]) c'est à dire à la visualisation de trajectoires guidées par des familles de champs de vecteurs.

La première partie du simulateur porte donc sur les systèmes dynamiques. Nous avons choisi de représenter les entrées et sorties des systèmes par des courbes de Bézier, très utilisées en *C.A.O.*, elles offrent de nombreux avantages. Sur le plan graphique, cette représentation offre la possibilité de modifier la courbe de manière rapide (par déplacement des *points de contrôle*). Ce paramétrage donne également une idée claire de la vitesse le long de la courbe. Sur le plan des calculs, les algorithmes de tracé et de manipulation algébrique sont rapides et numériquement stables (voir [17]).

L'implantation d'un paquetage générique d'évaluation a permis en outre d'utiliser des entrées de différents types (courbes de Bézier, courbes paramétrées exprimées dans la base canonique, fonctions transcendantes de Scratchpad, fonctions polynomiales-exponentielles que nous avons implantées, ...) et de fournir l'expression de la sortie sous forme symbolique et sous forme graphique via le simulateur.

Nous allons dans ce chapitre détailler les outils graphiques que nous avons implantés en Scratchpad. Nous verrons ensuite le fonctionnement du simulateur, les limites et les avantages qu'il présente, et enfin nous comparerons, sur un exemple, ses résultats avec ceux du calcul numérique dont nous avons implantés quelques algorithmes.

3.2 Les courbes de Bézier

3.2.1 Introduction

Les courbes de Bézier ont été introduites dans les années 60 pour approximer uniformément les courbes paramétrées sur un intervalle fermé $[0,1]$ (voir [3, 12]). Elles sont un élément important de la simulation. Ce mode de représentation graphique a été choisi car il offre à l'utilisateur une grande souplesse de manipulation. En effet, lorsqu'on exprime une courbe par ses équations dans la base canonique, on n'apporte aucune information sur sa forme

générale. Par contre, les points de contrôle ont une signification géométrique : le point P de la courbe correspondant au temps $t \in [0,1]$ est le barycentre des points de contrôle P_k affectés de poids positifs : le k^{ieme} polynôme de Bernstein au temps t. Cette information supplémentaire permet *d'anticiper* sur l'allure de la courbe : on modifie sa forme en "tirant" sur ses points de contrôle alors que sur une expression dans la base canonique, il faut recalculer l'ensemble des coefficients.

Cet avantage, à priori anodin, a des *conséquences importantes au niveau de la simulation* : Il entraine la *naissance d'une nouvelle démarche* pour l'étude des systèmes dynamiques. Une "méthode classique" consiste à donner un système d'équations en entrée pour obtenir un système d'équations en sortie que l'on prendra éventuellement la peine de tracer. Notre nouvelle méthode permet, à partir d'un système d'équations décrivant une entrée, d'observer le comportement du système pour une famille de courbes proches au niveau de leur forme (alors que leurs expressions en base canonique peuvent être bien différentes). On va déformer la courbe sans se soucier de la nouvelle expression mathématique qui en résulte. La modification s'effectuant de manière *simple et rapide*, on passe d'une *observation statique* (une entrée pour une sortie) à une *observation dynamique* : l'évolution de la sortie d'un système en fonction de l'évolution de ses entrées.

Cet avantage, capital au niveau de l'interface homme-machine, n'est cependant pas le seul : des études effectuées par Farouki et Rajan sur les polynômes de Bernstein ([17]), qui sont l'outil indispensable à la construction des courbes de Bézier, montrent que les algorithmes appliqués aux polynômes de Bernstein divergent beaucoup moins vite que ceux appliqués aux polynômes exprimés dans la base canonique. Et même si Scratchpad permet le calcul exact, le graphisme nécessite toujours l'emploi de nombres flottants et entraine donc une propagation d'erreur. Enfin, l'utilisation des courbes de Bézier permet l'application d'algorithmes récursifs, décrits dans ce chapitre, qui sont particulièrement efficaces en Scratchpad.

Les concepteurs de Scratchpad se sont aperçus très tard de la portée, commerciale et scientifique, d'une interface graphique attrayante. Ces outils, proches de la C.A.O., n'avaient donc aucune raison de faire partie de la bibliothèque de base. Ceci nous a naturellement amené à développer l'ensemble des domaines et paquetages nécessaires à la mise en place de cette méthode de description graphique.

3.2.2 La base de Bernstein

Les courbes de Bézier sont des courbes paramétrées exprimées dans la base de Bernstein.

Definition 3.2.1 *Soit $\mathbb{P}_n$ l'espace des polynômes de degré n.*
Soit

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i \in \{0, 1, \ldots, n\}$$

avec

$$\binom{n}{i} = \frac{n!}{(n-i)!i!}$$

alors l'ensemble des $B_i^n(t)$ $\quad i \in \{0, 1, \ldots, n\}$, est une base de $\mathbb{P}_n$ appelée base de Bernstein (de degré n).

Exemple 3.2.1 *La base de Bernstein de degré 3 est formée des polynômes suivants :*

$$\begin{aligned} B_0^3(t) &= (1-t)^3 \\ B_1^3(t) &= 3(1-t)^2 t \\ B_2^3(t) &= 3(1-t)t^2 \\ B_3^3(t) &= t^3 \end{aligned}$$

Dans le but d'obtenir des algorithmes de tracé stables et efficaces, nous considérons l'expression d'un polynôme dans la base de Bernstein pour $t \in [0,1]$.
Lorsque nous voulons exprimer un polynôme pour un intervalle de valeurs de t différent, nous effectuons un changement de variable affine (voir les exemples en page 85) pour nous ramener à $t \in [0,1]$.

3.2.2.1 Quelques propriétés

1. positivité : $B_i^n(t) \geq 0$
2. symétrie : $B_i^n(t) = B_{n-i}^n(1-t)$
3. $B_i^n(t)$ atteint son maximum dans $[0,1]$ pour $t = \frac{i}{n}$
4. partition de l'unité : $\sum_{i=0}^{n} B_i^n(t) = 1$
5. relation de récurrence :
   $B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$
   $B_0^n(t) = (1-t)B_0^{n-1}(t)$
   $B_n^n(t) = tB_{i-1}^{n-1}(t)$
6. notons $\delta B_i^n(t)$ la dérivée première de $B_i^n(t)$, alors :
   $\delta B_i^n(t) = n(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)) \quad i \in [1..n]$

3.2.2.2 Opérations de base

3.2.2.2.1 Passage de la base canonique vers la base de Bernstein

Soit

$$P_n(t) = \sum_{i=0}^{n} a_i t^i$$

un polynôme exprimé dans la base canonique, alors son expression dans la base de Bernstein sera :

$$P_n(t) = \sum_{i=0}^{n} B_i^n(t) b_i$$

avec

$$b_k = \frac{k!}{n!} \sum_{i=0}^{k} \frac{(n-i)!}{(k-i)!} a_i$$

Cette formule est cependant très lourde et peu performante. Pour effectuer les changements de base, nous utilisons un algorithme beaucoup plus rapide, dont on peut trouver une description détaillée dans [18] et dont l'implantation en Scratchpad est donnée en annexe A (page A.1).

3.2.2.2.2 Expression dans une base de degré supérieur

Tout polynôme de degré n peut être exprimé dans une base de Bernstein de degré $m, (m \geq n)$ en utilisant la formule (voir [18]) :

$$\begin{aligned} P_n(t) &= \sum_{i=0}^{n} a_i B_i^n(t) \\ &= \sum_{i=0}^{n+1} b_i B_i^{n+1}(t) \end{aligned}$$

avec

$$\begin{aligned} b_0 &= a_0 \\ b_i &= \frac{i.a_{i-1} + (n-i+1)a_i}{n+1}, \quad i \in [1..n] \\ b_{n+1} &= a_n \end{aligned}$$

3.2.2.2.3 Addition

L'addition de deux polynômes P_1 et P_2, définis dans la même base de Bernstein de degré n est un polynôme défini dans la base de Bernstein de degré n :

Soit

$$P_1(t) = \sum_{i=0}^{n} a_i B_i^n(t)$$

et

$$P_2(t) = \sum_{i=0}^{n} b_i B_i^n(t)$$

alors

$$P_1(t) + P_2(t) = \sum_{i=0}^{n} c_i B_i^n(t)$$

avec

$$c_k = a_k + b_k$$

Ainsi, la première opération à effectuer avant de faire la somme est d'exprimer les deux polynômes dans la même base.

3.2.2.2.4 multiplication

La multiplication de deux polynômes P_1 et P_2, respectivement définis dans des bases de Bernstein de degré n et m donne un polynôme défini dans la base de degré $n + m$.

Soient

$$P_1(t) = \sum_{i=0}^{n} a_i B_i^n(t)$$

et

$$P_2(t) = \sum_{i=0}^{m} b_i B_i^m(t)$$

alors

$$P_1(t) * P_2(t) = \sum_{i=0}^{n+m} c_i B_i^{n+m}(t)$$

avec

$$c_k = \sum_{j=max(0,k-n)}^{min(k,m)} \frac{\binom{n}{j}\binom{m}{k-j}}{\binom{m+n}{k}} \cdot a_j \cdot b_{k-j}$$

3.2.2.2.5 Intégration

L'intégration d'un polynôme défini dans la base de Bernstein de degré n est donnée par la formule suivante (voir [16]) :
Soit

$$P(t) = \sum_{i=0}^{n} a_i B_i^n(t)$$

alors

$$\int_0^t P(\tau)d\tau = \sum_{i=0}^{n+1} I_i^{n+1} B_i^{n+1}(t)$$

avec :

$$\begin{cases} I_i^{n+1} = \dfrac{1}{n+1} \displaystyle\sum_{k=0}^{i-1} a_k \quad i = 1, 2, \ldots, n \\ I_0^{n+1} = 0. \end{cases}$$

Plus simplement, nous effectuons l'intégration du polynôme P par la formule itérative suivante (voir [6]) :

$$\int_0^t P(\tau)d\tau = \sum_{i=0}^{n+1} \sigma a_i B_i^{n+1}(t)$$

avec

$$\begin{cases} \sigma a_0 = 0 \\ \sigma a_k = \dfrac{1}{n+1} a_{k-1} + \sigma a_{k-1}, \ k \in [1, .., n+1]. \end{cases}$$

3.2.2.3 Implantation

Les polynômes de Bernstein sont implantés dans le domaine *UnivariateBernsteinPolynomial*. La programmation générique permet l'emploi de ces polynômes à coefficients dans n'importe quel anneau.

```
)abb domain UBP UnivariateBernsteinPolynomial
UnivariateBernsteinPolynomial(t:Expression,R:Ring): pub == priv where

  NNI          ==> NonNegativeInteger
  I            ==> Integer
  E            ==> Expression
  L            ==> List
  C            ==> IntegerCombinatoricFunctions
  UP           ==> UnivariatePoly(t,R)

  pub == Join(UnivPolyCat R,RetractableTo R,DifferentialRing) with

       bdegree      :    $          -> NNI
       b            :    (NNI,NNI)  -> $
       listCoef     :    $          -> L R
       if R has Field then
         elevate    :   ($,NNI)     -> $
         ajust      :   ($,NNI)     -> $
```

```
        integrate  :    $          -> $
        listeP     :  ($,I)       -> L $
      eval         :    ($,R)      -> R
      listsub      :    ($,R)      -> L L R
      bpoly        :    L R        -> $
      coerce       :    UP         -> $
      coerce       :    $          -> UP
      bform        :   (UP,R,R)    -> $
      minform      :    $          -> $

  priv == add

-- REPRESENTATION EN MEMOIRE

   Rep := L R
        .
        .
        .
```

On trouve sur la figure 3.1 le graphe d'héritage du domaine des polynômes exprimés en base de Bernstein. Ces héritages permettent de garder une compatibilité avec les autres domaines de polynômes (*UnivariatePolyCat*), de posséder l'ensemble des primitives de recherche des coefficients (*RetractableTo*) et de calcul des dérivées (*DifferentialRing*).

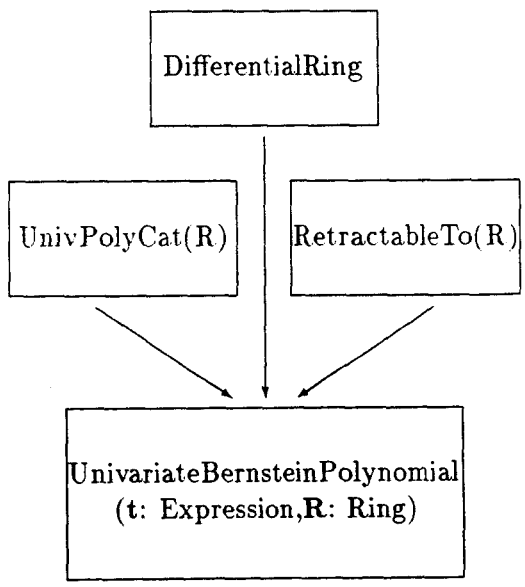


Figure 3.1 : Graphe d'héritage des Polynômes en base de Bernstein

3.2.2.3.1 Les opérateurs spécifiques au domaine *UnivariateBernsteinPolynomial*

1. Création d'un polynôme exprimé dans la base de Bernstein.

- En convertissant un polynôme exprimé dans la base canonique.

```
(1) ->p1: UP(t,RationalNumber):=t**2 - 1/2 * t + 5
```

$$(1) \qquad t^2 - \left(\frac{1}{2}\right) t + 5$$

```
(2) ->b1:=p1:: UnivariateBernsteinPolynomial(t,RationalNumber)
```

$$(2) \qquad 5\, B_0^2(t) + \left(\frac{19}{4}\right) B_1^2(t) + \left(\frac{11}{2}\right) B_2^2(t)$$

- En donnant la liste de ses coefficients.

```
(3) ->b2: UnivariateBernsteinPolynomial(t,RationalNumber):=
                                              bpoly [1/2,5,6,2/3]
```

$$(3) \qquad \left(\frac{1}{2}\right) B_0^3(t) + 5\, B_1^3(t) + 6\, B_2^3(t) + \left(\frac{2}{3}\right) B_3^3(t)$$

- En fournissant un polynôme en base canonique et les bornes minimum et maximum d'expression du polynôme en base de Bernstein.

```
(4) ->b3:=
      bform(p1,-2,4)$UnivariateBernsteinPolynomial(t,RationalNumber)
```

$$(4) \qquad 10\, B_0^2(t) - \left(\frac{7}{2}\right) B_1^2(t) + 19\, B_2^2(t)$$

- En combinant des polynômes de Bernstein fournis par la commande `b(i,n)` qui donne le i^{eme} polynôme de la base de degré n

```
(5) ->b4:= b(1,3) + 2*b(2,4)
```

$$(5) \qquad \left(\frac{3}{4}\right) B_1^4(t) + \left(\frac{5}{2}\right) B_2^4(t)$$

2. Expression d'un polynôme dans une base de degré supérieur.

```
(6) ->b17:= ajust(b1,7)
```

$$(6) \qquad \begin{array}{l} 5\, B_0^7(t) + \left(\frac{69}{14}\right) B_1^7(t) + \left(\frac{103}{21}\right) B_2^7(t) + \left(\frac{69}{14}\right) B_3^7(t) \\ +5\, B_4^7(t) + \left(\frac{215}{42}\right) B_5^7(t) + \left(\frac{37}{7}\right) B_6^7(t) + \left(\frac{11}{2}\right) B_7^7(t) \end{array}$$

3. Degré de la base dans laquelle est exprimé le polynôme.

```
(7) ->bdegree b17
```

(7) 7

4. Degré réel du polynôme.

```
(8) ->degree b17
```

(8) 2

3.2.2.4 Exemples

instanciation de deux polynômes p1 et p2

```
(2) ->p1 := t**3
```

(2) t^3

```
                                    Type:  Polynomial[t]RationalNumber
                                     Time:  .2 (IN) + .1 (OT) = .3 sec
```

```
(3) ->p2 := (1/2 - t)*(t+1)
```

(3) $$-t^2 - \left(\frac{1}{2}\right) t + \frac{1}{2}$$

```
                                    Type:  Polynomial[t]RationalNumber
                                         Time:  1.333 (IN) = 1.333 sec
```

déclaration de deux variables b1 et b2 comme des polynômes en base de Bernstein, leurs valeurs respectives étant celles des polynômes p1 et p2

```
 (4) ->b1: UnivariateBernsteinPolynomial(t,RationalNumber) := p1
```

(4) $B_3^3(t)$

```
                  Type:  UnivariateBernsteinPolynomial(t,RationalNumber)
                                                  Time:  .2 (IN) = .2 sec
```

```
(5) ->b2: UnivariateBernsteinPolynomial(t,RationalNumber) := p2
```

(5) $$\left(\frac{1}{2}\right) B_0^2(t) + \left(\frac{1}{4}\right) B_1^2(t) - B_2^2(t)$$

```
                    Type: UnivariateBernsteinPolynomial(t,RationalNumber)
                                                  Time:  .1 (IN) = .1 sec
```

intégration de b2 dans la base de Bernstein

```
(6) ->integrate b2
```

$$(6) \qquad \left(\frac{1}{6}\right) B_1^3(t) + \left(\frac{1}{4}\right) B_2^3(t) - \left(\frac{1}{12}\right) B_3^3(t)$$

```
                    Type: UnivariateBernsteinPolynomial(t,RationalNumber)
                                 Time:  .1 (IN) + .6 (OT) = .7 sec
```

multiplication de b1 et b2

```
(7) ->b1 * b2
```

$$(7) \qquad \left(\frac{1}{20}\right) B_3^5(t) + \left(\frac{1}{10}\right) B_4^5(t) - B_5^5(t)$$

```
                    Type: UnivariateBernsteinPolynomial(t,RationalNumber)
                              Time:  .233 (IN) + .1 (EV) = .333 sec
```

dérivation de b1 dans la base de Bernstein

```
(8) ->deriv b1
```

$$(8) \qquad 3\, B_2^2(t)$$

```
                    Type: UnivariateBernsteinPolynomial(t,RationalNumber)
                                                  Time:  .6 (OT) = .6 sec
```

vérification: expression de b2 dans la base de degré 5.
La soustraction de b25 et de b2 donne 0

```
(9) ->b25:=ajust(b2,5)
```

$$(9) \qquad \left(\frac{1}{2}\right) B_0^5(t) + \left(\frac{2}{5}\right) B_1^5(t) + \left(\frac{1}{5}\right) B_2^5(t) - \left(\frac{1}{10}\right) B_3^5(t) - \left(\frac{1}{2}\right) B_4^5(t) - B_5^5(t)$$

```
                    Type: UnivariateBernsteinPolynomial(t,RationalNumber)
                                 Time:  .1 (IN) + .1 (OT) = .2 sec
```

```
(10) ->b2-b25
```

$$(10) \qquad 0$$

```
Type:  UnivariateBernsteinPolynomial(t,RationalNumber)
                                    Time:  .1 (EV) = .1 sec
```

3.2.3 Les courbes de Bézier polynomiales

3.2.3.1 Principe et propriétés

Nous nous intéressons aux courbes polynomiales données par p polynômes en une variable. Il faut simplement exprimer ces polynômes dans la même base de Bernstein : Celle du plus haut degré. Nous pouvons alors exprimer la courbe sous forme d'un seul polynôme dans $\mathbb{R}^p[t]$.

$$B_n(\mathcal{Q},t) = \sum_{i=0}^{n} B_i^n(t) Q_i \qquad Q_i \in \mathbb{R}^p$$

$B_n(\mathcal{Q},t)$ sera appelée la courbe *associée* au polygone de Bézier

$$\mathcal{Q} = \{Q_0, Q_1, ..., Q_n\}$$

Les points Q_k sont appelés les *points de contrôle* de la courbe.

3.2.3.1.1 Quelques propriétés

1. $B_n(\mathcal{Q},0) = Q_0$
2. $B_n(\mathcal{Q},1) = Q_n$
3. Chaque point de la courbe est un barycentre des Q_i avec des poids positifs : la courbe $B_n(\mathcal{Q},t)$ est dans l'enveloppe convexe de $\mathcal{Q}$ lorsque t varie entre 0 et 1.

3.2.3.2 Algorithme de De Casteljau

Cet algorithme permet le calcul de la valeur du polynôme $P(t)$ au temps t_0 en utilisant son polygone associé $\mathcal{Q}$ ([12]).

DeCasteljau$(\mathcal{Q}, t_0)$

pour $i = 0$ **à** n **faire** $Q_i^{(0)}(t_0) = Q_i$
pour $j = 1$ **à** n **faire**
  pour $i = 0$ **à** $n - j$ **faire**
    $Q_i^{(j)}(t_0) = (1 - t_0) Q_i^{(j-1)}(t_0) + t_0 Q_{i+1}^{(j-1)}(t_0)$

resultat : $P(t_0) = Q_0^{(n)}(t_0)$

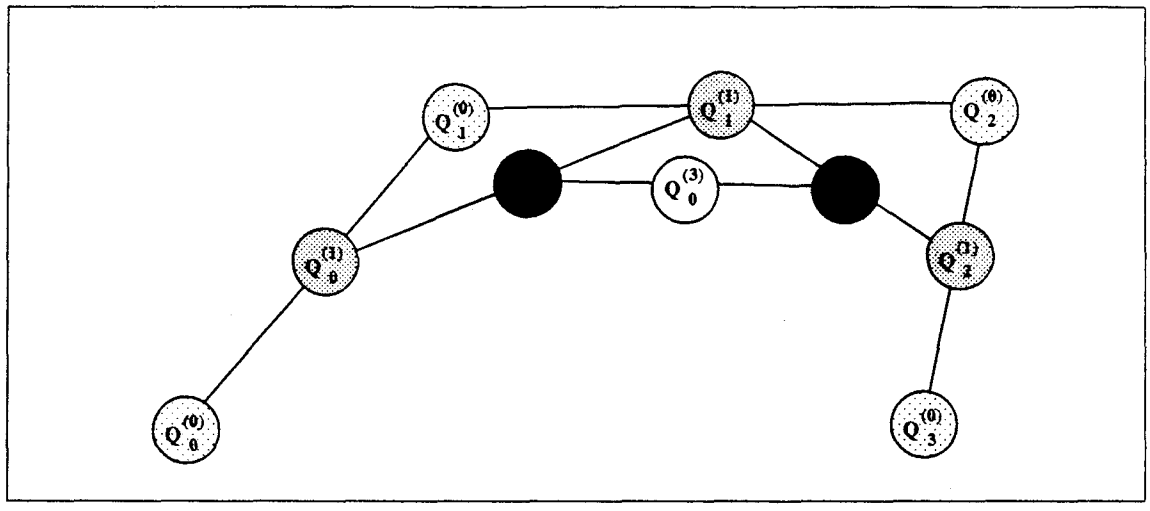


Figure 3.2 : Algorithme de De Casteljau

La figure 3.2 illustre le fonctionnement de cet algorithme.

Le polygone initial est $\mathcal{Q} = \{Q_0^{(0)}, Q_1^{(0)}, Q_2^{(0)}, Q_3^{(0)}\}$

3.2.3.3 Algorithme de subdivision

C'est l'algorithme effectivement utilisé pour le tracé des courbes ([18, 19]).
Soit la courbe $B_n(t)$ et son polygone associé

$$\mathcal{Q} = \{Q_0, ..., Q_n\}$$

Nous calculons sa valeur au temps $t_0 = \frac{1}{2}$ par l'algorithme de De Casteljau. J.M. Lane et R.F. Riesenfeld ([48]) ont prouvé que les deux courbes $B_1(t)$ et $B_2(t)$, respectivement associées aux polygones

$$\mathcal{Q}_1 = \{Q_0^{(0)}, Q_0^{(1)}, ..., Q_0^{(n)}\}$$

et

$$\mathcal{Q}_2 = \{Q_0^{(n)}, Q_1^{(n-1)}, ..., Q_n^{(0)}\}$$

décrivent les deux parties de la courbe définies sur $t \in [0, \frac{1}{2}]$ et $t \in [\frac{1}{2}, 1]$ ([48]). Sachant que les calculs intermédiaires fournis par l'algorihme de De Casteljau donnent les points $Q_i^{(j)}$.

Remarquons que ces points se calculent de manière très simple et rapide lorsque $t = \frac{1}{2}$ par :

$$Q_i^{(j)} = \frac{Q_i^{(j-1)} + Q_{i+1}^{(j-1)}}{2}$$

Il ne reste plus qu'à itérer le processus sur $\mathcal{Q}_1$ et $\mathcal{Q}_2$ et ainsi de suite jusqu'à obtenir une suite de polygones se rapprochant (très rapidement) de la courbe.

Cet algorithme est décrit par la figure 3.3. Les résultats intermédiaires fournis par l'algorithme de De Casteljau donnent les deux nouveaux polygones $\{P_0, P_1, P_2, P_3\}$ et $\{Q_0, Q_1, Q_2, Q_3\}$ dont les courbes de Bézier associées restrictions de la courbe initiale sur $[0, \frac{1}{2}]$ et $[\frac{1}{2}, 1]$.

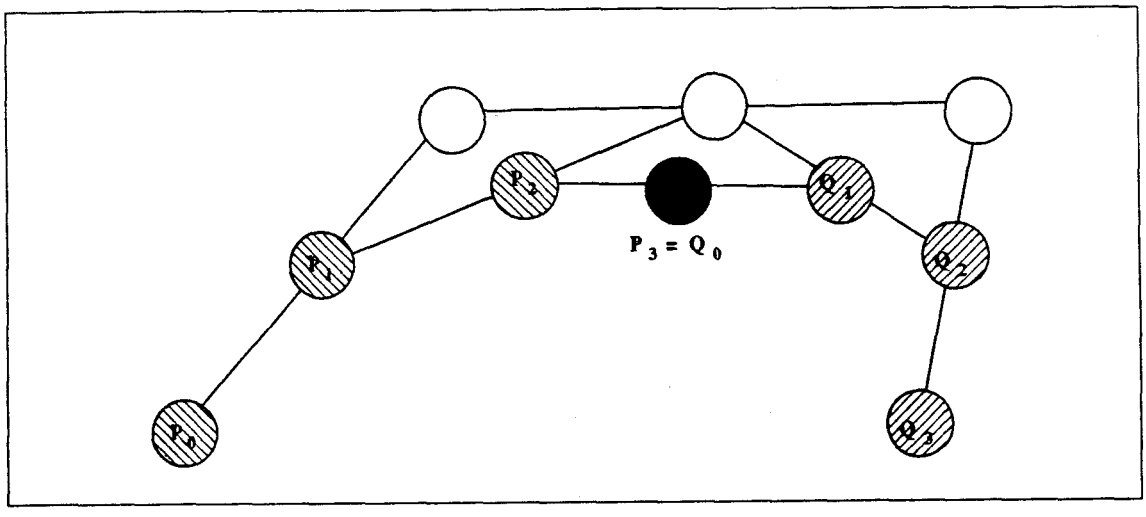


Figure 3.3 : Algorithme de subdivision

subdivision($\mathcal{Q}$)

DeCasteljau($\mathcal{Q}$, $\frac{1}{2}$)
$\mathcal{Q}_1 = \{Q_0^{(0)}, Q_0^{(1)}, \ldots, Q_0^{(n)}\}$
$\mathcal{Q}_2 = \{Q_0^{(n)}, Q_1^{(n-1)}, \ldots, Q_n^{(0)}\}$
si (arrêt = faux)
alors
  $\mathcal{R}$:= [subdivision($\mathcal{Q}_1$), subdivision($\mathcal{Q}_2$)]
fsi

résultat : $\mathcal{R}$

Le test "(arrêt = faux)" peut être de diverses natures : On peut fixer à l'avance le nombre de subdivisions (en général, 3 à 5 suffisent) ou, lorsqu'on connait la résolution de l'écran, mesurer la distance entre deux points successifs, et décider de l'arrêt dès lors qu'elle est inférieure à une valeur v fixée (lorsque v est suffisamment petite, cela signifie que les points seront confondus à l'écran). Nous avons utilisé ces deux méthodes, la première pour les sorties sur imprimante, la seconde pour les sorties à l'écran.

3.2.3.4 Implantation

Le calcul du tracé est implanté dans quatre fonctions :

1. Première boucle de l'algorithme de De Casteljau. Le nom de la fonction est dû à la représentation graphique qu'elle donne (voir la figure 3.2).

```
internPolygon(l) ==
        internPolygon(l) ==
                [[(e1.xCoord+e2.xCoord)/2,(e1.yCoord+e2.yCoord)/2]
                                        for e1 in l for e2 in rest l]
```

l est un polygone, c'est à dire une liste de points. Le résultat est un polygone possédant un point de moins.

2. Subdivision : On calcul l'ensemble des polygones internes. Le dernier n'est constitué que d'un seul point.

```
subdiv(l) ==
        #l = 1 => [l]
        cons(l,subdiv(internPolygon(l)))
```

Le résultat est une liste de listes de points : la liste des polygones internes (voir figure 3.2).

3. Création des deux nouveaux polygones.

```
resultPolygon(l) ==
        lp := subdiv(l)
        [[p.first for p in lp],[p.last for p in reverse lp]]
```

Le resultat est une liste formée de deux listes de points : les deux nouveux polygones : $[P_0, P_1, P_2, P_3]$ et $[Q_0, Q_1, Q_2, Q_3]$ (voir figure 3.3).

4. calcul recursif de l'ensemble des points de la courbe.

```
bcurve(l,n) ==
        n = 0 => [l]
        p := resultPolygon(l)
        append(bcurve(p.first,n-1),bcurve(p.last,n-1))
```

l est le polygone initial. n est le nombre d'appels récursifs. On voit ici clairement la complexité exponentielle de l'algorithme en fonction du nombre de subdivisions : `bcurve(l,n)` fait deux fois appel à `bcurve(l,n-1)`.

L'implantation des courbes de Bézier se fait dans un domaine :
BezierCurve(t: Expression,R: Ring) et un paquetage : *BezierDraw2D()*.

Le premier permet de créer des variables de type *courbe de Bézier* par conversion de listes de polynômes exprimés en base de Bernstein ou en base canonique.

Le second permet l'affichage effectif des courbes par application des primitives décrites plus haut. Il n'est pas générique. L'utilisation de nombres flottants est ici obligatoire pour deux raisons : un soucis de rapidité de calcul et l'obligation de rester compatible avec les primitives graphique de Scratchpad déjà en place.

3.2.3.5 Exemples

Création de deux polynômes exprimés en base de Bernstein.

```
(1) ->b1: UnivariateBernsteinPolynomial(t,RationalNumber) := bpoly [1,1/2,3,6]
```

$$(1) \qquad B_0^3(t) + \left(\frac{1}{2}\right) B_1^3(t) + 3\ B_2^3(t) + 6\ B_3^3(t)$$

```
(2) ->b2: UnivariateBernsteinPolynomial(t,RationalNumber) := bpoly [1/3,4/5,8]
```

$$(2) \qquad \left(\frac{1}{3}\right) B_0^2(t) + \left(\frac{4}{5}\right) B_1^2(t) + 8\ B_2^2(t)$$

Conversion en objet *courbe de Bezier*. b2 est d'abord converti dans la base de degré 3. On exprime ensuite la courbe comme un polynôme à coefficients dans $\mathbb{R}^2$.

```
(3) ->[b1,b2] ::  BC(t,RationalNumber)
```

$$(3) \qquad \begin{bmatrix} 1 \\ \frac{1}{3} \end{bmatrix} B_0^3(t) + \begin{bmatrix} \frac{1}{2} \\ \frac{29}{45} \end{bmatrix} B_1^3(t) + \begin{bmatrix} 3 \\ \frac{16}{5} \end{bmatrix} B_2^3(t) + \begin{bmatrix} 6 \\ 8 \end{bmatrix} B_3^3(t)$$

3.3 Les courbes polynomiales exponentielles

Nous désirons tracer des courbes dont l'expression est de la forme :

$$\sum_{i=0}^{n} P_i(t) e^{\alpha_i t}$$

Cette expression peut provenir de l'évaluation d'une fraction rationnelle dont l'entrée est polynomiale, c'est à dire exprimée sous forme d'une courbe de Bézier.

Pour cela, nous développons le domaine des polynômes exponentiels dont l'implantation générique permet d'utiliser, en paramètre n'importe quel type de polynôme. Nous avons déjà présenté une utilisation des polynômes exponentiels en 2.2.3.2 page 61. Nous nous sommes alors contentés d'en donner des exemples d'évaluation pour des polynômes exprimés en base canonique ou pour des nombres complexes. Nous donnons ici une partie de cession Scratchpad permettant la création de polynômes exponentiels en base de Bernstein ainsi qu'un exemple de conversion via le paquetage *PolExpoCoercionPackage*.

- Déclaration des domaines

```
(1) ->Ub:= UBP(t,RN)

   (1)  UnivariateBernsteinPolynomial(t,RationalNumber)
                                                        Type: Domain
                                              Time: .033 (IN) = .033 sec

(2) ->Px := POLEXP(t,RN,Ub)

   (2)
  PolynomialExponential(t,RationalNumber,UnivariateBernsteinPolynomial
  (t,RationalNumber))
                                                        Type: Domain
                                                         Time: 0 sec

(3) ->pec := PECPKG(t,RN)

   (3)  PolExpoCoercionPackage(t,RationalNumber)
                                                        Type: Domain
                                              Time: .067 (IN) = .067 sec
```

- Création de deux polynômes-exponentiels, le symbôle %e généré par Scratchpad représente l'exponentielle.

```
(4) ->px1 :=  exp(-2)$Px
```

$$(4) \qquad \%e^{-2\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,
                      UnivariateBernsteinPolynomial(t,RationalNumber))
                                  Time: .133 (IN) + .067 (EV) = .2 sec

(5) ->px2 := exp(3)$Px *b(4,5)$Ub
```

$$(5) \qquad B_4^5(t)\,\%e^{3\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,
                      UnivariateBernsteinPolynomial(t,RationalNumber))
                                              Time: .867 (IN) = .867 sec
```

- Création d'un troisième par addition

```
(6) ->px3 := px1 + px2
```

$$(6) \qquad B_4^5(t)\,\%e^{3\,t} + \%e^{-2\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,
                    UnivariateBernsteinPolynomial(t,RationalNumber))
                                    Time: .067 (IN) + .1 (OT) = .167 sec
```

- Intégration, utilisée lors de l'évaluation

```
(7) ->ipx3 :=integrate px3
```

(7)
$$\left(\frac{320}{243}\,B_0^5(t)+\frac{128}{243}\,B_1^5(t)+\frac{80}{243}\,B_2^5(t)+\frac{32}{243}\,B_3^5(t)+\frac{56}{243}\,B_4^5(t)+\frac{35}{243}\,B_5^5(t)\right)\,\%e^{3\,t}-\frac{397}{486}-\frac{1}{2}\,\%e^{-2\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,
                    UnivariateBernsteinPolynomial(t,RationalNumber))
                  Time: .067 (IN) + .333 (EV) + .133 (OT) = .533 sec
```

- Conversion des polynômes en base canonique

```
(8) ->px4 := coerce(px3)$pec
```

(8)
$$\left(-5\,t^5+5\,t^4\right)\,\%e^{3\,t}+\%e^{-2\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,
                                UnivariatePoly(t,RationalNumber))
                                    Time: .9 (IN) = .9 sec
```

- Nouvelle intégration d'un polynôme exponentiel

```
(9) ->ipx4 := integrate px4
```

(9)
$$\left(-\left(\frac{5}{3}\right)t^5+\left(\frac{40}{9}\right)t^4-\left(\frac{160}{27}\right)t^3+\left(\frac{160}{27}\right)t^2-\left(\frac{320}{81}\right)t+\frac{320}{243}\right)\,\%e^{3\,t}-\frac{397}{486}-\left(\frac{1}{2}\right)\,\%e^{-2\,t}$$

```
Type: PolynomialExponential(t,RationalNumber,
                                UnivariatePoly(t,RationalNumber))
                  Time: .067 (IN) + .1 (EV) = .167 sec
```

Une première solution est d'exprimer le polynôme de Bézier en base canonique avant d'évaluer. On perd alors tout l'intérêt des polynômes de Bernstein et on augmente le nombre d'opérations à effectuer.

Une deuxième solution, que nous avons choisie, consiste à conserver l'expression dans la base de Bernstein des polynômes en entrée comme en sortie. Chaque polynôme P_i est donc exprimé sous la forme :

$$\sum_{j=0}^{m_i} b_j B_j^{m_i}(t)$$

Nous effectuons alors le tracé de la manière suivante :

1. Nous exprimons tous les P_i dans une base de même degré : nous recherchons pour cela le polynôme de plus haut degré.

   $$m = max(deg(P_i)) \quad pour\ i \in [0, 1, \ldots, n]$$

   Nous obtenons alors une nouvelle expression des P_i :

   $$\sum_{j=0}^{m} b_j B_j^{m}(t)$$

2. Nous effectuons un calcul de subdivision (décrit en 3.2.3.3) sur chacun des polynômes P_i dont nous ne conservons que les points exacts. Les polynômes étant tous exprimés dans une base de même degré, un nombre égal de subdivision sur chacun des polynômes donne un même nombre de points calculés.

3. Le nombre de subdivisions effectuées nous donne le pas de discrétisation du temps, nous pouvons alors, en fonction de ce pas, calculer numériquement chacune des expressions e^{a_i}.

4. Nous multiplions les valeurs des polynômes par celles des exponentielles, nous sommons enfin les listes de points ainsi obtenues.

5. Nous traçons la liste de points fournie par ces calculs.

Cette méthode peut sembler compliquée, mais elle possède des avantages par rapport au calcul numérique des valeurs des polynômes et des exponentielles : pour calculer les valeurs des polynômes, nous utilisons l'algorithme de subdivision, nous nous limitons ainsi à des opérations d'addition et de division par 2. Une méthode de calcul numérique va faire intervenir un nombre important de multiplications, pour les calculs de puissance et ainsi entrainer un nombre plus important d'erreurs. Ces erreurs, multipliées par un coefficient exponentiel peuvent prendre des proportions importantes, que nous limitons ici.

D'autre part, nous ne calculons par cette technique qu'un petit nombre de points (par rapport aux méthode numériques que nous décrivons en 3.6), nous effectuons donc peu de calculs d'exponentielles.

Enfin, en conservant les polynômes de Bézier, nous évitons tout changement de base entre les affichages des entrées et des sortie, conservant ainsi une cohérence des calculs.

3.4 Les courbes rationnelles, forme (BR)

3.4.1 Présentation

Nous avons implanté les domaines et paquetages nécessaires au tracé des courbes Rationnelles mises sous forme B-Rationnelles (voir [19]) dans les buts d'élargir la classe de courbes que nous pouvons représenter et également de simuler le comportement à l'infini des courbes polynomiales.

Pour implanter les courbes rationnelles, nous nous basons sur les travaux de J.C. Fiorot et P. Jeannin qui introduisent la notion de *vecteurs de contrôle*. On trouve une présentation complète de cette technique dans leur livre ([19]). Cette notion permet de décrire les courbes par des *vecteurs massiques*.

Un vecteur massique est soit un *point pondéré*, c'est à dire la donnée d'un couple (coordonnées,masse), soit un vecteur de $\vec{\mathcal{E}}$ où :
$\mathcal{E}$ est un espace affine euclidien réel de dimension finie.
$\vec{\mathcal{E}}$ est l'espace vectoriel associé.
On note $\hat{\mathcal{E}} = (\mathcal{E} \times I\!R - \{0\}) \cup \vec{\mathcal{E}}$ l'ensemble des vecteurs massiques.
$\mathcal{E}$ muni des lois $\oplus$ et $*$ définies ci-dessous possède une structure d'espace vectoriel.
Soient (A, α) et (B, β) deux points pondérés, $\vec{u}$ et $\vec{v}$ deux vecteurs de $\vec{\mathcal{E}}$ et λ un élément de $I\!R$.

1. $(A,\alpha) \oplus (B,\beta) = (\dfrac{\alpha A + \beta B}{\alpha + \beta}, \alpha + \beta) \quad si\ \alpha + \beta \neq 0$
   $(A,\alpha) \oplus (B,\beta) = \alpha\vec{A} + \beta\vec{B} = \alpha\vec{BA} = \beta\vec{AB} \quad si\ \alpha + \beta = 0$
2. $(A,\alpha) \oplus \vec{u} = (A + (\frac{1}{\alpha})\vec{u}, \alpha) \quad si\ \alpha \neq 0$
   $(A,\alpha) \oplus \vec{u} = \vec{u} \quad si\ \alpha = 0$
3. $\vec{u} \oplus \vec{v} = \vec{u} + \vec{v}$
4. $\lambda * (A,\alpha) = (A, \lambda \cdot \alpha)$
5. $\lambda * \vec{u} = \lambda \cdot \vec{u}$

On décrit alors la courbe par la donnée d'un *polygone massique* qu'on détermine de la manière suivante :
Plaçons nous sur un plan affine $\mathcal{E}$ de repère cartésien $(O, \vec{i}, \vec{j})$
Considérons le courbe rationnelle donnée par :

$$BR(t) = \frac{A_1(t)}{D(t)}\vec{i} + \frac{A_2(t)}{D(t)}\vec{j}$$

où A_1, A_2 et D sont des polynômes de degré inférieur ou égal à n.

Considérons les *coordonnées homogènes* cartésiennes de la courbe :

$$(A_1(t),\ A_2(t),\ D(t))$$

exprimons ces polynômes dans la base de Bernstein :

$$A_k(t) = \sum_{i=0}^{n} \alpha_i^k B_i^n(t)\ ,$$

$$D(t) = \sum_{i=0}^{n} \beta_i B_i^n(t)$$

On déduit alors l'ensemble des vecteurs massiques $\theta_i \quad i \in \{0, 1, ..., n\}$ définis par :

$$\theta_i = \begin{cases} \alpha_i^1 \vec{i} + \alpha_i^2 \vec{j} & si \quad \beta_i = 0 \\ \left(\dfrac{\alpha_i^1 \vec{i} + \alpha_i^2 \vec{j}}{\beta_i}, \beta_i \right) & si \quad \beta_i \neq 0 \end{cases}$$

Une courbe rationnelle s'exprime alors sous la forme :

$$BR[\theta](t) = \frac{\sum\limits_{i \in I} \beta_i B_i^n(t) P_i}{\sum\limits_{i \in I} \beta_i B_i^n(t)} + \frac{\sum\limits_{i \in \bar{I}} \beta_i B_i^n(t) \vec{U}_i}{\sum\limits_{i \in I} \beta_i B_i^n(t)}$$

avec

$$I = \{\theta_i \mid \theta_i = (P_i, \beta_i) \quad avec \ \beta_i \neq 0\}$$
$$\bar{I} = \{\theta_i \mid \theta_i = \vec{U}_i\}$$

Cette forme est appelée *forme (BR)* de la courbe rationnelle.

Il y a identité entre les courbes (BR) et les courbes rationnelles.

Dans le cas où $\beta(t)$ est identiquement nul, la courbe n'est définie que par des vecteurs, elle est entièrement à l'infini.

Le calcul effectif des points de la courbe est réalisé par projection de l'algorithme de subdivision décrit en 3.2.3.3, appliqué à la courbe polynomiale définie par les polynômes A_1, A_2 et D. On décrit la courbe sur l'intervalle fermé $[0, 1]$. Si la courbe est exprimée sur un intervalle différent, fini ou infini, on effectue un changement de variable, encore appelé *paramétrisation* de la courbe.

3.4.2 Paramétrisation

On paramétrise les courbes (polynomiales et rationnelles) afin de toujours ramener l'intervalle d'étude à $t \in [0, 1]$. La paramétrisation fait intervenir de 1 à 3 paramètres réels. Pour plus de précisions sur la paramétrisation et son incidence sur le contrôle des courbes, voir [19] et [66].

1. Changement de paramètre affine.
   Nous l'avons déjà utilisé pour les courbes polynômiales (voir paragraphe 1), il permet d'exprimer la courbe sur un intervalle fermé quelconque.
   La transformation est la suivante :

$$\begin{array}{rccc} \Phi_1 : & [0,1] & \rightarrow & [\alpha, \beta] \\ & t & \mapsto & \alpha + t \cdot (\beta - \alpha) \end{array}$$

   avec $\alpha < \beta$

2. Changement de paramètre homographique.

$$\begin{array}{rccc} \Phi_2 : & [0,1] & \rightarrow & [0, +\infty] \\ & t & \mapsto & \dfrac{\alpha t}{1 - t} \end{array}$$

avec $\alpha > 0$

3. Changement de paramètre quadratique.

$$\begin{array}{rccl} \Phi_3 : & [0,1] & \longrightarrow & I\!R \\ & t & \longmapsto & \dfrac{\alpha B_0^2(t) + \beta B_1^2(t) + \gamma B_2^2(t)}{B_1^2(t)} \end{array}$$

avec $\alpha\gamma < 0$
Ce changement de paramètres double le nombre de points de contrôle, une moitié d'entre eux étant déduite simplement de l'autre (propriété de ρ-réciprocité, voir [20, 66]) .

3.4.3 Exemples

Création de vecteurs massiques.

```
(1) -> p1:VM(RationalNumber) := pp([1/2,8],-3)
```

(1) $$-3\begin{bmatrix} \frac{1}{2} \\ 8 \end{bmatrix}$$

```
(2) -> p2:VM(RationalNumber):= pp([0,4/3],8/9)
```

(2) $$\frac{8}{9}\begin{bmatrix} 0 \\ \frac{4}{3} \end{bmatrix}$$

```
(3) -> p3:VM(RationalNumber):= pp([4,3],3)
```

(3) $$3\begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

```
(4) -> v1:VM(RationalNumber):= vect [2/3,5/4]
```

(4) $$\begin{bmatrix} \frac{2}{3} \\ \frac{5}{4} \end{bmatrix}$$

```
(5) -> v2:VM(RationalNumber):= vect [1/2,5]
```

(5) $$\begin{bmatrix} \frac{1}{2} \\ 5 \end{bmatrix}$$

Les procédures d'affichage de Scratchpad ne permettent pas de représenter aisément les vecteurs. Sur les exemples ci-dessus, on remarque que les points pondérés sont précédés de leur masse tandis que les vecteurs n'en possèdent pas.

Addition de 2 points pondérés.


```
(6) ->p1+p2
```

$$(6) \qquad -\frac{19}{9}\begin{bmatrix} \frac{27}{38} \\ \\ \frac{616}{57} \end{bmatrix}$$

Addition d'un point pondéré et d'un vecteur.

```
(7) ->p1+v1
```

$$(7) \qquad -3\begin{bmatrix} -\frac{31}{18} \\ \\ -\frac{293}{12} \end{bmatrix}$$

Addition de 2 vecteurs.

```
(8) ->v1+v2
```

$$(8) \qquad \begin{bmatrix} \frac{7}{6} \\ \\ \frac{25}{4} \end{bmatrix}$$

Addition de 2 points pondérés de poids opposés. Le résultat est un vecteur, ce que l'on vérifie à l'aide de la primitive `v?` dont le résultat est un booléen.

```
(9) ->p3+p1
```

$$(9) \qquad \begin{bmatrix} \frac{21}{2} \\ \\ -15 \end{bmatrix}$$

```
(10) ->v? %
```

$$(10) \qquad \textit{true}$$

Multiplication d'un vecteur par un élément du corps de base.

```
(11) ->7/2 * v1
```

$$(11) \qquad \begin{bmatrix} \frac{7}{3} \\ \\ \frac{35}{8} \end{bmatrix}$$

Multiplication d'un point pondéré par un élément du corps de base.

```
(12) ->1/3 * p2
```

$$(12) \qquad \frac{8}{27}\begin{bmatrix}0\\ \\ \frac{4}{3}\end{bmatrix}$$

Création d'un objet "courbe rationnelle" à partir d'une liste de vecteurs massiques.

```
(13) ->[p1,v1,p2] :: RC(t,RationalNumber)
```

$$(13) \qquad \frac{\left(-3\begin{bmatrix}\frac{1}{2}\\ 8\end{bmatrix}\right) B_0^2(t) + \left(\frac{8}{9}\begin{bmatrix}0\\ \frac{4}{3}\end{bmatrix}\right) B_2^2(t)}{-3\ B_0^2(t) + \left(\frac{8}{9}\right) B_2^2(t)} + \frac{\begin{bmatrix}\frac{2}{3}\\ \frac{5}{4}\end{bmatrix} B_1^2(t)}{-3\ B_0^2(t) + \left(\frac{8}{9}\right) B_2^2(t)}$$

La sortie Scratchpad donne deux fractions dont les dénominateurs sont égaux : la première regroupe l'ensemble des points pondérés et la deuxième les vecteurs.

Création d'un objet courbe rationnelle (ici dans un espace de dimension 3) à partir d'une liste de fractions rationnelles.

```
(14) ->l: L QF UP(t,RationalNumber) := [1+t**2,(t+3*t**3)/(1-1/2*t**2),1/t]
```

$$(14) \qquad \left[t^2+1, \frac{-6\,t^3-2\,t}{t^2-2}, \frac{1}{t}\right]$$

```
(15) ->br(l)$BRC(t,RationalNumber)
```

$$(15) \qquad \frac{-\frac{2}{5}\begin{bmatrix}1\\0\\5\end{bmatrix} B_1^5(t) + -\frac{4}{5}\begin{bmatrix}1\\ \frac{1}{4}\\ \frac{19}{8}\end{bmatrix} B_2^5(t) + -\frac{11}{10}\begin{bmatrix}\frac{13}{11}\\ \frac{6}{11}\\ \frac{17}{11}\end{bmatrix} B_3^5(t) + -\frac{6}{5}\begin{bmatrix}\frac{5}{3}\\ 2\\ \frac{7}{6}\end{bmatrix} B_4^5 t + -1\begin{bmatrix}2\\8\\1\end{bmatrix} B_5^5(t)}{-\frac{2}{5}\ B_1^5(t) - \frac{4}{5}\ B_2^5(t) - \frac{11}{10}\ B_3^5(t) - \frac{6}{5}\ B_4^5(t) - B_5^5(t)}$$

$$+ \qquad \frac{\begin{bmatrix}0\\0\\-2\end{bmatrix} B_0^5(t)}{-\frac{2}{5}\ B_1^5(t) - \frac{4}{5}\ B_2^5(t) - \frac{11}{10}\ B_3^5(t) - \frac{6}{5}\ B_4^5(t) - B_5^5(t)}$$

On voit ici un avantage de Scratchpad : une seule et même implantation quelque soit la dimension de l'espace considéré. Si nous voulons tracer des courbes, il est néanmoins obligatoire de travailler en dimension 2. Nous en donnons ici un exemple avec le calcul de l'expression du *folium de Descartes* sur différents intervalles de temps et le tracé des courbes.
Sur cet exemple, les opérations (1) à (7) sont des instanciations de domaines et paquetage.

Définition de la courbe par deux fractions rationnelles

```
(8) -> q1:qf:= (3*t)/(1+t**3)
```

$$(8) \qquad \frac{3\,t}{t^3+1}$$

```
(9) -> q2:qf:= (3*t**2)/(1+t**3)
```

$$(9)\qquad \frac{3\,t^2}{t^3+1}$$

Définition des équations de la courbe sur $[0,1]$

```
(11) -> br1:brc:= br(1)$brc
```

$$(11)\qquad \frac{\left(1\begin{bmatrix}0\\0\end{bmatrix}\right)B_0^3(t)+\left(1\begin{bmatrix}1\\0\end{bmatrix}\right)B_1^3(t)+\left(1\begin{bmatrix}2\\1\end{bmatrix}\right)B_2^3(t)+\left(2\begin{bmatrix}\frac{3}{2}\\\frac{3}{2}\end{bmatrix}\right)B_3^3(t)}{B_0^3(t)+B_1^3(t)+B_2^3(t)+2\,B_3^3(t)}$$

```
(12) -> br1::L vm
```

$$(12)\qquad \left[\left(1\begin{bmatrix}0\\0\end{bmatrix}\right),\left(1\begin{bmatrix}1\\0\end{bmatrix}\right),\left(1\begin{bmatrix}2\\1\end{bmatrix}\right),\left(2\begin{bmatrix}\frac{3}{2}\\\frac{3}{2}\end{bmatrix}\right)\right]$$

Définition des équations de la courbe sur $[0,+\infty]$

```
(13) -> br2 := halfspace(br1)$param
```

$$(13)\qquad \frac{\left(\frac{1}{3}\begin{bmatrix}0\\0\end{bmatrix}\right)B_0^3(t)+\left(\frac{1}{3}\begin{bmatrix}0\\0\end{bmatrix}\right)B_3^3(t)}{\left(\frac{1}{3}\right)B_0^3(t)+\left(\frac{1}{3}\right)B_3^3(t)}+\frac{\begin{bmatrix}\frac{1}{3}\\0\end{bmatrix}B_1^3(t)+\begin{bmatrix}0\\\frac{1}{3}\end{bmatrix}B_2^3(t)}{\left(\frac{1}{3}\right)B_0^3(t)+\left(\frac{1}{3}\right)B_3^3(t)}$$

Les vecteurs massiques

```
(14) -> br2::L vm
```

$$(14)\qquad \left[\left(\frac{1}{3}\begin{bmatrix}0\\0\end{bmatrix}\right),\begin{bmatrix}\frac{1}{3}\\0\end{bmatrix},\begin{bmatrix}0\\\frac{1}{3}\end{bmatrix},\left(\frac{1}{3}\begin{bmatrix}0\\0\end{bmatrix}\right)\right]$$

Les fractions rationnelles

```
(15) ->br2::lqf
```

$$(15)\qquad \left[\frac{t^3-2\,t^2+t}{t^2-t+\frac{1}{3}},\frac{-t^3+t^2}{t^2-t+\frac{1}{3}}\right]$$

Définition des équations de la courbe sur $]-\infty,+\infty[$

```
(16) -> br3 := allspace(br1)$param
```

$$(16)\qquad \frac{\left(-1\begin{bmatrix}0\\0\end{bmatrix}\right)B_0^6(t)+\left(\frac{1}{5}\begin{bmatrix}-1\\0\end{bmatrix}\right)B_2^6(t)+\left(-\frac{1}{20}\begin{bmatrix}0\\-6\end{bmatrix}\right)B_3^6(t)+\left(-\frac{1}{5}\begin{bmatrix}-1\\0\end{bmatrix}\right)B_4^6(t)+\left(1\begin{bmatrix}0\\0\end{bmatrix}\right)B_6^6(t)}{-B_0^6(t)+\left(\frac{1}{5}\right)B_2^6(t)-\left(\frac{1}{20}\right)B_3^6(t)-\left(\frac{1}{5}\right)B_4^6(t)+B_6^6(t)}$$

$$+\qquad \frac{\begin{bmatrix}0\\-\frac{1}{2}\end{bmatrix}B_1^6(t)+\begin{bmatrix}0\\-\frac{1}{2}\end{bmatrix}B_5^6(t)}{-B_0^6(t)+\left(\frac{1}{5}\right)B_2^6(t)-\left(\frac{1}{20}\right)B_3^6(t)-\left(\frac{1}{5}\right)B_4^6(t)+B_6^6(t)}$$

Les vecteurs massiques

```
(17) -> br3::L vm
```

$$(17) \quad \left[\left(-1\begin{bmatrix}0\\0\end{bmatrix}\right), \begin{bmatrix}0\\-\frac{1}{2}\end{bmatrix}, \left(\frac{1}{5}\begin{bmatrix}-1\\0\end{bmatrix}\right), \left(-\frac{1}{20}\begin{bmatrix}0\\-6\end{bmatrix}\right), \left(-\frac{1}{5}\begin{bmatrix}-1\\0\end{bmatrix}\right), \begin{bmatrix}0\\-\frac{1}{2}\end{bmatrix}, \left(1\begin{bmatrix}0\\0\end{bmatrix}\right)\right]$$

Les fractions rationnelles

```
(18) ->br3::lqf
```

$$(18) \quad \left[\frac{6\,t^5 - 15\,t^4 + 12\,t^3 - 3\,t^2}{t^6 - 3\,t^5 + 3\,t^4 + 7\,t^3 - 12\,t^2 + 6\,t - 1}, \frac{12\,t^4 - 24\,t^3 + 15\,t^2 - 3\,t}{t^6 - 3\,t^5 + 3\,t^4 + 7\,t^3 - 12\,t^2 + 6\,t - 1}\right]$$

Le tracé de ces courbes est donné sur les figures 3.4, 3.5 et 3.6.

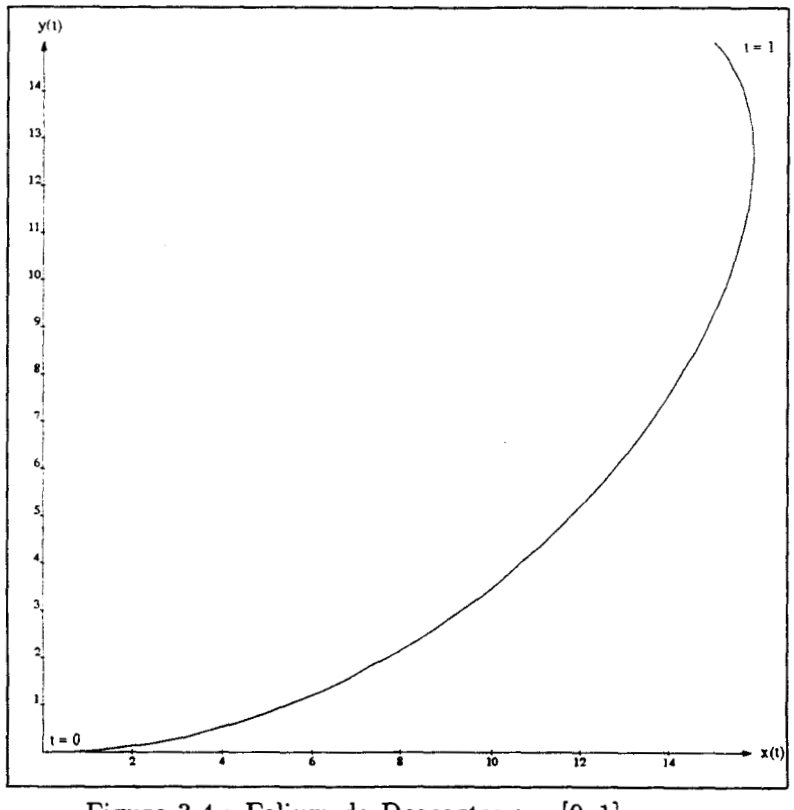


Figure 3.4 : Folium de Descartes sur $[0, 1]$

3.5 La simulation par les courbes de Bézier

3.5.1 Avantages et limites des courbes de Bézier

Nous avons choisi la représentation de Bézier car elle est d'une grande souplesse d'utilisation. En effet, les points de contrôle donnent l'allure générale de la courbe qui, pour nous, est l'entrée d'un système dynamique. Cela permet la modification en temps réel de l'entrée tout en devinant sa forme générale (sans en connaitre, à cette étape, l'expression mathématique

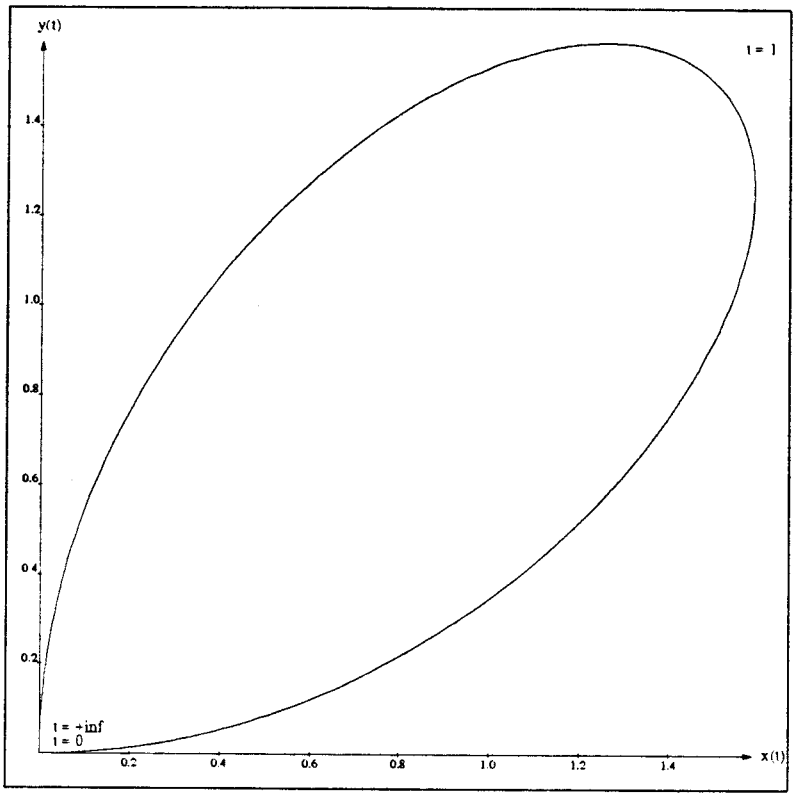


Figure 3.5 : Folium de Descartes sur $[0, +\infty[$

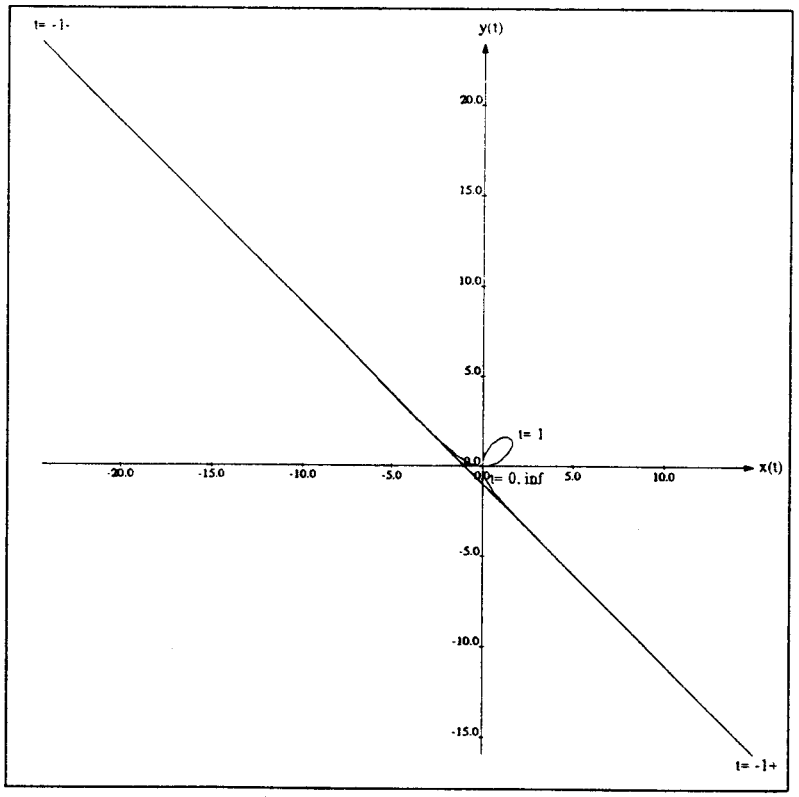


Figure 3.6 : Folium de Descartes sur $]-\infty, +\infty[$

exacte). Nous ne possédons pas les fichiers sources de l'interface graphique de Scratchpad (écrite en C), ce qui nous empèche de développer des programmes de déformation directe sur la courbe. Nous l'effectuons donc par des commandes Scratchpad en utilisant des paramètres réels à la place de valeurs numériques fixes pour les points de contrôle (voir paragraphe suivant).

L'implantation d'algorithmes rapides de tracé permet d'obtenir une visualisation rapide des courbes, ce qui rend l'utilisation du simulateur agréable.

Enfin, le principe de calcul des points par subdivisions de l'intervalle de temps (au lieu d'un pas de discrétisation pour la représentation canonique) permet de représenter de manière efficace la vitesse tout au long de la courbe (voir par exemple les figures 3.10 et 3.11 page 109 et 111).

L'utilisation de la paramétrisation et des courbes de Bézier rationnelles sous forme (BR) a permis la représentation des courbes polynomiales et rationnelles sur un intervalle de temps quelconque. Cela ne permettant pas de tracer des courbes exponentielles ou trigonométriques, nous simulons le comportement entrée/sortie de systèmes approchés par des fractions rationnelles en utilisant des polynômes exponentiels. Nous avons pour cela développé une technique de tracé par subdivision avec expression des polynômes dans la base de Bernstein (voir 3.3).

3.5.2 Déplacement des points de contrôle par utilisation de paramètres réels

Nous simulons dans cette partie le comportement entrée/sortie de systèmes dynamiques n'ayant pas de signification physique particulière. Nous montrons ici les possibilités de simulation qu'offrent nos programmes et notamment les avantages offerts par l'utilisation de paramètres réels dans l'expression des entrées.

3.5.2.1 Exemple 1

Considérons un sytème (Σ) défini sur un alphabet à deux lettres $Z = \{z_1, z_2\}$ par deux séries génératrices :

$$(\Sigma)\begin{cases} g_1 = & 3 - z_1 z_2 - z_2 z_1 + z_2^2 \\ g_2 = & -z_1^2 + z_2 z_1 z_2 \end{cases}$$

pour l'entrée dépendant du paramètre réel k donnée par la courbe de Bézier :

$$u(t) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} B_0^5(t) + \begin{bmatrix} 5 \\ 0 \end{bmatrix} B_1^5(t) + \begin{bmatrix} 6 \\ 3 \end{bmatrix} B_2^5(t) + \begin{bmatrix} 3 \\ 5 \end{bmatrix} B_3^5(t) + \begin{bmatrix} 5 \\ -k \end{bmatrix} B_4^5(t) + \begin{bmatrix} 0 \\ 2 \end{bmatrix} B_5^5(t)$$

c'est à dire, en base canonique :

$$\begin{cases} u_1(t) = & (5\,k + 21)\,t^5 + (-5\,k - 5)\,t^4 - 50\,t^3 + 40\,t^2 - 5\,t + 1 \\ u_2(t) = & -29\,t^5 + 40\,t^4 + 10\,t^3 - 50\,t^2 + 30\,t - 1 \end{cases}$$

Nous visualisons la courbe représentant cette entrée sur la figure 3.7 (page 107) pour k variant de -20 à 20.

Après évaluation de (Σ), nous obtenons l'expression symbolique des sorties, *dépendant toujours du paramètre* k, sous la forme de la courbe de Bézier suivante :

$$
\begin{aligned}
y(t) = &\begin{bmatrix} 3 \\ 0 \end{bmatrix} B_0^{18}(t) + \begin{bmatrix} 3 \\ 0 \end{bmatrix} B_1^{18} t + \begin{bmatrix} \frac{461}{153} \\ -\frac{1}{306} \end{bmatrix} B_2^{18}(t) + \begin{bmatrix} \frac{14645}{4896} \\ -\frac{1}{153} \end{bmatrix} B_3^{18}(t) + \begin{bmatrix} \frac{21749}{7344} \\ -\frac{67}{4896} \end{bmatrix} B_4^{18}(t) \\
&+ \begin{bmatrix} \frac{9476}{3213} \\ -\frac{179}{7344} \end{bmatrix} B_5^{18}(t) + \begin{bmatrix} -\frac{11}{111384}k + \frac{1968563}{668304} \\ \frac{1}{18564}k - \frac{56557}{1336608} \end{bmatrix} B_6^{18}(t) + \begin{bmatrix} \frac{1}{6048}k + \frac{3890381}{1336608} \\ \frac{359}{1336608}k - \frac{558629}{8019648} \end{bmatrix} B_7^{18}(t) \\
&+ \begin{bmatrix} \frac{1679}{565488}k + \frac{3738061}{1336608} \\ \frac{733}{668304}k - \frac{534647}{5513508} \end{bmatrix} B_8^{18}(t) + \begin{bmatrix} \frac{95789}{7351344}k + \frac{2224763}{864864} \\ \frac{2869}{864864}k - \frac{9526145}{88216128} \end{bmatrix} B_9^{18}(t) \\
&+ \begin{bmatrix} \frac{491}{13104}k + \frac{32876003}{14702688} \\ -\frac{1}{87516}k^2 + \frac{506573}{66162096}k - \frac{752669}{8270262} \end{bmatrix} B_{10}^{18}(t) + \begin{bmatrix} \frac{28295}{334152}k + \frac{785051}{432432} \\ -\frac{19}{190944}k^2 + \frac{501833}{33081048}k - \frac{19367791}{529296768} \end{bmatrix} B_{11}^{18}(t) \\
&+ \begin{bmatrix} \frac{217313}{1336608}k + \frac{951211}{700128} \\ -\frac{613}{1336608}k^2 + \frac{50209777}{1852538688}k + \frac{256212395}{3705077376} \end{bmatrix} B_{12}^{18}(t) + \begin{bmatrix} \frac{9479}{34272}k + \frac{347551}{376992} \\ -\frac{151}{102816}k^2 + \frac{5569099}{132324192}k + \frac{566567}{2395008} \end{bmatrix} B_{13}^{18}(t) \\
&+ \begin{bmatrix} \frac{43847}{102816}k + \frac{209149}{376992} \\ -\frac{53}{14688}k^2 + \frac{3689239}{66162096}k + \frac{215417}{462672} \end{bmatrix} B_{14}^{18}(t) + \begin{bmatrix} \frac{30901}{51408}k + \frac{77725}{282744} \\ -\frac{13}{1836}k^2 + \frac{1000495}{16540524}k + \frac{29238773}{40715136} \end{bmatrix} B_{15}^{18}(t) \\
&+ \begin{bmatrix} \frac{2476}{3213}k + \frac{15233}{565488} \\ -\frac{41}{3672}k^2 + \frac{7959773}{132324192}k + \frac{9576983}{10178784} \end{bmatrix} B_{16}^{18}(t) + \begin{bmatrix} \frac{667}{756}k - \frac{11587}{33264} \\ -\frac{1}{72}k^2 + \frac{3173587}{44108064}k + \frac{1792283}{1696464} \end{bmatrix} B_{17}^{18}(t) \\
&+ \begin{bmatrix} \frac{667}{756}k - \frac{33763}{33264} \\ -\frac{1}{72}k^2 + \frac{3990403}{44108064}k + \frac{1446707}{1696464} \end{bmatrix} B_{18}^{18}(t)
\end{aligned}
$$

C'est à dire en base canonique :

$$
\left\{
\begin{aligned}
y_1(t) = &\left(\frac{145}{24}k + \frac{667}{18}\right) t^{12} + \left(-\frac{52}{3}k - \frac{2936}{33}\right) t^{11} + \left(\frac{26}{3}k - \frac{200}{3}\right) t^{10} \\
&+ \left(\frac{730}{27}k + \frac{11665}{27}\right) t^9 + \left(-\frac{1175}{24}k - \frac{6985}{16}\right) t^8 + \left(\frac{1145}{42}k - \frac{4796}{21}\right) t^7 \\
&+ \left(-\frac{11}{6}k + \frac{14605}{18}\right) t^6 - \frac{1969}{3} t^5 + \frac{2755}{12} t^4 - \frac{235}{6} t^3 + 2\,t^2 + 3 \\
\\
y_2(t) = &\left(\frac{4205}{1296}k + \frac{5887}{432}\right) t^{18} + \left(-\frac{4495}{306}k - \frac{18589}{374}\right) t^{17} + \left(\frac{5273}{288}k - \frac{4055}{528}\right) t^{16} \\
&+ \left(\frac{7729}{324}k + \frac{1115339}{3564}\right) t^{15} + \left(-\frac{297965}{3024}k - \frac{34973075}{66528}\right) t^{14} \\
&+ \left(\frac{2012665}{19656}k - \frac{2553697}{16632}\right) t^{13} + \left(-\frac{25}{72}k^2 + \frac{95015}{9072}k + \frac{301487723}{199584}\right) t^{12} \\
&+ \left(\frac{5}{6}k^2 - \frac{168785}{1386}k - \frac{9883039}{5544}\right) t^{11} + \left(-\frac{1}{2}k^2 + \frac{193283}{1512}k + \frac{506395}{3024}\right) t^{10} \\
&+ \left(-\frac{103625}{1512}k + \frac{14666195}{9072}\right) t^9 + \left(\frac{3355}{168}k - \frac{105341}{56}\right) t^8 \\
&+ \left(-\frac{145}{42}k + \frac{268855}{252}\right) t^7 + \left(k - \frac{13127}{36}\right) t^6 + \frac{1169}{12} t^5 \\
&- \frac{175}{8} t^4 + \frac{8}{3} t^3 - \frac{1}{2} t^2
\end{aligned}
\right.
$$

La série est évaluée *une fois pour toutes*. Nous obtenons alors un résultat symbolique en fonction du paramètre k dont dépendent les entrées. Pour obtenir la représentation graphique, nous pouvons utiliser ce résultat en instanciant k. Cette méthode est cependant assez lente : pour les courbes présentées dans cet exemple, chaque instanciation demande environ 25 secondes.

Une technique beaucoup plus rapide, que nous utilisons ici, consiste à créer sous l'interpréteur des fonctions (ou éventuellement des règles de réécriture) qui seront compilées lors de la première exécution, ce qui prend environ 10 secondes. Chaque exécution sera ensuite très rapide, permettant de tracer de manière agréable les sorties du système en fonction de k.

Le tabeau ci-dessous nous donne les temps de calcul sur IBM PC-RT pour obtenir les expressions symboliques et les points de la courbe en fonction de la méthode utilisée. Le calcul de l'expression symbolique générale, dépendant du paramètre réel k, est très long (plus de 70 secondes pour l'expression de y_2) mais n'est pas utile pour le calcul des points de la courbe. Son intérêt est cependant loin d'être négligeable : on peut espérer repérer des valeurs particulières de k engendrant une expression intéressante de la sortie (diminution du degré ou annulation d'un point de contrôle par exemple).

Sur ce tableau, on remarque d'autre part que l'utilisation de fonctions ou de règles de réécriture est particulièrement avantageuse : Le tracé de la courbe de sortie est alors de 0.9 secondes, de l'instanciation de k à la donnée des points de la courbe. La méthode "sans paramètre réel" nécessite, après chaque instanciation de k, un calcul d'évaluation de (Σ) (sur notre exemple, environ 7 secondes), le temps de calcul des 41 courbes de la figure 3.8 devient alors énorme (et peu agréable à utiliser) par rapport à la méthode plus rapide et plus souple d'utilisation des fonctions et des règles de réécriture de Scratchpad : le calcul des points des 41 courbes de la figure 3.8 a pris 141 secondes, soit environ 3.4 secondes par courbe (ce temps comprend également la gestion des listes de points générées ainsi que le temps inévitable de gestion de l'espace mémoire du garbage collector).

Temps de calcul de la sortie du système (Σ)		
	Avec paramétre réel	Sans paramétre réel
Expression symbolique $y_1(t)$	57.9 sec	3.2 sec
Expression symbolique $y_2(t)$	70.833 sec	3.6 sec
points de la courbe	0.9 sec	0.4 sec

Nous donnons sur la figure 3.8 une simulation graphique des sorties de (Σ) pour k variant de -20 à 20. Sur la figure 3.9, nous donnons une représentation point par point des sorties, donnant une idée de la vitesse sur la courbe, pour $k = 2, 3, 4, 5$.

3.5.2.2 Exemple 2

Nous simulons un système défini par des séries sur deux lettres :

$$(\Sigma_1)\begin{cases} g_1 &= (z_1 - \frac{3}{4})(z_2 - \frac{1}{4}) \\ g_2 &= z_2 \end{cases}$$

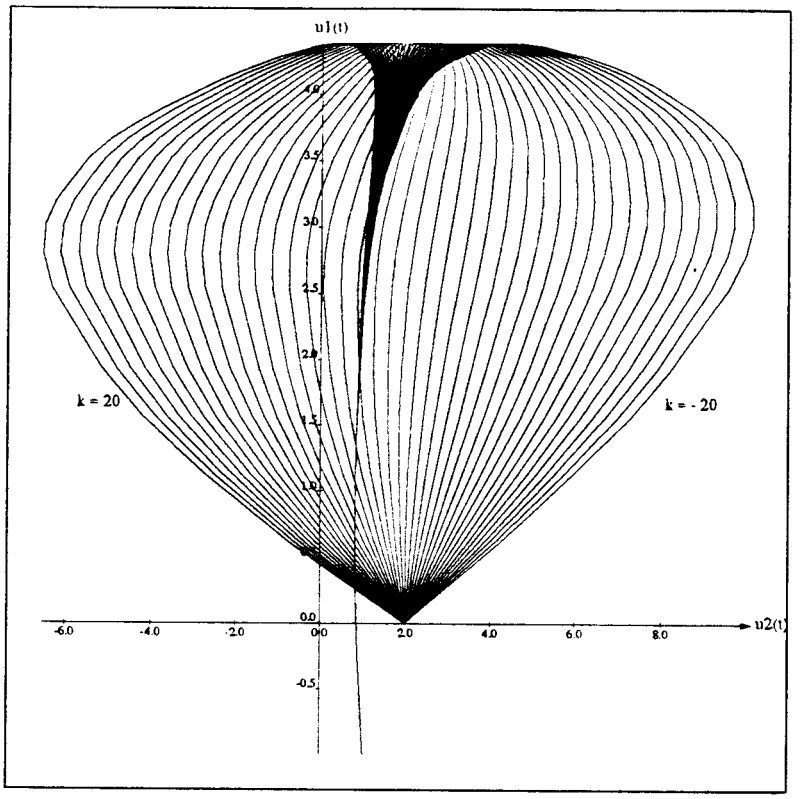


Figure 3.7 : représentation graphique de $(u_1(t), u_2(t))$ pour $-20 \leq k \leq 20$

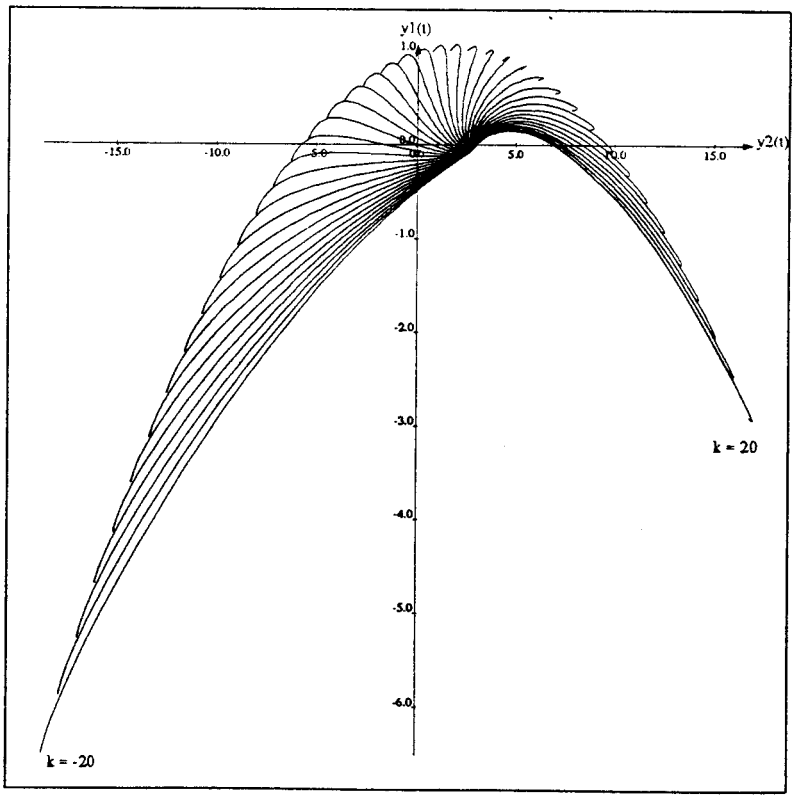


Figure 3.8 : représentation graphique de $(y_1(t), y_2(t))$ pour $-20 \leq k \leq 20$

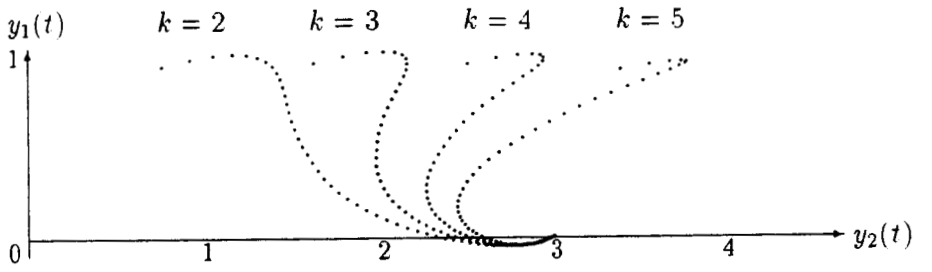


Figure 3.9 : représentation graphique de $(y_1(t), y_2(t))$ pour $k = 2, 3, 4, 5$.

Les entrées, données sous forme d'une courbe de Bézier ont pour valeur :

$$\begin{bmatrix}1\\1\end{bmatrix} B_0^3(t) + \begin{bmatrix}0\\-k1\end{bmatrix} B_1^3(t) + \begin{bmatrix}0\\k2\end{bmatrix} B_2^3(t) + \begin{bmatrix}1\\0\end{bmatrix} B_3^3(t)$$

Ce qui donne en base canonique :

$$\begin{cases} u_1(t) = 3\,t^2 - 3\,t + 1 \\ u_2(t) = (-3\,k2 - 3\,k1 - 1)\,t^3 + (3\,k2 + 6\,k1 + 3)\,t^2 + (-3\,k1 - 3)\,t + 1 \end{cases}$$

Par modification des paramètres réels $k1$ et $k2$, on déplace les deuxièmes et troisièmes points de contrôle de l'entrée : on "tire" ici le premier vers le bas et le second vers le haut de manière à déformer la courbe, on observe alors sur la figure 3.10 les répercussions sur la sortie.

L'expression symbolique de la sortie est :

$$\begin{bmatrix}\frac{3}{16}\\0\end{bmatrix} B_0^8(t) + \begin{bmatrix}\frac{1}{16}\\\frac{1}{8}\end{bmatrix} B_1^8(t) + \begin{bmatrix}\frac{9}{224}k1 + \frac{1}{112}\\-\frac{3}{56}k1 + \frac{11}{56}\end{bmatrix} B_2^8(t) + \begin{bmatrix}-\frac{3}{224}k2 + \frac{17}{224}k1 - \frac{1}{56}\\\frac{1}{56}k2 - \frac{1}{8}k1 + \frac{13}{56}\end{bmatrix} B_3^8(t)$$

$$+ \begin{bmatrix}-\frac{39}{1120}k2 + \frac{121}{1120}k1 - \frac{33}{1120}\\\frac{11}{280}k2 - \frac{53}{280}k1 + \frac{69}{280}\end{bmatrix} B_4^8(t) + \begin{bmatrix}-\frac{15}{224}k2 + \frac{29}{224}k1 - \frac{39}{1120}\\\frac{1}{8}k2 - \frac{13}{56}k1 + \frac{1}{4}\end{bmatrix} B_5^8(t)$$

$$+ \begin{bmatrix}-\frac{23}{224}k2 + \frac{15}{112}k1 - \frac{7}{160}\\\frac{11}{56}k2 - \frac{1}{4}k1 + \frac{1}{4}\end{bmatrix} B_6^8(t) + \begin{bmatrix}-\frac{13}{112}k2 + \frac{15}{112}k1 - \frac{69}{1120}\\\frac{1}{4}k2 - \frac{1}{4}k1 + \frac{1}{4}\end{bmatrix} B_7^8(t)$$

$$+ \begin{bmatrix}-\frac{13}{112}k2 + \frac{15}{112}k1 - \frac{13}{140}\\\frac{1}{4}k2 - \frac{1}{4}k1 + \frac{1}{4}\end{bmatrix} B_8^8(t)$$

c'est à dire, en base canonique :

$$\begin{cases} y_1(t) = \left(-\frac{3}{7}k2 - \frac{3}{7}k1 - \frac{1}{7}\right)t^7 + \left(\frac{5}{4}k2 + \frac{7}{4}k1 + \frac{3}{4}\right)t^6 + \left(-\frac{3}{2}k2 - 3k1 - \frac{17}{10}\right)t^5 \\ \qquad + \left(\frac{21}{16}k2 + \frac{51}{16}k1 + \frac{37}{16}\right)t^4 + \left(-\frac{3}{4}k2 - \frac{5}{2}k1 - \frac{5}{2}\right)t^3 + \left(\frac{9}{8}k1 + 2\right)t^2 - t + \frac{3}{16} \\ y_2(t) = \left(-\frac{3}{4}k2 - \frac{3}{4}k1 - \frac{1}{4}\right)t^4 + \left(k2 + 2k1 + 1\right)t^3 + \left(-\frac{3}{2}k1 - \frac{3}{2}\right)t^2 + t \end{cases}$$

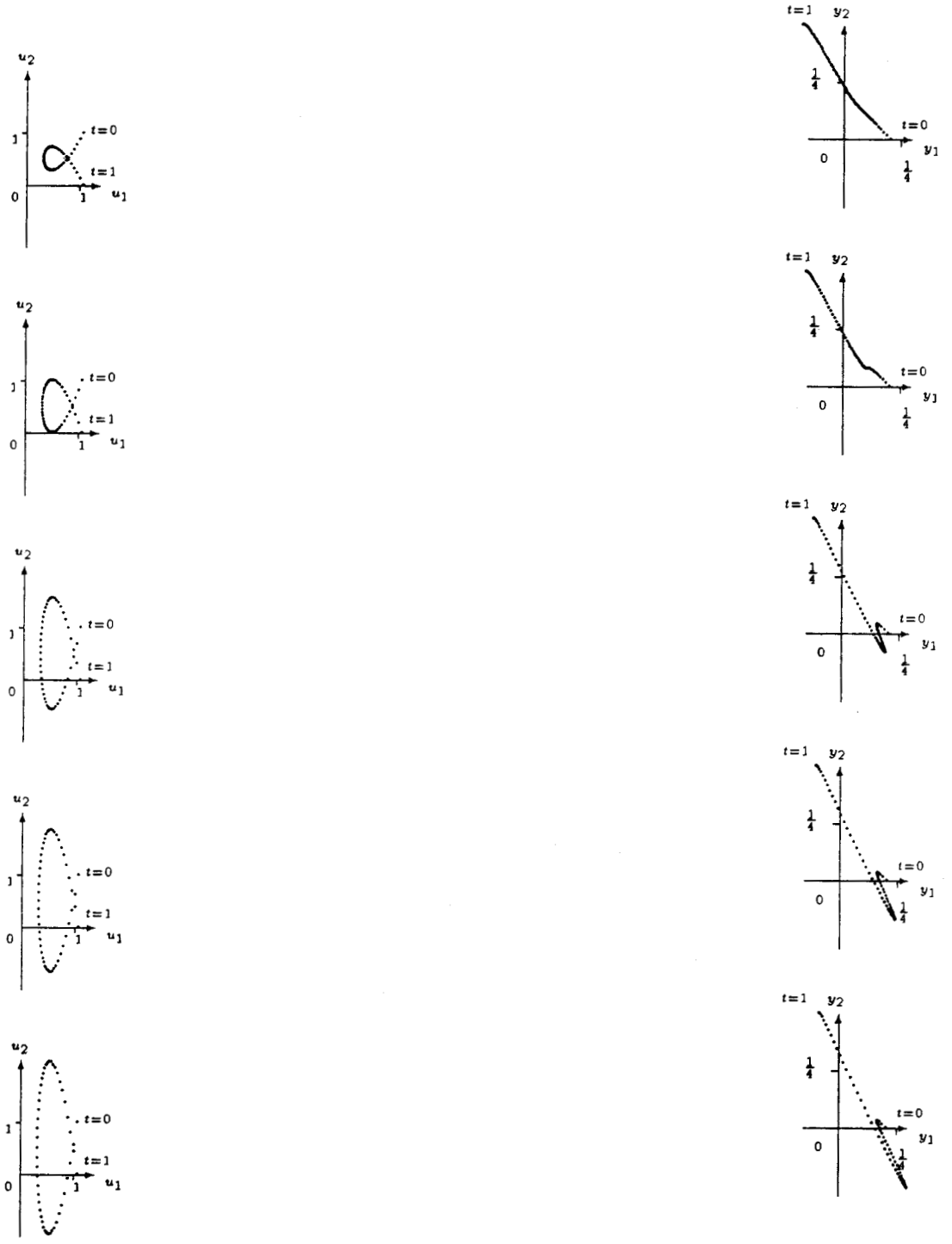


Figure 3.10 : simulation du comportement du système (Σ_1)

Temps de calcul de la sortie du système (Σ_1)		
	Avec paramètre réel	Sans paramètre réel
Expression symbolique $y_1(t)$	17.6 sec	1.2 sec
Expression symbolique $y_2(t)$	1.433 sec	0.333 sec
points de la courbe	0.867 sec	0.3 sec

On confirme sur ce tableau les remarques faites sur l'exemple précédent : l'instanciation des paramètres réels avant évaluation n'est pas avantageuse : les temps cumulés pour le tracé de la sortie donnent 1.833 secondes contre 0.867 secondes lors de l'utilisation de fonctions dépendant des paramètres réels. De plus, la relative simplicité des expressions des sorties nous donne ici des temps de calcul acceptables des expressions symboliques.

3.5.2.3 Exemple 3

Considérons le système à deux entrées et deux sorties défini par deux séries génératrices sur deux lettres :

$$(\Sigma_2)\begin{cases} g_1 = (z_1 - \frac{1}{2})(z_2 - \frac{1}{2}) \\ g_2 = z_1 \end{cases}$$

Nous modifions ici les deux entrées u_1 et u_2 par déplacement des points de contrôle dans la direction Nord-Est : On observe les déformations de la sortie sur la figure 3.11 lorsque k prend les valeurs 3, 4, 5 et 6 avec une représentation graphique point par point qui donne une information sur la vitesse de parcours le long de la courbe.

Notre technique de simulation permet également des variations plus petites du paramètre réel : on observe sur les figures 3.12 et 3.13 (page 112) l'évolution des entrées et des sorties lorsque le paramètre k varie de 5.1 à 8.1 avec un pas de 0.1.

Les entrées sont données par la courbe de Bézier suivante :

$$u(t) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} B_0^2(t) + \begin{bmatrix} k \\ k \end{bmatrix} B_1^2(t) + \begin{bmatrix} -7 \\ 7 \end{bmatrix} B_2^2(t)$$

c'est à dire, en base canonique :

$$\begin{cases} u_1(t) = (-2\,k - 7\,t^2) + 2\,k\,t \\ u_2(t) = (-2\,k + 7\,t^2) + 2\,k\,t \end{cases}$$

Nous obtenons après évaluation des séries g_1 et g_2 l'expression symbolique de la sortie, donnée

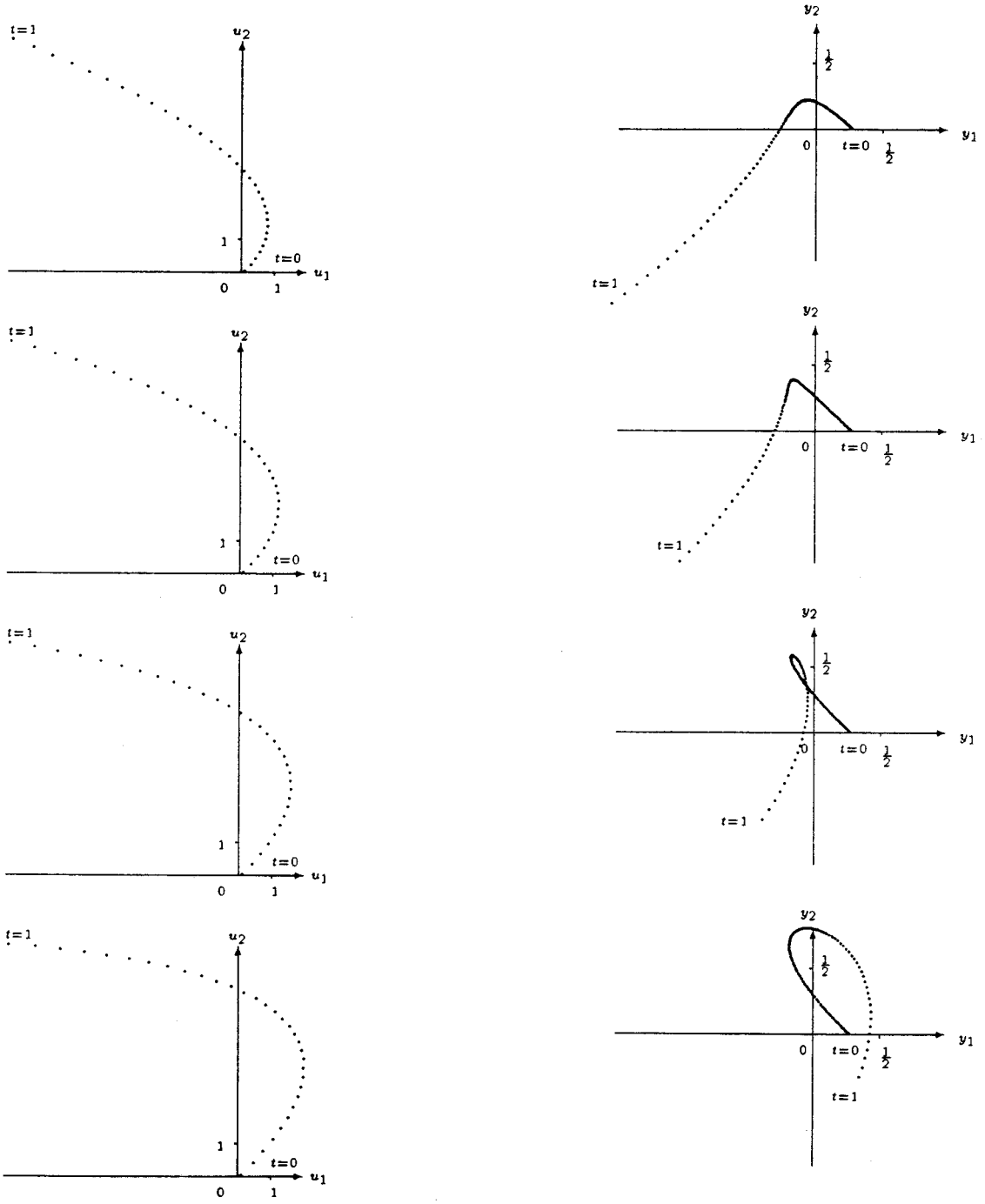


Figure 3.11 : simulation du comportement du système (Σ_2)

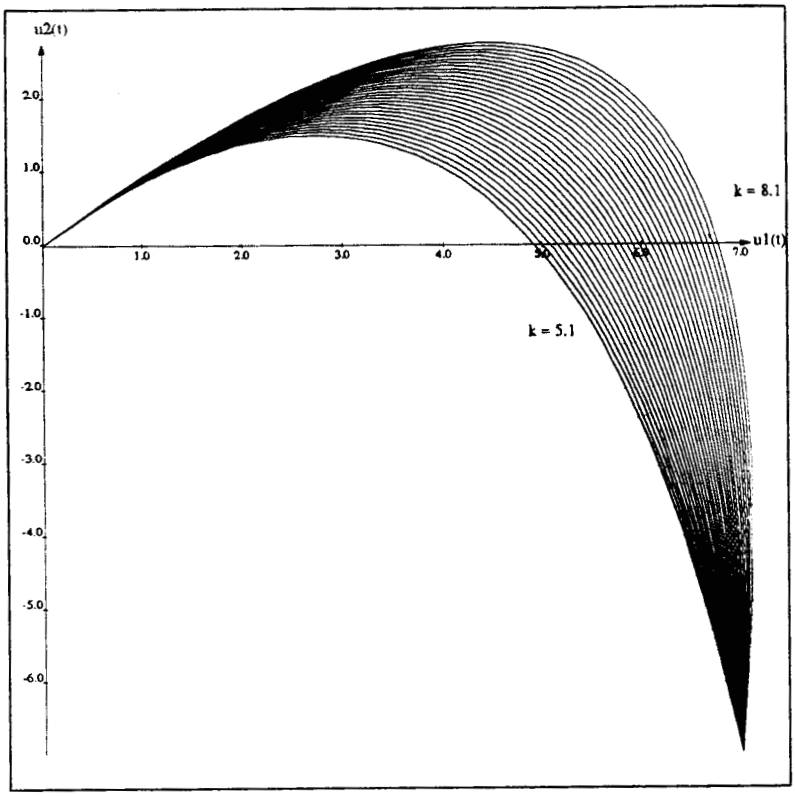


Figure 3.12 : Entrées du système Σ_2 pour $k \in [5.1, 8.1]$, $t \in [0, 1]$

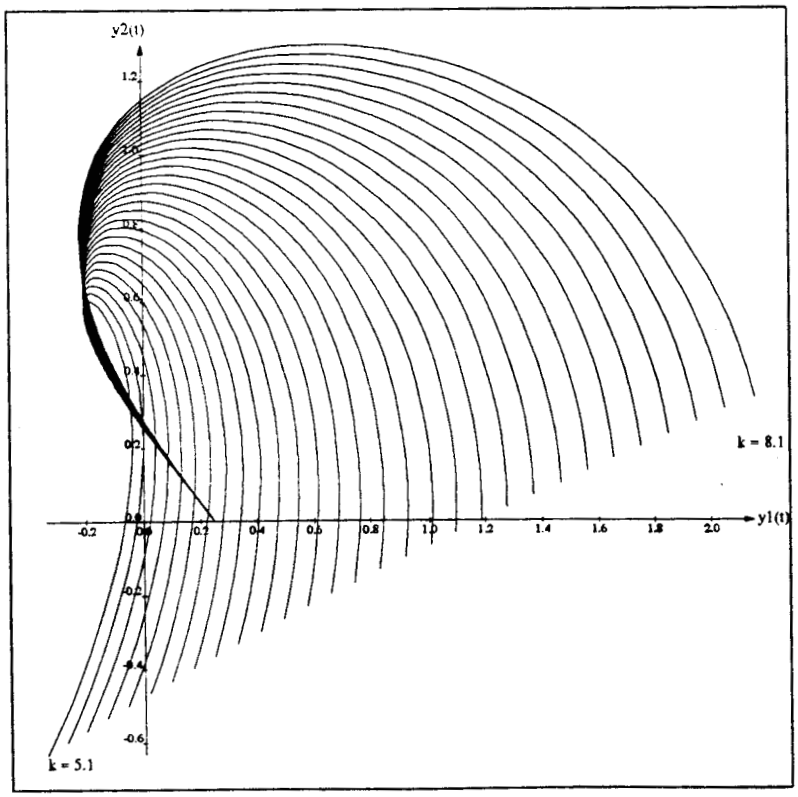


Figure 3.13 : Sorties du système Σ_2 pour $k \in [5.1, 8.1]$, $t \in [0, 1]$

par la courbe de Bézier suivante :

$$y(t) = \begin{bmatrix} \frac{1}{4} \\ 0 \end{bmatrix} B_0^6(t) + \begin{bmatrix} \frac{1}{4} \\ 0 \end{bmatrix} B_1^6(t) + \begin{bmatrix} -\frac{1}{15}k + \frac{1}{4} \\ \frac{1}{15}k \end{bmatrix} B_2^6(t) + \begin{bmatrix} -\frac{1}{6}k + \frac{1}{4} \\ \frac{1}{6}k + \frac{7}{60} \end{bmatrix} B_3^6(t) + \begin{bmatrix} \frac{1}{30}k^2 - \frac{4}{15}k + \frac{1}{4} \\ \frac{4}{15}k + \frac{7}{15} \end{bmatrix} B_4^6(t) + \begin{bmatrix} \frac{1}{18}k^2 - \frac{23}{90}k + \frac{1}{4} \\ \frac{1}{3}k + \frac{7}{6} \end{bmatrix} B_5^6(t) + \begin{bmatrix} \frac{1}{18}k^2 + \frac{2}{15}k - \frac{89}{36} \\ \frac{1}{3}k + \frac{7}{3} \end{bmatrix} B_6^6(t)$$

c'est à dire, dans la base canonique :

$$\begin{cases} y_1(t) = \left(\frac{2}{9}k^2 - \frac{49}{18}\right)t^6 + \left(-\frac{2}{3}k^2 + \frac{7}{15}k\right)t^5 + \frac{1}{2}k^2t^4 + \frac{2}{3}kt^3 - kt^2 + \frac{1}{4} \\ y_2(t) = \left(-\frac{2}{3}k + \frac{7}{3}\right)t^3 + kt^2 \end{cases}$$

Temps de calcul de la sortie du système (Σ_2)		
	Avec paramètre réel	Sans paramètre réel
Expression symbolique $y_1(t)$	15.7 sec	0.4 sec
Expression symbolique $y_2(t)$	2.4 sec	0.333 sec
points de la courbe	0.5 sec	0.35 sec

3.5.3 Approximation de séries génératrices infinies

Nous utilisons ici le simulateur pour visualiser la sortie du système décrit par la série génératrice

$$S = z_1 + S^{\omega^2} z_0$$

dans l'exemple 2.2.1 (page 52).
on note y_k l'évaluation de la série polynomiale S_k, $k \geq 1$ approchant S à l'ordre k.
Pour l'entrée $u_1 = 1$, on obtient :

$$y_1 = t$$
$$y_2 = t + \frac{t^3}{3}$$
$$y_3 = t + \frac{t^3}{3} + \frac{2t^5}{15} + \frac{t^7}{63}$$

La représentation graphique de ces trois sorties est donnée sur la figure 3.14.
Pour l'entrée $u_2 = 17t^2 - 14t + 2$, on obtient :

$$y_1 = \frac{17}{3}t^3 - 7t^2 + 2t$$

$$y_2 = \frac{4913}{81}t^9 - \frac{2023}{9}t^8 + \frac{3077}{9}t^7 - 273t^6 + \frac{362}{3}t^5 - 28t^4 + \frac{25}{3}t^3 - 7t^2 + 2t$$

$$y_3 = \frac{410338673}{137781}t^{21} - \frac{168962983}{6561}t^{20} + \frac{674432075}{6561}t^{19} - \frac{182994511}{729}t^{18} + \frac{305396993}{729}t^{17}$$
$$- \frac{122814307}{243}t^{16} + \frac{6829733}{15}t^{15} - \frac{76391749}{243}t^{14} + \frac{42206114}{243}t^{13} - \frac{6839336}{81}t^{12}$$
$$+ \frac{3324770}{81}t^{11} - \frac{904442}{45}t^{10} + \frac{697421}{81}t^9 - \frac{26551}{9}t^8 + \frac{58403}{63}t^7 - \frac{1043}{3}t^6 + \frac{1874}{15}t^5$$
$$- 28t^4 + \frac{25}{3}t^3 - 7t^2 + 2t$$

La représentation graphique de ces trois sorties est donnée sur la figure 3.15.

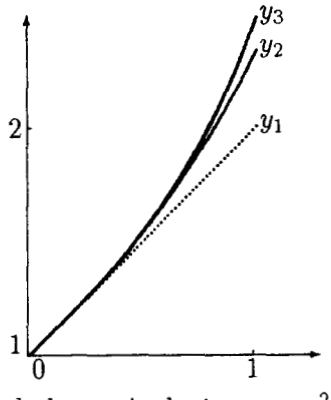


Figure 3.14 : approximation de la sortie de $\dot{y} = u + y^2, y(0) = 0$ pour l'entrée $u = 1$.

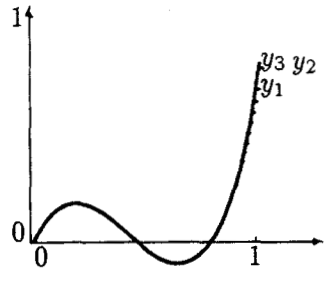


Figure 3.15 : approximation de la sortie de $\dot{y} = u + y^2, y(0) = 0$ pour l'entrée $u = 17t^2 - 14t + 2$.

Les tables ci-dessous fournissent les temps d'évaluation et de tracé sur IBM PC/RT. Notons que le temps de tracé comprend le temps (relativement important) de transfert de données entre Scratchpad et la fenêtre graphique.

$u = 1$	y_1	y_2	y_3
Temps d'évaluation	0.2*sec*	0.3*sec*	1.3*sec*
Temps de tracé	5.0*sec*	5.0*sec*	5.5*sec*

$u = 17t^2 - 14t + 2$	y_1	y_2	y_3
Temps d'évaluation	$0.3sec$	$1.0sec$	$12sec$
Temps de tracé	$5.0sec$	$5.6sec$	$6.1sec$

On remarque ici l'augmentation extrèmement rapide des temps de calcul d'évaluation en fonction de l'augmentation du degré (et donc de la précision) de la série polynomiale évaluée. En comparaison, les temps de tracé des sorties, qui augmentent linéairement, deviennent raisonnables pour des approximations significatives des séries génératrices : on voit notamment sur la figure 3.14 (page 114) que les courbes y_1 et y_2 sont très éloignées de la courbe y_3, seule proche (à 10^{-2} près du résultat réel qui est e (voir [28]).

3.5.4 D'autres exemples

Nous reprenons ici les exemples traités au chapitre 2 et pour lesquels nous nous étions arrétés au niveau du résultat symbolique. Nous observons le comportement du système sur différents intervalles de temps et sur divers types d'entrées.

3.5.4.1 Exemple 1

La simulation comprend en premier lieu la représentation graphique des différentes entrées des systèmes dynamiques. Nous en prenons, sur ce premier exemple, de deux types différents :

1. Entrée polynomiale $u_1(t)$ sous forme de courbes de Bézier.

$$u_1(t) = 5B_0^3(t) - 30B_1^3(t) - 20B_2^3(t) + 3B_3^3(t)$$

   Ce qui donne en base canonique :

$$u_1(t) = -32t^3 + 135t^2 - 105t + 5$$

   qui est représentée sur la figure 3.16 page 116

2. Entrée polynomiale-exponentielle.

   On peut également simuler le comportement des approximations polynomiales pour des entrées polynomiales exponentielles : Nous prenons ici une entrée $u_2(t)$:

$$u_2(t) = e^t + B_1^2(t) - 2e^{-2t}$$

   Ce qui donne en base canonique :

$$u_2(t) = e^t - 2e^{-2t} - 2t^2 + 2t$$

   qui est représentée sur la figure 3.17 page 116

Un premier exemple de simulation :
nous reprenons l'exemple 2.2.2 (page 53) du codage de l'équation différentielle $\dot{y} = u + y + y^2$.
Nous calculons l'évaluation des approximations polynomiales d'ordre 4,5 et 6 de

$$S_2 = z_1 z_0^* + 2z_1 z_0^* z_1 (2z_0)^* z_0 z_0^*$$

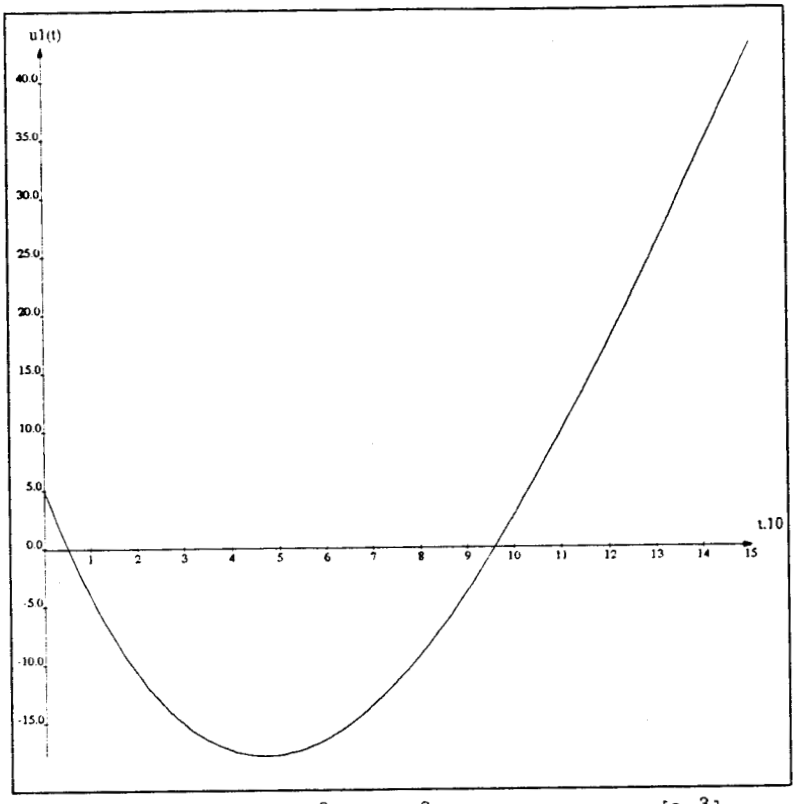


Figure 3.16 : $u_1(t) = -32t^3 + 135t^2 - 105t + 5, \quad t \in [0, \frac{3}{2}]$

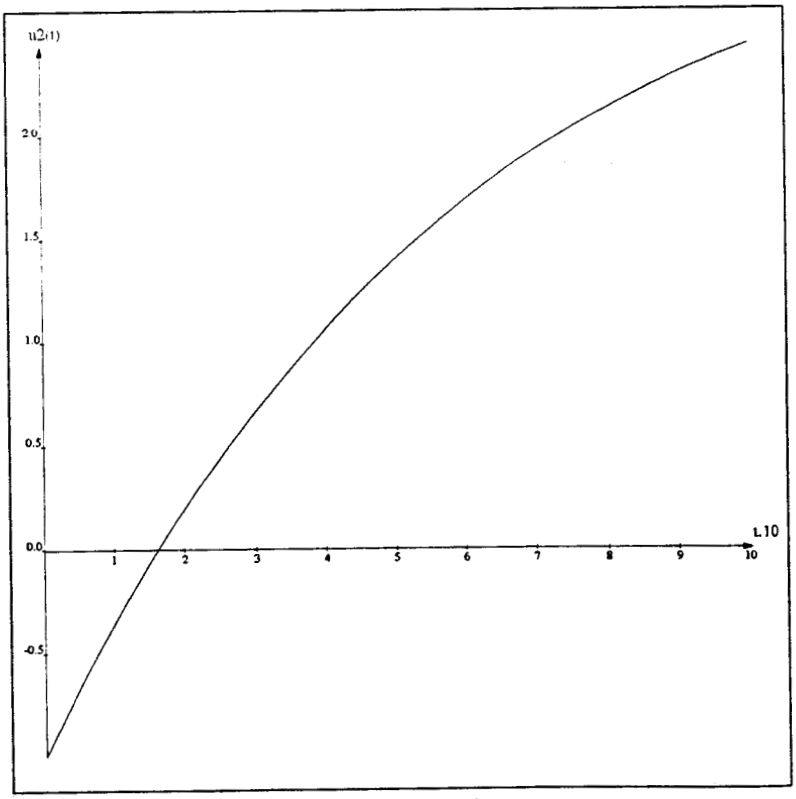


Figure 3.17 : $u_2(t) = e^t - 2e^{-2t} - 2t^2 + 2t, \quad t \in [0, 1]$

pour les entrées $u_1(t)$ et $u_2(t)$ citées précédemment.

Les expressions symboliques obtenues sont :

1. Approximation polynomiale d'ordre 4.

   (a) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{1}{4}]$

$$\begin{aligned} y^{(4)}_{1,[0,\frac{1}{4}]} =& \frac{1}{8}B^{10}_1(t) + \frac{13}{72}B^{10}_2(t) + \frac{395}{2304}B^{10}_3(t) + \frac{21523}{215040}B^{10}_4(t) \\ &- \frac{322739}{10321920}B^{10}_5(t) - \frac{22600127}{103219200}B^{10}_6(t) - \frac{377143243}{825753600}B^{10}_7(t) \\ &- \frac{134894733}{183500800}B^{10}_8(t) - \frac{343905337}{330301440}B^{10}_9(t) - \frac{2243770433}{1651507200}B^{10}_{10}(t) \end{aligned}$$

   (b) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{1}{2}]$

$$\begin{aligned} y^{(4)}_{1,[0,\frac{1}{2}]} =& \frac{1}{4}B^{10}_1(t) + \frac{2}{9}B^{10}_2(t) - \frac{13}{288}B^{10}_3(t) - \frac{22231}{40320}B^{10}_4(t) \\ &- \frac{407899}{322560}B^{10}_5(t) - \frac{3363757}{1612800}B^{10}_6(t) - \frac{6116689}{2150400}B^{10}_7(t) \\ &- \frac{21413197}{6451200}B^{10}_8(t) - \frac{21091393}{6451200}B^{10}_9(t) - \frac{15933457}{6451200}B^{10}_{10}(t) \end{aligned}$$

   (c) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, 1]$

$$\begin{aligned} y^{(4)}_{1,[0,1]} =& \frac{1}{2}B^{10}_1(t) - \frac{1}{9}B^{10}_2(t) - \frac{55}{36}B^{10}_3(t) - \frac{3397}{840}B^{10}_4(t) - \frac{64079}{10080}B^{10}_5(t) \\ &- \frac{9823}{1800}B^{10}_6(t) + \frac{18359}{12600}B^{10}_7(t) + \frac{25469}{1680}B^{10}_8(t) + \frac{213961}{6300}B^{10}_9(t) \\ &+ \frac{1380487}{25200}B^{10}_{10}(t) \end{aligned}$$

   (d) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{3}{2}]$

$$\begin{aligned} y^{(5)}_{1,[0,\frac{3}{2}]} =& \frac{3}{4}B^{10}_1(t) - B^{10}_2(t) - \frac{135}{32}B^{10}_3(t) - \frac{48879}{4480}B^{10}_4(t) - \frac{338153}{35840}B^{10}_5(t) \\ &+ \frac{3297663}{179200}B^{10}_6(t) + \frac{48425631}{716800}B^{10}_7(t) + \frac{79595147}{716800}B^{10}_8(t) \\ &+ \frac{18311859}{143360}B^{10}_9(t) + \frac{93461223}{716800}B^{10}_{10}(t) \end{aligned}$$

   (e) Evaluation pour l'entrée polynomiale-exponentielle $u_2(t)$ sur $[0, 1]$

$$\begin{aligned} y^{(4)}_{2,[0,1]} =& \frac{3}{2}e^{2t} \\ &+ \left(11B^4_0(t) + 7B^4_1(t) + 4B^4_2(t) + \frac{11}{6}B^4_3(t)\right)e^{t} \\ &- \frac{101}{8}B^8_0(t) - \frac{395}{32}B^8_1(t) - \frac{167}{14}B^8_2(t) - \frac{7675}{672}B^8_3(t) - \frac{761}{70}B^8_4(t) \\ &- \frac{8671}{840}B^8_5(t) - \frac{49}{5}B^8_6(t) - \frac{15641}{1680}B^8_7(t) - \frac{22247}{2520}B^8_8(t) \\ &- 3e^{-t} \\ &+ \left(\frac{25}{8}B^4_0(t) + \frac{29}{8}B^4_1(t) + \frac{33}{8}B^4_2(t) + \frac{113}{24}B^4_3(t) + \frac{45}{8}B^4_4(t)\right)e^{-2t} \end{aligned}$$

2. Approximation polynomiale d'ordre 5

(a) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{1}{4}]$

$$y^{(5)}_{1,[0.\frac{1}{4}]} = \frac{5}{44}B^{11}_1(t) + \frac{15}{88}B^{11}_2(t) + \frac{551}{3168}B^{11}_3(t) + \frac{127769}{1013760}B^{11}_4(t) + \frac{807449}{28385280}B^{11}_5(t)$$
$$- \frac{10587547}{90832896}B^{11}_6(t) - \frac{693499111}{2270822400}B^{11}_7(t) - \frac{6451076983}{12111052800}B^{11}_8(t)$$
$$- \frac{11493400123}{14533263360}B^{11}_9(t) - \frac{2777209559}{2595225600}B^{11}_{10}(t) - \frac{19742719897}{14533263360}B^{11}_{11}(t)$$

(b) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{1}{2}]$

$$y^{(5)}_{1,[0,\frac{1}{2}]} = \frac{5}{22}B^{11}_1(t) + \frac{5}{22}B^{11}_2(t) + \frac{1}{36}B^{11}_3(t) - \frac{7757}{21120}B^{11}_4(t) - \frac{833981}{887040}B^{11}_5(t)$$
$$- \frac{352199}{215040}B^{11}_6(t) - \frac{83840737}{35481600}B^{11}_7(t) - \frac{84463537}{28385280}B^{11}_8(t) - \frac{234390181}{70963200}B^{11}_9(t)$$
$$- \frac{447676379}{141926400}B^{11}_{10}(t) - \frac{82980299}{35481600}B^{11}_{11}(t)$$

(c) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, 1]$

$$y^{(5)}_{1,[0,1]} = \frac{5}{11}B^{11}_1(t) - \frac{113}{99}B^{11}_3(t) - \frac{12391}{3960}B^{11}_4(t) - \frac{18362}{3465}B^{11}_5(t)$$
$$- \frac{664043}{110880}B^{11}_6(t) - \frac{1643749}{554400}B^{11}_7(t) + \frac{1058149}{184800}B^{11}_8(t) + \frac{11584637}{554400}B^{11}_9(t)$$
$$+ \frac{2105479}{50400}B^{11}_{10}(t) + \frac{36728423}{554400}B^{11}_{11}(t)$$

(d) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{3}{2}]$

$$y^{(5)}_{1,[0,\frac{3}{2}]} = \frac{15}{22}B^{11}_1(t) - \frac{15}{22}B^{11}_2(t) - \frac{147}{44}B^{11}_3(t) - \frac{59679}{7040}B^{11}_4(t) - \frac{991227}{98560}B^{11}_5(t)$$
$$+ \frac{411189}{157696}B^{11}_6(t) + \frac{148958679}{3942400}B^{11}_7(t) + \frac{1434259539}{15769600}B^{11}_8(t)$$
$$+ \frac{113324067}{788480}B^{11}_9(t) + \frac{2782257081}{15769600}B^{11}_{10}(t) + \frac{8455209}{45056}B^{11}_{11}(t)$$

(e) Evaluation pour l'entrée polynomiale-exponentielle $u_2(t)$ sur $[0, 1]$,

$$y^{(5)}_{2,[0,1]} = 3e^{2t}$$
$$+ \left(\frac{35}{2}B^5_0(t) + \frac{117}{10}B^5_1(t) + \frac{69}{10}B^5_2(t) + \frac{29}{10}B^5_3(t) - \frac{8}{15}B^5_4(t) - \frac{56}{15}B^5_5(t)\right)e^t$$
$$- \frac{83}{4}B^9_0(t) - \frac{1441}{72}B^9_1(t) - \frac{1837}{96}B^9_2(t) - \frac{5219}{288}B^9_3(t) - \frac{7337}{432}B^9_4(t)$$
$$- \frac{237931}{15120}B^9_5(t) - \frac{4835}{336}B^9_6(t) - \frac{587213}{45360}B^9_7(t) - \frac{258197}{22680}B^9_8(t) - \frac{109439}{11340}B^9_9(t)$$
$$- \frac{9}{2}e^{-t}$$
$$+ \left(\frac{79}{16}B^5_0(t) + \frac{87}{16}B^5_1(t) + \frac{483}{80}B^5_2(t) + \frac{543}{80}B^5_3(t) + \frac{1861}{240}B^5_4(t)\right.$$
$$\left. + \frac{2173}{240}B^5_5(t)\right)e^{-2t}$$
$$- \frac{3}{16}e^{-4t}$$

3. Approximation polynomiale d'ordre 6

(a) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{1}{4}]$

$$\begin{aligned} y^{(6)}_{1,[0,\frac{1}{4}]} = & \frac{5}{48}B^{12}_1(t) + \frac{85}{528}B^{12}_2(t) + \frac{731}{4224}B^{12}_3(t) + \frac{215929}{1520640}B^{12}_4(t) \\ & + \frac{1120943}{16220160}B^{12}_5(t) - \frac{60027449}{1362493440}B^{12}_6(t) - \frac{4256230379}{21799895040}B^{12}_7(t) \\ & - \frac{439629813}{1153433600}B^{12}_8(t) - \frac{1301896625}{2179989504}B^{12}_9(t) - \frac{1460394391783}{1743991603200}B^{12}_{10}(t) \\ & - \frac{1908206072389}{1743991603200}B^{12}_{11}(t) - \frac{169224152717}{124570828800}B^{12}_{12}(t) \end{aligned}$$

(b) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{1}{2}]$

$$\begin{aligned} y^{(6)}_{1,[0,\frac{1}{2}]} = & \frac{5}{24}B^{12}_1(t) + \frac{5}{22}B^{12}_2(t) + \frac{41}{528}B^{12}_3(t) - \frac{22391}{95040}B^{12}_4(t) - \frac{1066691}{1520640}B^{12}_5(t) \\ & - \frac{5488249}{4257792}B^{12}_6(t) - \frac{110131969}{56770560}B^{12}_7(t) - \frac{2186199817}{851558400}B^{12}_8(t) \\ & - \frac{371981581}{121651200}B^{12}_9(t) - \frac{5582655473}{1703116800}B^{12}_{10}(t) - \frac{1049876131}{340623360}B^{12}_{11}(t) \\ & - \frac{988991471}{425779200}B^{12}_{12}(t) \end{aligned}$$

(c) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, 1]$

$$\begin{aligned} y^{(6)}_{1,[0,1]} = & \frac{5}{12}B^{12}_1(t) + \frac{5}{66}B^{12}_2(t) - \frac{113}{132}B^{12}_3(t) - \frac{14651}{5940}B^{12}_4(t) - \frac{69617}{15840}B^{12}_5(t) \\ & - \frac{1877063}{332640}B^{12}_6(t) - \frac{1260223}{266112}B^{12}_7(t) - \frac{87719}{1108800}B^{12}_8(t) + \frac{31952329}{3326400}B^{12}_9(t) \\ & + \frac{7525079}{302400}B^{12}_{10}(t) + \frac{75465353}{1663200}B^{12}_{11}(t) + \frac{232081897}{3326400}B^{12}_{12}(t) \end{aligned}$$

(d) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0, \frac{3}{2}]$

$$\begin{aligned} y^{(6)}_{1,[0,\frac{3}{2}]} = & \frac{5}{8}B^{12}_1(t) - \frac{5}{11}B^{12}_2(t) - \frac{471}{176}B^{12}_3(t) - \frac{23813}{3520}B^{12}_4(t) - \frac{529339}{56320}B^{12}_5(t) \\ & - \frac{2926743}{788480}B^{12}_6(t) + \frac{107880541}{6307840}B^{12}_7(t) + \frac{1759154149}{31539200}B^{12}_8(t) \\ & + \frac{678732759}{6307840}B^{12}_9(t) + \frac{10120305829}{63078400}B^{12}_{10}(t) + \frac{1804699421}{9011200}B^{12}_{11}(t) \\ & + \frac{3493612191}{15769600}B^{12}_{12}(t) \end{aligned}$$

(e) Evaluation pour l'entrée polynomiale-exponentielle $u_2(t)$ sur $[0,1]$

$$
\begin{aligned}
y^{(6)}_{2,[0,1]} =& 5e^{2t} \\
&+\Big(\frac{63}{4}B^6_0(t)+\frac{59}{6}B^6_1(t)+\frac{91}{20}B^6_2(t)-\frac{1}{5}B^6_3(t)-\frac{91}{20}B^6_4(t) \\
&\quad -\frac{1561}{180}B^6_5(t)-\frac{2303}{180}B^6_6(t)\Big)e^t \\
&-\frac{667}{32}B^{10}_0(t)-\frac{317}{16}B^{10}_1(t)-\frac{8909}{480}B^{10}_2(t)-\frac{49213}{2880}B^{10}_3(t)-\frac{77587}{5040}B^{10}_4(t) \\
&-\frac{283}{21}B^{10}_5(t)-\frac{57079}{5040}B^{10}_6(t)-\frac{12851}{1440}B^{10}_7(t)-\frac{78653}{12600}B^{10}_8(t) \\
&-\frac{1462673}{453600}B^{10}_9(t)+\frac{1259}{5670}B^{10}_{10}(t) \\
&-\frac{23}{4}e^{-t} \\
&+\Big(\frac{189}{32}B^6_0(t)+\frac{635}{96}B^6_1(t)+\frac{237}{32}B^6_2(t)+\frac{1331}{160}B^6_3(t)+\frac{4511}{480}B^6_4(t) \\
&\quad +\frac{15409}{1440}B^6_5(t)+\frac{17717}{1440}B^6_6(t)\Big)e^{-2t} \\
&-\frac{1}{16}e^{-4t}
\end{aligned}
$$

4. Approximation rationnelle d'ordre 1

(a) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0,1]$

$$y^{(r1)}_1 = -22e^t + 22B^3_0(t) + 31B^3_1(t) + 27B^3_2(t) + 42B^3_3(t)$$

(b) Evaluation pour l'entrée polynomiale-exponentielle $u_2(t)$ sur $[0,1]$

$$y^{(r1)}_2 = \Big(-\frac{8}{3}B^1_0(t)-\frac{5}{3}B^1_1(t)\Big)e^t + 2B^2_0(t) + 3B^2_1(t) + 6B^2_2(t) + \frac{2}{3}e^{-2t}$$

5. Approximation rationnelle d'ordre 2

(a) Evaluation pour l'entrée polynomiale $u_1(t)$ sur $[0,1]$

$$
\begin{aligned}
y^{(r2)}_1 =& 484e^{2t} \\
&+\Big(510740B^4_0(t)+510498B^4_1(t)+510157B^4_2(t)+509860B^4_3(t) \\
&\quad +509398B^4_4(t)\Big)e^t \\
&-511224B^6_0(t)-\frac{1192693}{2}B^6_1(t)-\frac{3492287}{5}B^6_2(t)-821818B^6_3(t) \\
&-\frac{4860611}{5}B^6_4(t)-\frac{2314209}{2}B^6_5(t)-1388172B^6_6(t)
\end{aligned}
$$

(b) Evaluation pour l'entrée polynomiale-exponentielle $u_2(t)$ sur $[0,1]$

$$
\begin{aligned}
y_2^{(r2)} = &\left(\frac{130}{9}B_0^2(t) + \frac{97}{9}B_1^2(t) + \frac{73}{9}B_2^2(t)\right)e^{2t} \\
&+\left(\frac{65981}{405}B_0^4(t) + \frac{260009}{1620}B_1^4(t) + \frac{127597}{810}B_2^4(t) + \frac{248579}{1620}B_3^4(t) + \frac{60221}{405}B_4^4(t)\right)e^{t} \\
&-178B_0^4(t) - \frac{443}{2}B_1^4(t) - \frac{836}{3}B_2^4(t) - \frac{711}{2}B_3^4(t) - 462B_4^4(t) \\
&+\left(\frac{13}{9}B_0^1(t) + \frac{7}{9}B_1^1(t)\right)e^{-t} \\
&+\left(-\frac{58}{81}B_0^2(t) - \frac{118}{81}B_1^2(t) - \frac{250}{81}B_2^2(t)\right)e^{-2t} \\
&-\frac{4}{45}e^{-4t}
\end{aligned}
$$

Nous observons sur la courbe 3.18 une parfaite similitude entre les différentes méthodes d'évaluation. Cependant, dès que l'intervalle de temps devient trop important (ici $t > 0.25$). les différentes méthodes donnent des résultats totalement différents. Nous pouvons ici être assurés de l'allure générale de la courbe : lorsque $t \in [0, 3/2]$, toutes les simulations donnent la même forme de courbe (voir figure 3.21) mais nullement des valeurs réelles de la sortie dès que $t > 0.25$. Nous avons également calculé l'expression symbolique de la sortie pour l'approximation rationnelle de degré 3, mais son expression devient tellement complexe qu'il nous est impossible actuellement de la simuler (les accumulations d'erreurs numériques nous donnant une courbe incohérente).

Temps de calcul des courbes pour l'entrée polynomiale $u_1(t) = -32t^3 + 135t^2 - 105t + 5$

	Calcul de l'expression symbolique	nombre de points calculés	Calcul des points de la courbe
Approximation polynomiale d'ordre 4	0.467 sec	81	0.233 sec
Approximation polynomiale d'ordre 5	0.7 sec	89	0.233 sec
Approximation polynomiale d'ordre 6	0.9 sec	97	0.233 sec
Approximation rationnelle d'ordre 1	0.167 sec	65	0.967 sec
Approximation rationnelle d'ordre 2	1.0 sec	65	2.067 sec
Méthode numérique de Runge-Kutta	-	100	1.333 sec

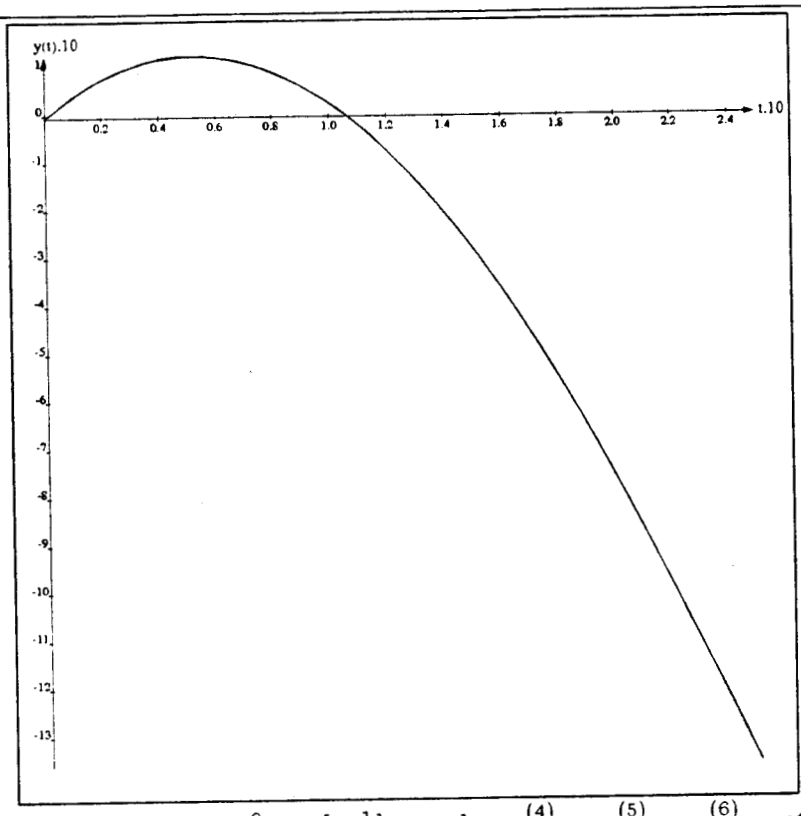


Figure 3.18 : sortie de $\dot{y} = u_1 + y + y^2$ sur $[0, \frac{1}{4}]$. courbes $y^{(4)}_{1,[0,\frac{1}{4}]}, y^{(5)}_{1,[0,\frac{1}{4}]}, y^{(6)}_{1,[0,\frac{1}{4}]}$ et simulation numérique.

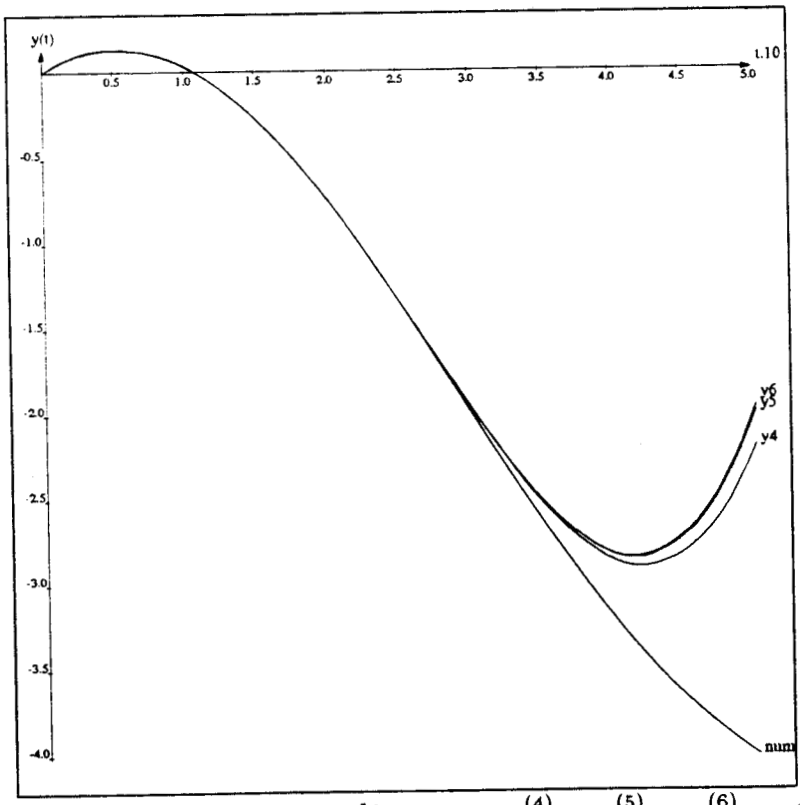


Figure 3.19 : sortie de $\dot{y} = u_1 + y + y^2$ sur $[0, \frac{1}{2}]$. courbes $y^{(4)}_{1,[0,\frac{1}{2}]}, y^{(5)}_{1,[0,\frac{1}{2}]}, y^{(6)}_{1,[0,\frac{1}{2}]}$ et simulation numérique.

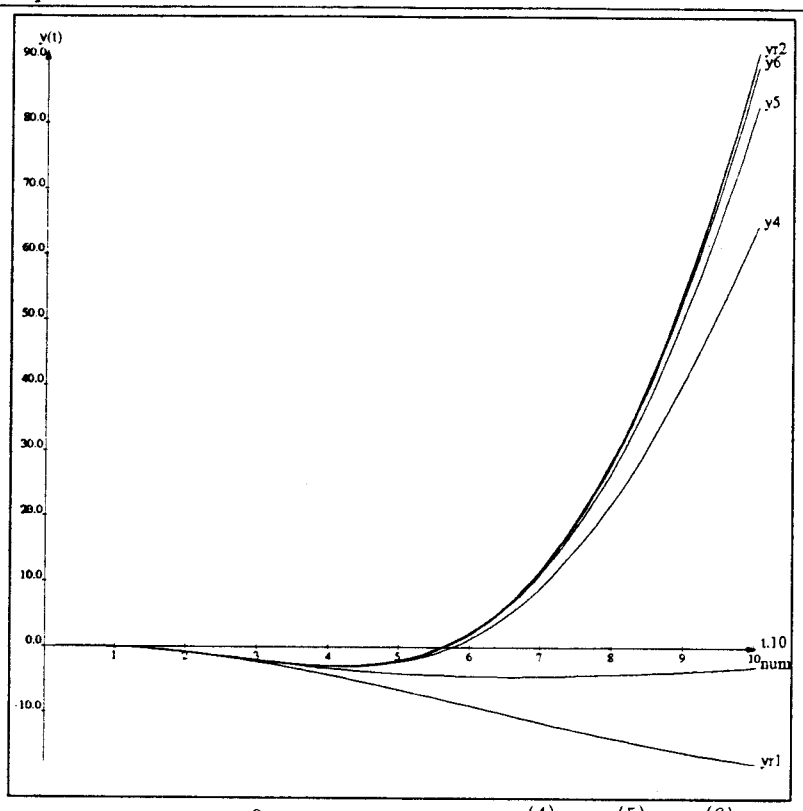


Figure 3.20 : sortie de $\dot{y} = u_1 + y + y^2$ sur $[0,1]$. courbes $y^{(4)}_{1,[0,1]}, y^{(5)}_{1,[0,1]}, y^{(6)}_{1,[0,1]}$ et simulation numérique.

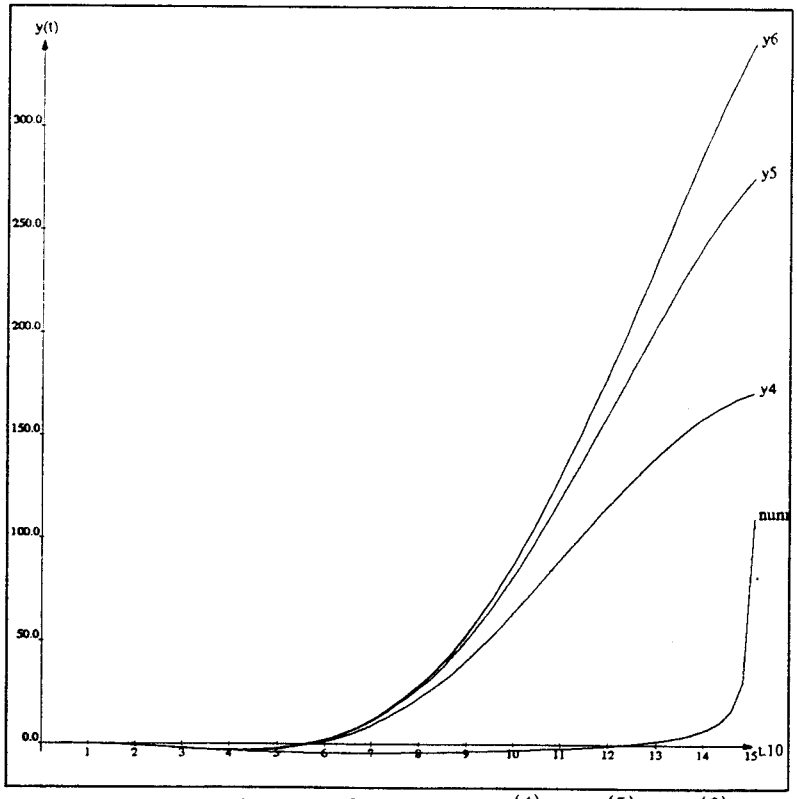


Figure 3.21 : sortie de $\dot{y} = u_1 + y + y^2$ sur $[0, \frac{3}{2}]$. courbes $y^{(4)}_{1,[0,\frac{3}{2}]}, y^{(5)}_{1,[0,\frac{3}{2}]}, y^{(6)}_{1,[0,\frac{3}{2}]}$ et simulation numérique.

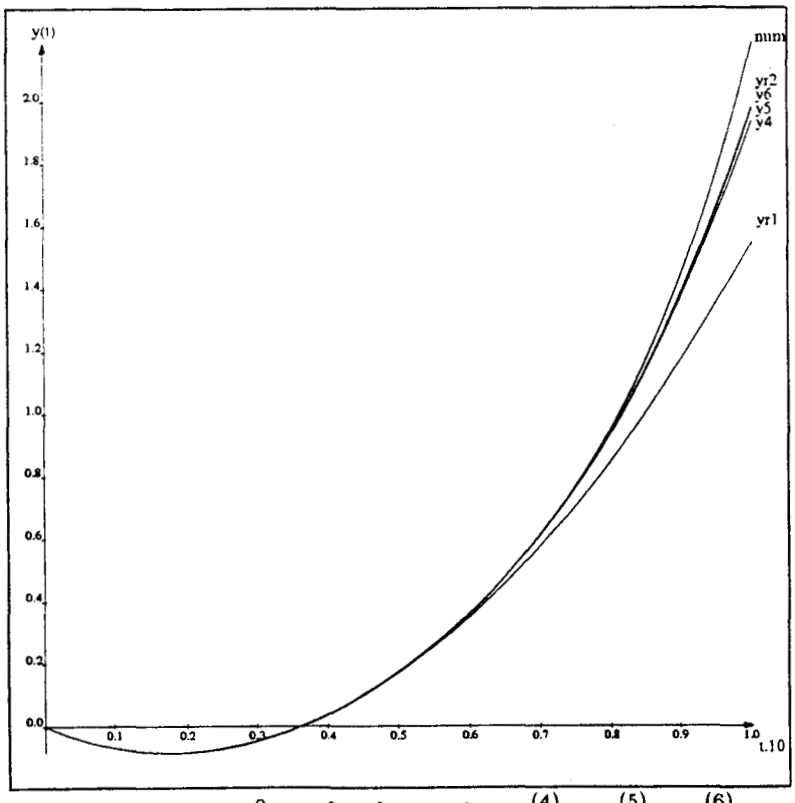


Figure 3.22 : sortie de $\dot{y} = u_2 + y + y^2$ sur $[0,1]$. courbes $y^{(4)}_{2,[0,1]}, y^{(5)}_{2,[0,1]}, y^{(6)}_{2,[0,1]}$ et simulation numérique.

Ce tableau met en évidence les avantages des techniques de simulation des approximations rationnelles par rapport à la simulation polynomiale : les temps de calcul des expressions symboliques fournies par l'approximation polynomiale d'ordre 6 et par l'approximation rationnelle d'ordre 2 sont presque identiques. Tout calcul d'une approximation polynomiale d'ordre supérieur sera plus long et moins précis que celui de l'approximation rationnelle.

Les temps de calcul des points de la courbe pour les sorties des approximations rationnelles sont beaucoup plus grands que les autres car les sorties (voir pages 3.5.4.1 à 3.5.4.1) sont polynomiale-exponentielles, ce qui entraine un plus grand nombre de calculs (voir la technique décrite en 3.3).

Le tableau suivant (pour l'entrée polynomiale-exponentielle $u_2(t)$), confirme nos affirmations : de 0.7 secondes de calcul pour l'expression symbolique de la sortie pour l'approximation rationnelle, nous passons à 26.7 secondes pour l'approximation polynomiale d'ordre 6. Cette différence de temps s'explique aisément : la technique d'approximation polynomiale revient à approcher les expressions exponentielles des résultats exacts par des polynômes. Pour obtenir des approximations acceptables qualitativement, il est nécessaire de prendre des polynômes de degré élevé (c'est le cas de l'approximation polynomiale de degré 6), ce qui génère lors de l'évaluation un nombre important d'intégrations (nous utilisons ici une représentation récursive des polynômes, nous n'effectuons alors pas de convolution). Ceci entraine des temps de calculs importants auquels s'ajoutent la gestion de coefficients de plus en plus imposants lors du calcul exact de la sortie (voir par exemple l'expression de $y^{(6)}_{2,[0,1]}$ page 120).

Temps de calculs des courbes pour l'entrée polynomiale exponentielle $u_2(t) = e^t - 2e^{-2t} - 2t^2 + 2t$

	Calcul de l'expression symbolique	nombre de points calculés	Calcul des points de la courbe
Approximation polynomiale d'ordre 4	4.0 sec	65	4.6 sec
Approximation polynomiale d'ordre 5	18.733 sec	65	6.4 sec
Approximation polynomiale d'ordre 6	26.7 sec	65	18.133 sec
Approximation rationnelle d'ordre 1	0.1 sec	65	1.0 sec
Approximation rationnelle d'ordre 2	0.7 sec	65	2.167 sec
Méthode numérique de Runge-Kutta	-	65	0.767 sec

3.5.4.2 Exemple 2

Nous reprenons ici la même série génératrice que dans l'exemple précédent, mais pour l'entrée polynomiale $u_3(t) = t$.

Nous observons sur la courbe 3.23 page 127 le tracé des sorties calculées par évaluation des mêmes approximations polynomiales que précédemment, par la méthode numérique de Runge-Kutta et par des évaluations des frations rationnelles S_1, S_2 et S_3 données en 2.2.2 page 53. L'expression très simple de l'entrée nous permet ici d'obtenir des courbes de sortie qui convergent sur un intervalle de temps plus important.

Nous observons alors sur les figures 3.24, 3.25 et 3.26 pages 127 et 128 les différences respectivement obtenues entre les approximations polynomiales d'ordre 6, les approximations rationnelles d'ordre 2 et 3 et le calcul par la méthode numérique de Runge-Kutta.

On voit que les valeurs obtenues par simulation numérique et par approximation rationnelle oscillent l'une par rapport à l'autre (courbes 3.25 et 3.26) sur un intervalle de temps donné avant de diverger de manière exponentielle. La forme surprenante de la courbe 3.24 est due à la méthode de calcul par des polynômes de Bézier : Les points de subdivision exacts coïncident avec la simulation numérique, tandis que les points approchés des polygones internes génèrent un écart qui donne la forme en arc de cercle.

Les expressions symboliques obtenues par les différentes méthodes sont :

1. Evaluation de l'approximation polynomiale de degré 4

$$y_3^{(4)} = \frac{1}{30}B_2^6(t) + \frac{13}{120}B_3^6(t) + \frac{17}{72}B_4^6(t) + \frac{317}{720}B_5^6(t) + \frac{289}{360}B_6^6(t)$$

2. Evaluation de l'approximation polynomiale de degré 5

$$y_3^{(5)} = \frac{1}{42}B_2^7(t) + \frac{8}{105}B_3^7(t) + \frac{137}{840}B_4^7(t) + \frac{53}{180}B_5^7(t) + \frac{827}{1680}B_6^7(t) + \frac{4129}{5040}B_7^7(t)$$

3. Evaluation de l'approximation polynomiale de degré 6

$$\begin{aligned} y_3^{(6)} = &\frac{1}{56}B_2^8(t) + \frac{19}{336}B_3^8(t) + \frac{67}{560}B_4^8(t) + \frac{1427}{6720}B_5^8(t) \\ &+ \frac{2311}{6720}B_6^8(t) + \frac{3071}{5760}B_7^8(t) + \frac{923}{1120}B_8^8(t) \end{aligned}$$

4. Evaluation de l'approximation rationnelle d'ordre 1

$$y_3^{(r1)} = e^t - B_0^1(t) - 2B_1^1(t)$$

5. Evaluation de l'approximation rationnelle d'ordre 2

$$y_3^{(r2)} = e^{2t} + \left(5B_0^2(t) + 4B_1^2(t) + 2B_2^2(t)\right)e^t - 6B_0^2(t) - \frac{17}{2}B_1^2(t) - 12B_2^2(t)$$

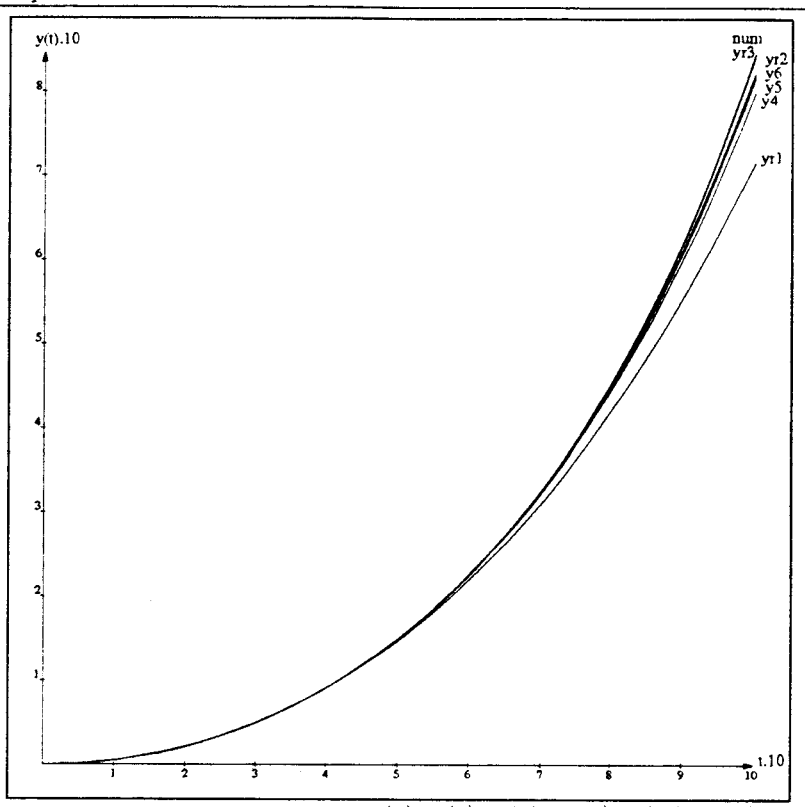


Figure 3.23 : Sortie de $\dot{y} = u_3 + y + y^2$. courbes $y_3^{(4)}$, $y_3^{(5)}$, $y_3^{(6)}$, $y_3^{(r1)}$, $y_3^{(r2)}$, $y_3^{(r3)}$, et simulation numérique.

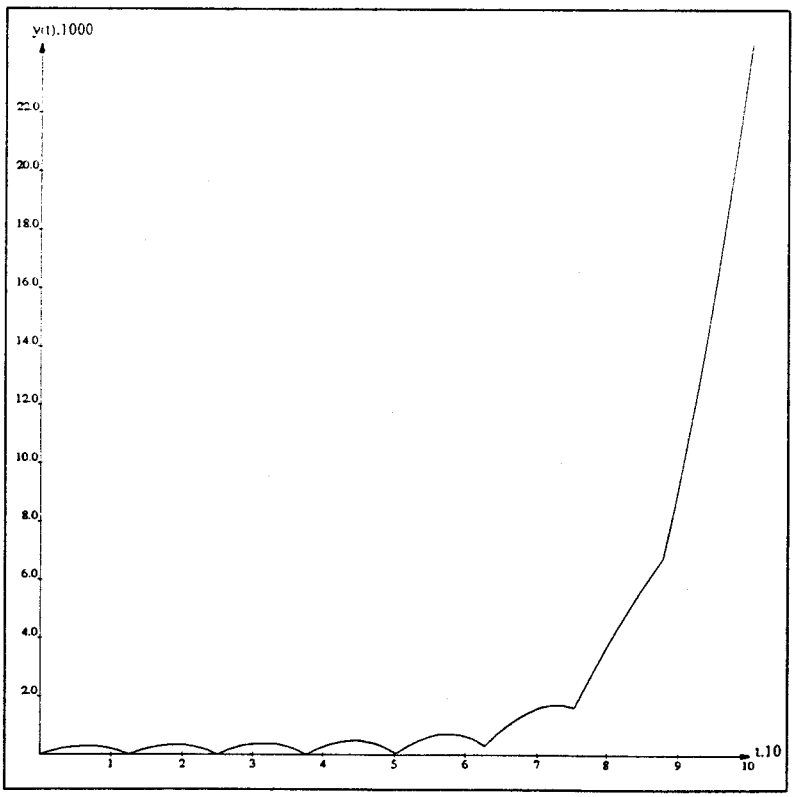


Figure 3.24 : Sortie de $\dot{y} = u_3 + y + y^2$. Différence entre l'approximation polynomiale d'ordre 6 et la méthode de simulation numérique.

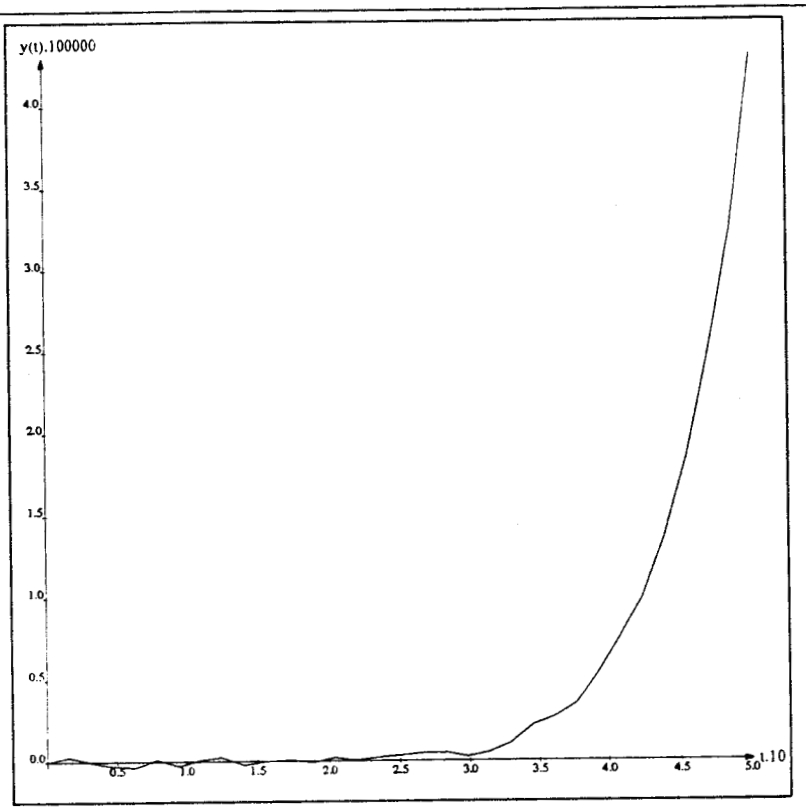


Figure 3.25 : Sortie de $\dot{y} = u_3 + y + y^2$. Différence entre l'approximation rationnelle d'ordre 2 et la méthode de simulation numérique.

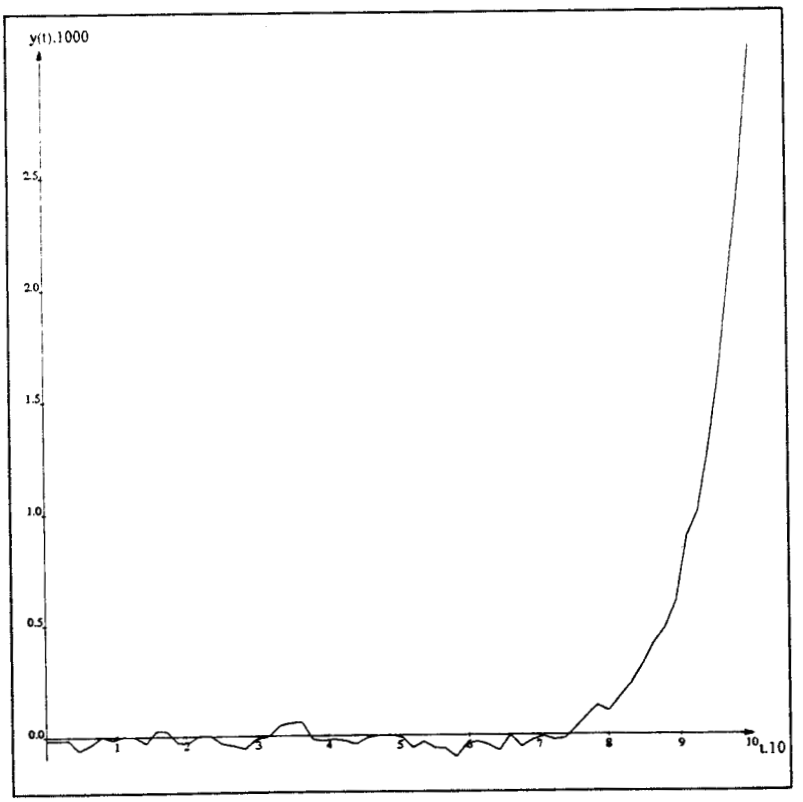


Figure 3.26 : Sortie de $\dot{y} = u_3 + y + y^2$. Différence entre l'approximation rationnelle d'ordre 3 et la méthode de simulation numérique.

6. Evaluation de l'approximation rationnelle d'ordre 3

$$\begin{aligned} y_3^{(r3)} = & \frac{1}{3}\epsilon^{4t} \\ & + \left(\frac{11}{2}B_0^2(t) + 5B_1^2(t) + \frac{7}{2}B_2^2(t)\right)\epsilon^{3t} \\ & + \left(27B_0^4(t) + \frac{47}{2}B_1^4(t) + \frac{56}{3}B_2^4(t) + \frac{25}{2}B_3^4(t) + 6B_4^4(t)\right)\epsilon^{2t} \\ & + \left(\frac{1333}{6}B_0^5(t) + \frac{1261}{6}B_1^5(t) + \frac{2953}{15}B_2^5(t) + 183B_3^5(t)\right. \\ & \qquad \left. + 170B_4^5(t) + \frac{802}{5}B_5^5(t)\right)\epsilon^{t} \\ & - 255B_0^4(t) - \frac{1239}{4}B_1^4(t) - \frac{1133}{3}B_2^4(t) - \frac{1849}{4}B_3^4(t) - 568B_4^4(t) \end{aligned}$$

3.5.4.3 Exemple 3

Nous reprenons l'exemple 2.2.3 donné page 59 de l'équation différentielle $\dot{y} + a(t)y = b(t)$ codée par la fraction rationnelle $S = z_b z_a^*$.

Nous évaluons ici S pour les trois entrées polynomiales-exponentielles suivantes :

1. Première entrée en base de Bernstein :

$$b_1 = 2B_1^1(t)\epsilon^{3t} - \epsilon^{2t}$$

soit en base canonique :

$$b_1 = 2t\epsilon^{3t} - \epsilon^{2t}$$

représentée sur la figure 3.27

2. Deuxième entrée en base de Bernstein :

$$b_2 = \frac{2}{5}B_1^1(t)\epsilon^{7t} - 2B_1^1(t)\epsilon^{3t} + e^{2t} + \epsilon^{-5t}$$

soit en base canonique :

$$b_2 = \frac{2}{5}t\epsilon^{7t} - 2t\epsilon^{3t} + \epsilon^{2t} + e^{-5t}$$

représentée sur la figure 3.29

3. Troisième entrée en base de Bernstein :

$$b_3 = b_1 + b_2 = \frac{2}{5}B_1^1(t)e^{7t} + e^{-5t}$$

soit en base canonique :

$$b_3 = \frac{2}{5}te^{7t} + e^{-5t}$$

représentée sur la figure 3.31

Les résultats de l'évaluation donnent les expressions symboliques suivantes :

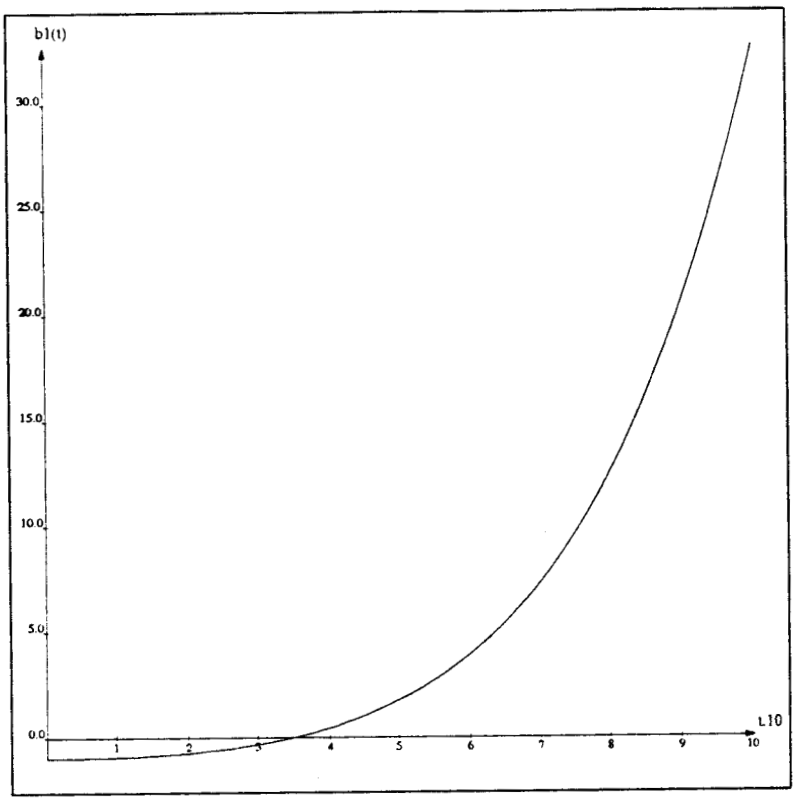


Figure 3.27 : $b_1(t) = 2te^{3t} - e^{2t}, \quad t \in [0,1]$

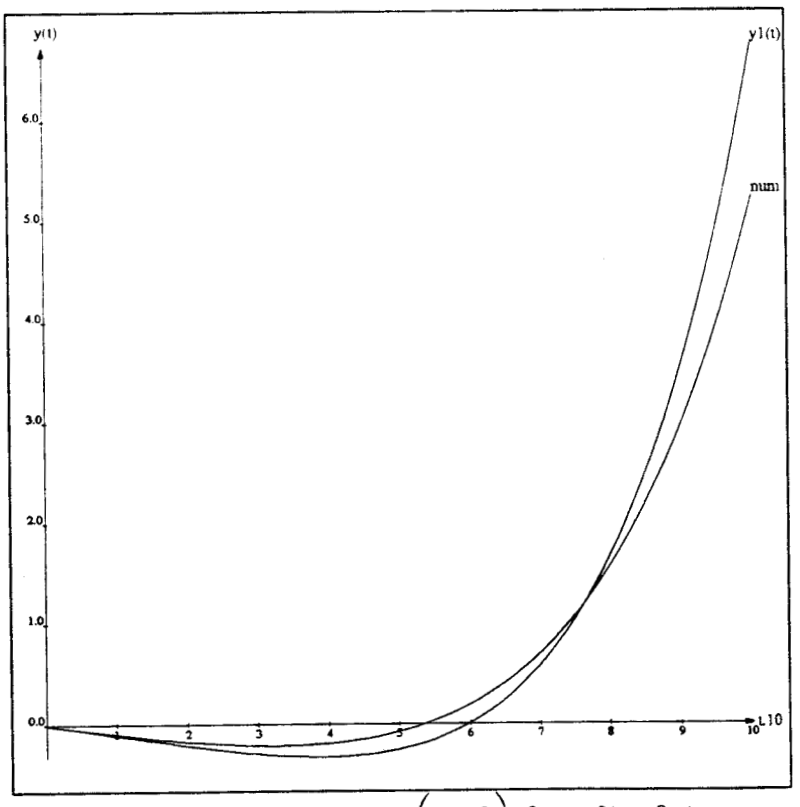


Figure 3.28 : Sortie de $\dot{y} + y = b_1$, $y_1(t) = \left(t - \frac{1}{2}\right)e^{3t} - e^{2t} + \frac{3}{2}e^t, \quad t \in [0,1]$

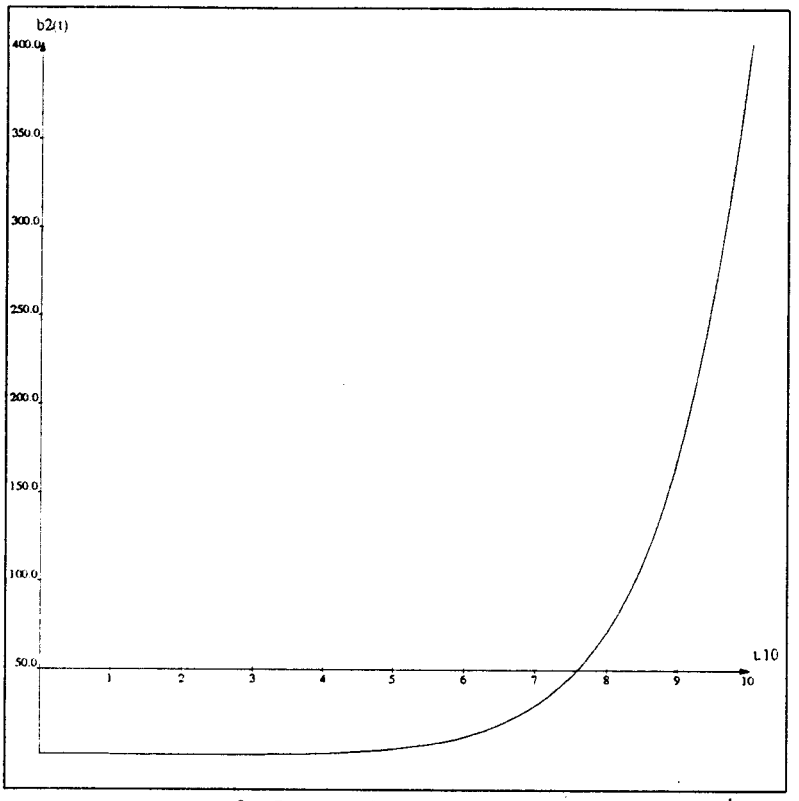


Figure 3.29 : $b_2 = \frac{2}{5}te^{7t} - 2te^{3t} + e^{2t} + e^{-5t}, \quad t \in [0,1]$

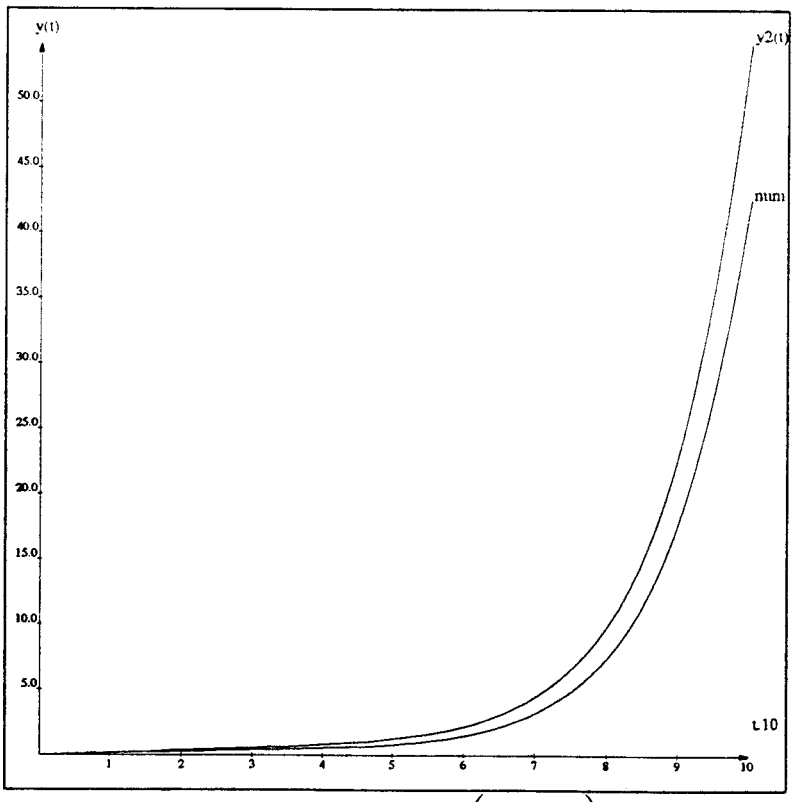


Figure 3.30 : Sortie de $\dot{y} + y = b_2$, $y_2 = \frac{1}{15}t - \frac{1}{90}e^{7t} + \left(-t + \frac{1}{2}\right)e^{3t} + e^{2t} - \frac{119}{90}e^{t} - \frac{1}{6}e^{-5t}$

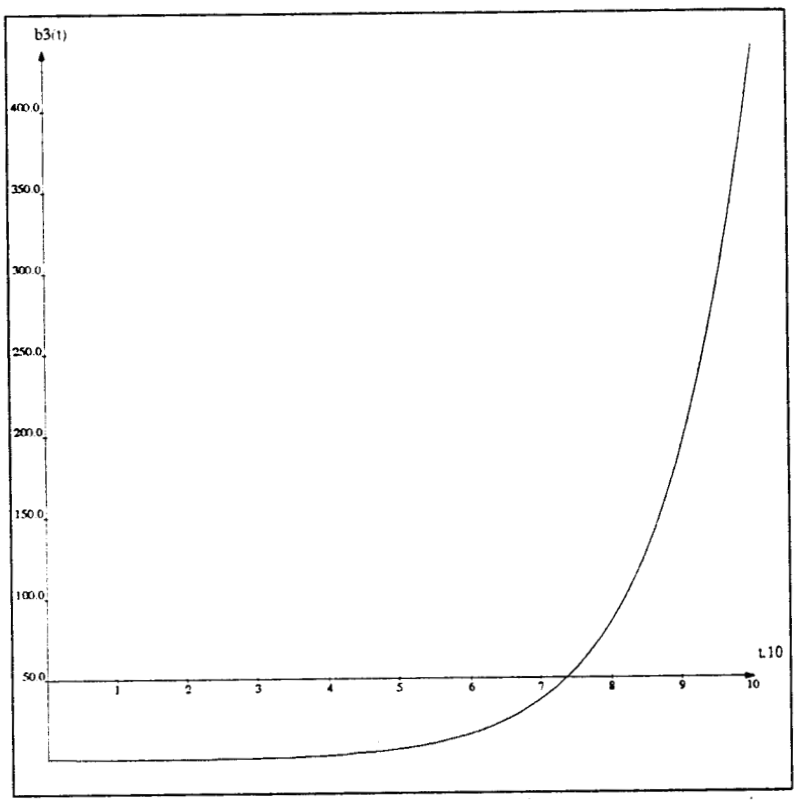


Figure 3.31 : $b_3 = \frac{2}{5}te^{7t} + e^{-5t}, \quad t \in [0,1]$

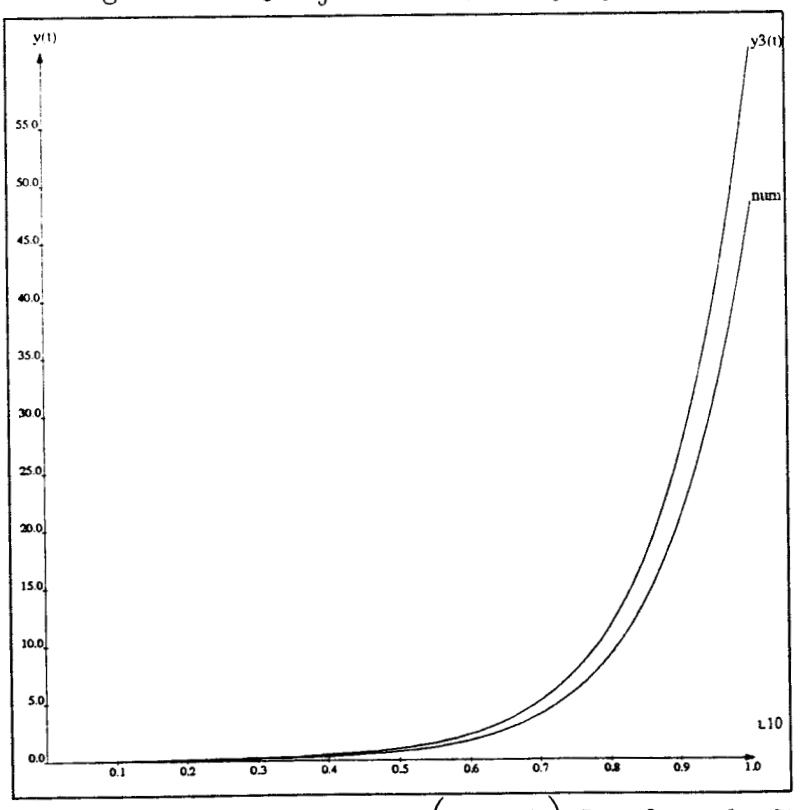


Figure 3.32 : Sortie de $\dot{y} + y = b_3$, $y_3 = \left(\frac{1}{15}t - \frac{1}{90}\right)e^{7t} + \frac{8}{45}e^{t} - \frac{1}{6}e^{-5t}$

1. Evaluation de S pour l'entrée b_1 :

$$y_1 = \Big(-\frac{1}{2}B_0^1(t) + \frac{1}{2}B_1^1(t)\Big)e^{3t} - e^{2t} + \frac{3}{2}e^t$$

soit en base canonique :

$$y_1 = \Big(t - \frac{1}{2}\Big)e^{3t} - e^{2t} + \frac{3}{2}e^t$$

représentée sur la figure 3.28

2. Evaluation de S pour l'entrée b_2 :

$$y_2 = \Big(-\frac{1}{90}B_0^1(t) + \frac{1}{18}B_1^1(t)\Big)e^{7t} + \frac{1}{2}B_0^1(t) - \frac{1}{2}B_1^1(t)e^{3t} + e^{2t} - \frac{119}{90}e^t - \frac{1}{6}e^{-5t}$$

soit en base canonique :

$$y_2 = \frac{1}{15}t - \frac{1}{90}e^{7t} + \Big(-t + \frac{1}{2}\Big)e^{3t} + e^{2t} - \frac{119}{90}e^t - \frac{1}{6}e^{-5t}$$

représentée sur la figure 3.30

3. Evaluation de S pour l'entrée b_3 :

$$y_3 = \Big(-\frac{1}{90}B_0^1(t) + \frac{1}{18}B_1^1(t)\Big)e^{7t} + \frac{8}{45}e^t - \frac{1}{6}e^{-5t}$$

soit en base canonique :

$$y_3 = \Big(\frac{1}{15}t - \frac{1}{90}\Big)e^{7t} + \frac{8}{45}e^t - \frac{1}{6}e^{-5t}$$

représentée sur la figure 3.32

Sur les simulations des sorties figurent également les simulations numériques obtenues par la méthode de Runge-Kutta. Les tableaux des temps de calculs pour ces trois entrées nous montrent l'intérêt de la méthode formelle (évaluation exacte de la fraction rationnelle $z_b z_a^*$) par rapport à la simulation numérique : pour des temps de calcul équivalents, la méthode formelle nous donne ici, en plus de la courbe, l'expression symbolique exacte de la sortie, ce qui est impossible par la méthode numérique. De plus, les figures 3.28, 3.30 et 3.32 nous montrent que la méthode numérique est d'autant moins satisfaisante qu'elle génère une sortie qui ne colle pas exactement au résultat, que nous savons exact, fourni par l'évaluation de la fraction rationnelle.

Temps de calculs des courbes
pour l'entrée polynomiale exponentielle

$$b_1(t) = 2te^{3t} - e^{2t}$$

	Calcul de l'expression symbolique	Nombre de points calculés	Calcul des points de la courbe
Evaluation de la fraction rationnelle	0.1 sec	65	0.9 sec
Méthode numérique de Runge-Kutta	-	65	0.667 sec

Temps de calculs des courbes
pour l'entrée polynomiale exponentielle

$$b_2(t) = \tfrac{2}{5}te^{7t} - 2te^{3t} + e^{2t} + e^{-5t}$$

	Calcul de l'expression symbolique	Nombre de points calculés	Calcul des points de la courbe
Evaluation de la fraction rationnelle	0.1 sec	65	1.0 sec
Méthode numérique de Runge-Kutta	-	65	0.9 sec

Temps de calculs des courbes
pour l'entrée polynomiale exponentielle

$$b_3(t) = \frac{2}{5}te^{7t} + e^{-5t}$$

	Calcul de l'expression symbolique	Nombre de points calculés	Calcul des points de la courbe
Evaluation de la fraction rationnelle	0.267 sec	65	0.867 sec
Méthode numérique de Runge-Kutta	-	65	0.6 sec

3.5.4.4 Exemple 4

Nous reprenons ici l'équation différentielle de l'exemple précédent : $\dot{y} + a(t)y = b(t)$, avec $a(t) = 1$, que nous codons par la série génératrice $z_b z_a^*$

Nous traitons cet exemple pour deux nouvelles entrées :

$$b_4(t) = 42t^3 - 63t^2 + 27t - 3 ,$$
$$b_5(t) = -e^{2t} + 1 + 3e^{-t}$$

Les représentations graphiques des évaluations de $z_b z_a^*$ pour $b_4(t)$ et $b_5(t)$, $t \in [0,1]$ sont respectivement données sur les figures 3.33 et 3.34 page 136 pour des méthodes de simulation par fractions rationnelles, par approximations polynomiales et par calcul numérique. On remarque sur ces figures la confirmation de la convergence uniforme ([22, 23, 24, 64, 27], sur un intervalle de temps donné, des approximations polynomiales successives. Nous nous attachons, pour ces deux entrées, à comparer les performances de rapidité et de précision des trois méthodes de simulation utilisées.

Pour cela, nous donnons ici des tableaux récapitulatifs des valeurs numériques fournies par Scratchpad à des intervalles de temps réguliers (voir les tableaux donnés aux pages 137 et 139) ainsi que des tableaux comparatifs des temps de calculs des points de la courbe (voir le tableau en page 138).

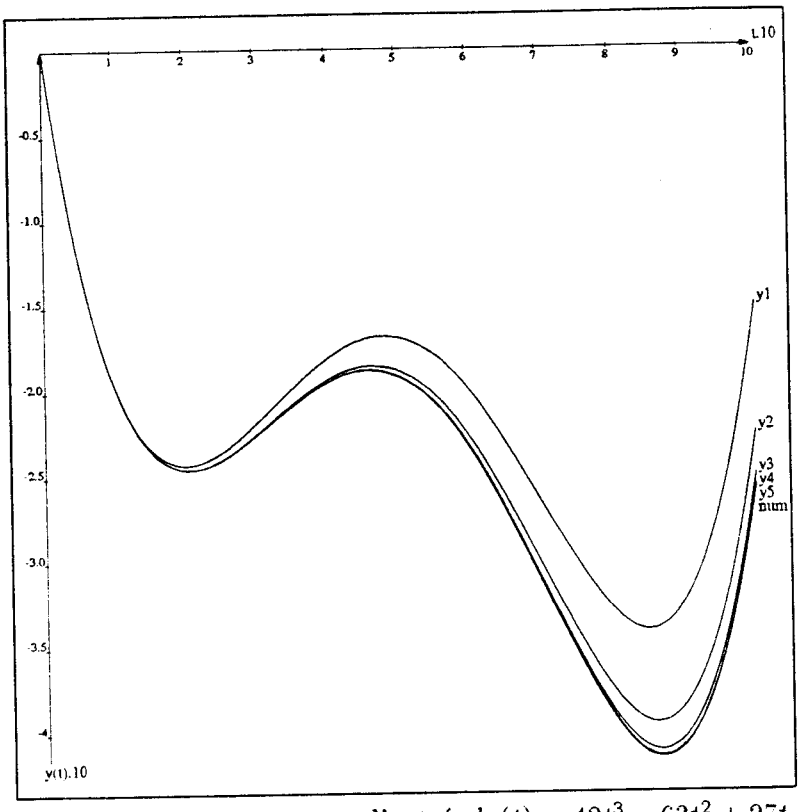


Figure 3.33 : Evaluation de $z_b z_a^*$ pour l'entrée $b_4(t) = 42t^3 - 63t^2 + 27t - 3$

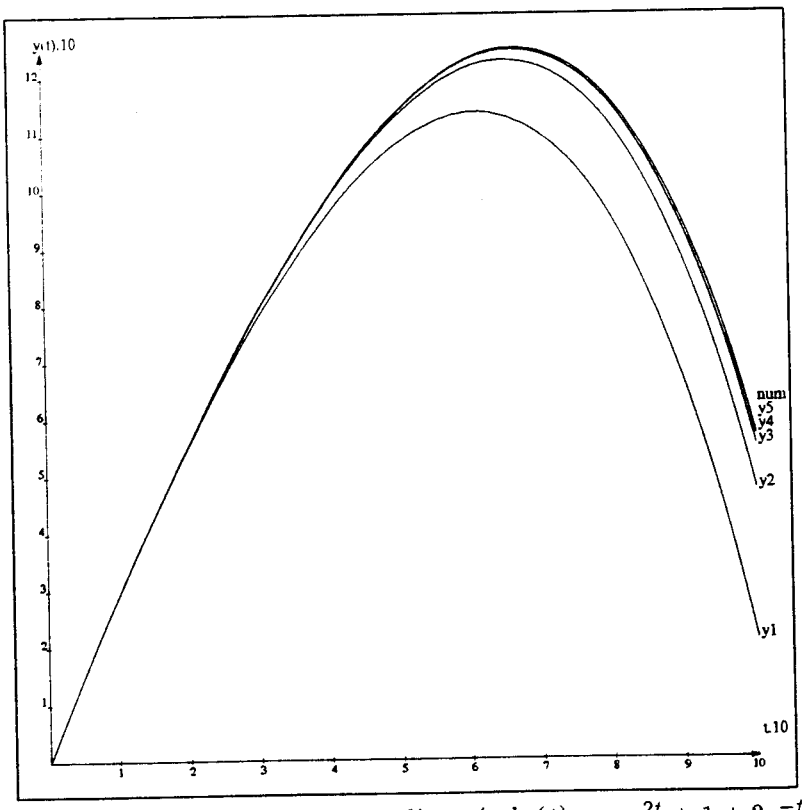


Figure 3.34 : Evaluation de $z_b z_a^*$ pour l'entrée $b_5(t) = -e^{2t} + 1 + 3e^{-t}$

Tableau récapitulatif des résultats pour les différentes méthodes de simulation pour l'entrée polynomiale $b_4(t) = 42t^3 - 63t^2 + 27t - 3$

	Evaluation de la fraction rationnelle $z_b z_a^*$	Méthode numérique de Runge-Kutta	Evaluation de l'approximation Polynomiale de degré 5 $z_b(1 + z_a + z_a^2 + z_a^3 + z_a^4)$	Evaluation de l'approximation Polynomiale de degré 4 $z_b(1 + z_a + z_a^2 + z_a^3)$	Evaluation de l'approximation Polynomiale de degré 3 $z_b(1 + z_a + z_a^2)$
	$150e^t - 42t^3 - 63t^2 - 153t - 150$	?	$\frac{1}{160}t^8 + \frac{1}{40}t^7 + \frac{17}{80}t^6 + \frac{5}{4}t^5 + \frac{25}{4}t^4 - 17t^3 + 12t^2 - 3t$	$\frac{1}{20}t^7 + \frac{7}{40}t^6 + \frac{51}{40}t^5 + \frac{25}{4}t^4 - 17t^3 + 12t^2 - 3t$	$\frac{7}{20}t^6 + \frac{21}{20}t^5 + \frac{51}{8}t^4 - 17t^3 + 12t^2 - 3t$
t	$y_5(t)$	$y_6(t)$	$y_4(t)$	$y_3(t)$	$y_2(t)$
1/16	- 0.14466858	- 0.14467883	- 0.1446788	- 0.1446788	- 0.14467709
1/8	- 0.21914673	- 0.2191383	- 0.21913826	- 0.21913764	- 0.21911335
3/16	- 0.24465942	- 0.24466181	- 0.24466164	- 0.2446573	- 0.24454775
1/4	- 0.2399292	- 0.2399375	- 0.23993674	- 0.23992005	- 0.23961183
5/16	- 0.22088623	- 0.22089088	- 0.22088815	- 0.22084185	- 0.2201719
3/8	- 0.20050049	- 0.20050657	- 0.20049903	- 0.20039429	- 0.19915639
7/16	- 0.18864441	- 0.18863882	- 0.18862107	- 0.188415	- 0.18636808
1/2	- 0.19181824	- 0.19180942	- 0.19177248	- 0.19140625	- 0.18828125
9/16	- 0.21298218	- 0.21299289	- 0.21292286	- 0.21232006	- 0.20782402
5/8	- 0.2513733	- 0.2513877	- 0.2512642	- 0.25032935	- 0.2441454
11/16	- 0.30218506	- 0.3021726	- 0.3019672	- 0.30058423	- 0.29236752
3/4	- 0.3562317	- 0.35624757	- 0.35592183	- 0.35395202	- 0.20782402
13/16	- 0.39996338	- 0.3999578	- 0.399461	- 0.39674056	- 0.38327587
7/8	- 0.41479492	- 0.41479975	- 0.4140665	- 0.41040373	- 0.39363116
15/16	- 0.37713623	- 0.37710878	- 0.3760561	- 0.3712282	- 0.35062438
1	- 0.25775146	- 0.25772586	- 0.25625002	- 0.25	- 0.22500001

Sur ce tableau, les valeurs de référence sont données par l'évaluation de la fraction rationnelle $z_b z_a^*$: l'expression de la sortie y_5 est ici obtenue sans approximation.

On remarque alors la précision (sur l'intervalle de temps $[0, 1]$) des méthodes de Runge-Kutta et de l'approximation polynomiale de degré 5. La qualité des approximations polynomiales est déjà suffisante pour le tracé, cependant le tableau des temps de calcul nous montre qu'elle sont peu rentable : pour un temps de calcul plus élevé que les autres méthodes de simulation (due à la non utilisation de la convolution et au degré élevé des polynômes à traiter), nous obtenons une expression symbolique approchée, c'est à dire valable uniquement sur l'intervalle de temps considéré. Le calcul numérique demeure ici le plus rapide, il ne peut cependant pas donner l'expression symbolique exacte de la sortie. La simulation par fractions rationnelles avec utilisation du théorème de convolution (voir [28]) est par conséquent la méthode la plus intéressante tant au point de vue rapidité de calcul qu'au point de vue fiabilité des résultats.

Temps de calcul de 129 points de la courbe de sortie					
Entrée	Evaluation rationnelle	Runge-Kutta	polynôme de degré 4	polynôme de degré 3	polynôme de degré 2
$42t^3 - 63t^2 + 27t - 3$	2.6 sec.	1.0 sec.	4.033 sec.	3.667 sec.	3.2 sec.
$-e^{2t} + 1 + 3e^{-t}$	2.667 sec.	2.6 sec.	5.833 sec.	4.1 sec.	3.85 sec.

Pour le tracé des différentes sorties, les polynômes sont exprimées dans la base de Bernstein :

$$y_5 = 150\ e^t - 150\ B_0^3 t - 201\ B_1^3 t - 273\ B_2^3 t - 408\ B_3^3 t$$

$$y_4 = -\frac{3}{8} B_1^8 t - \frac{9}{28} B_2^8 t - \frac{1}{7} B_3^8 t - \frac{3}{56} B_4^8 t - \frac{5}{32} B_5^8 t - \frac{923}{2240} B_6^8 t - \frac{3}{5} B_7^8 t - \frac{41}{160} B_8^8 t$$

$$y_3 = -\frac{3}{7} B_1^7 t - \frac{2}{7} B_2^7 t - \frac{2}{35} B_3^7 t - \frac{1}{20} B_4^7 t - \frac{93}{280} B_5^7 t - \frac{181}{280} B_6^7 t - \frac{1}{4} B_7^7 t$$

$$y_2 = -\frac{1}{2} B_1^6 t - \frac{1}{5} B_2^6 t + \frac{1}{20} B_3^6 t - \frac{7}{40} B_4^6 t - \frac{7}{10} B_5^6 t - \frac{9}{40} B_6^6 t$$

$$y_1 = -\frac{3}{5} B_1^5 t + \frac{3}{20} B_3^5 t - \frac{3}{4} B_4^5 t - \frac{3}{20} B_5^5 t$$

Points calculés par les différentes méthodes pour l'entrée polynomiale exponentielle $b_5(t) = -e^{2t} + 1 + 3e^{-t}$					
	Evaluation de la fraction rationnelle	Méthode numérique de Runge-Kutta	Approximation Polynomiale de degré 4	Approximation Polynomiale de degré 3	Approximation Polynomiale de degré 2
	$z_b z_a^*$		$z_b(1 + z_a + z_a^2 + z_a^3 + z_a^4)$	$z_b(1 + z_a + z_a^2 + z_a^3)$	$z_b(1 + z_a + z_a^2)$
	$-e^{2t} + \frac{7}{2}e^t - 1 - \frac{3}{2}e^{-t}$	?	$-\frac{31}{32}e^{2t} + \frac{1}{120}t^5 + \frac{3}{16}t^4 + \frac{7}{24}t^3 + \frac{39}{16}t^2 + \frac{31}{16}t + \frac{127}{32} - 3e^{-t}$	$-\frac{15}{16}e^{2t} + \frac{1}{24}t^4 + \frac{3}{4}t^3 + \frac{7}{8}t^2 + \frac{39}{8}t + \frac{15}{16}$	$-\frac{7}{8}e^{2t} + \frac{1}{6}t^3 + \frac{9}{4}t^2 + \frac{7}{4}t + \frac{31}{8} - 3\%^{-t}$
t	$y_5(t)$	$y_6(t)$	$y_4(t)$	$y_3(t$	$y_2(t)$
1/16	0.18346262	0.18346258	0.18346047	0.18346238	0.18346047
1/8	0.35824895	0.35824883	0.35821843	0.35824788	0.35821843
3/16	0.52327096	0.52327085	0.52312016	0.523265	0.52312016
1/4	0.6771667	0.6771665	0.6766956	0.67714274	0.6766956
5/16	0.8182632	0.81826335	0.81712544	0.8181915	0.81712544
3/8	0.9445362	0.9445361	0.9422023	0.94435906	0.9422023
7/16	1.0535579	1.053558	1.0492842	1.053179	1.0492842
1/2	1.1424463	1.1424468	1.1352444	1.1417148	1.1352444
9/16	1.2078004	1.2078004	1.1964133	1.2064948	1.1964133
5/8	1.2456257	1.245626	1.2285113	1.2434385	1.2285113
11/16	1.2512574	1.2512572	1.2265728	1.2477741	1.2265728
3/4	1.2192616	1.2192614	1.1848598	1.213944	1.1848598
13/16	1.143332	1.1433321	1.0967631	1.1354976	1.0967631
7/8	1.0161672	1.0161681	0.9546962	1.004972	0.9546962
15/16	0.8293352	0.8293358	0.74995947	0.81375504	0.74995947
1	0.57311106	0.5731114	0.47260332	0.55192614	0.47260332

3.6 Scratchpad et le calcul numérique

Après avoir développé des méthodes symboliques d'évaluation et de simulation, il s'avère intéressant de les comparer aux méthodes numériques en cours depuis longtemps. Nous avons choisi pour cela de simuler le comportement des systèmes dynamiques en utilisant les algorithmes d'intégration numérique, toujours très utilisés, de *Runge-Kutta* et *Adams* dont la fiabilité des résultats n'est plus à prouver : les erreurs de troncature et de méthode, ainsi que les conditions de stabiblité sont parfaitement connues.

Ceci nous permet tout d'abord de vérifier la validité d'approximations polynomiales, on peut pour cela se rapporter aux exemples développés en 3.5.4.1 page 115 et 3.5.4.2 page 126. Nous obtenons ainsi le degré de la série pour lequel l'approximation est confondue avec la courbe réelle, sur un intervalle compact donné. Nous pouvons ainsi connaitre une expression symbolique représentative de l'expression réelle sur l'intervalle de temps étudié.

D'autre part, nous avons vu sur les exemples développé en 3.5.4.3 page 129 et 3.5.4.4 page

135, que l'on sait, pour certaines séries, calculer l'expression exacte de la sortie, nous pouvons alors vérifier la validité du calcul numérique (voir par exemple les figures 3.28 page 130,3.30 page 131 et 3.32 page 132).

On peut alors remarquer que, pour des problèmes de stabilité numérique des algorithmes de *Runge-Kutta* et *Adams*, il est en général nécessaire de calculer beaucoup plus de points que n'en exige la représentation graphique.

Notre implantation en Scratchpad d'algorithmes d'intégration numérique possède un avantage indéniable au niveau des facilités d'utilisation : on exécute un calcul numérique en utilisant la bibliothèque Scratchpad, exactement comme les autres fonctions. Les algorithmes que nous développons ainsi restent cependant limités à un très petit nombre de fonctions : celles correspondant à nos propres besoins. Avec la commercialisation d'*Axiom* par *NAG*, on peut espérer la création d'une interface agréable permettant, en toute simplicité, un échange d'informations entre les fonctions numériques de la bibliothèque Fortran et celles de la bibliothéque Scratchpad.

La version de Scratchpad dont nous disposons ne permet cependant pas une interaction simple avec les logiciels performants de calcul numérique. Ceci nous a amené à développer nos propres programmes directement en Scratchpad. Le but n'est pas de réécrire les programmes existants, mais de disposer d'un outil permettant la comparaison sur un même graphique (dans une seule fenêtre) des solutions algébriques et numériques.

Ce dernier travail est d'autant plus utile qu'il permet de déterminer un mode d'échange d'informations entre le calcul formel et le calcul numérique, qui peut servir de base ou de référence pour la mise en place d'une interface réellement efficace entre *Axiom* et une bibliothèque de calcul numérique telle que celle développée par *NAG*.

Références

[1] **J. Berstel, C. Reutenauer.** Rational series and their langages. *Springer-Verlag 1984*

[2] **S. Baccar, G. Seignier, F. Lamnabhi-Lagarrigue.** Utilisation du calcul formel pour la modélisation et la simulation des circuits électroniques faiblement non linéaires. *Ann. Telecommun. 46, num. 5-6. p. 282-288. 1991*

[3] **P. Bézier.** Procédé de définition numérique des courbes et surfaces non mathématiques. *Système Unisurf, Automatisme, 13 1968*

[4] **J.L. Boulanger.** Etude de faisabilité d'un compilateur pour Scratchpad II. *mémoire de D.E.A. de l'Université de Lille 1991*

[5] **N. Bourbaki.** Groupes et algèbres de Lie. *Hermann 1972*

[6] **F. Boussemart, V. Hoang Ngoc Minh.** Simulation graphique du comportement entrée/sortie des systèmes dynamiques. *Formal power series and algebraic combinatorics. Bordeaux mai 1991*

[7] **F. Boussemart, V. Hoang Ngoc Minh.** Graphic simulation of nonlinear control systems behaviour in Scratchpad. *IMACS Symposium MCTS 1991*

[8] **B. Carré.** Méthodologie orientée objet pour la représentation des connaissances. Concepts de point de vue, de représentation multiple et évolutive d'objets. *Thèse de doctorat de l'Université de Lille 1989*

[9] **B.W. Char, G.J.Fee, K.O. Geddes, G.H. Gonnet, M.B. Monagan.** A tutorial introduction to Mapple. *Journal of Symbolic Computation. Vol 2. 1986, p. 179-200*

[10] **S. Dalmas.** Un langage fonctionnel polymorphe Application aux problèmes logiciels du calcul formel. *Thèse de doctorat de l'université de Nice 1991*

[11] **P. De Casteljau.** Outillage, méthode de calcul. *André Citroën Automobiles, S.A., Paris 1959*

[12] **P. De Casteljau.** Formes à pôles. *Mathématiques et C.A.O., Hermès 1985*

[13] **M. Delest.** Enumeration of polynominoes using Macsyma. *Theoret. Comput. Sci. vol. 79 p 209-227 1991*

[14] **G. Duchamp.** Orthogonal projection onto the free Lie algebra. *Theoret. Comput. Sci. vol. 79 p 241-257 1991*

[15] **J.P.Duval.** Génération d'une section des classes de conjugaison et arbre des mots de Lyndon de longueur bornée. *Theoret. Comput. Sci., 60, p.255-283, 1988.*

[16] **R.T. Farouki, V.T. Rajan.** Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design 1988*

[17] **R.T. Farouki, V.T. Rajan.** On the numérical condition of polynomials in Bernstein form. *Computer Aided Geometric Design 1987*

[18] **J.C. Fiorot.** Courbes Bézier. *Journées courbes et surfaces Bézier, B-splines, ATP Mathématiques, informatiques et applications. Thème Outils Mathématiques et Informatiques edes modèles géométriques. Rennes 11,13 mars 1987*

[19] **J.C. Fiorot, P. Jeannin.** Courbes et Surfaces Rationnelles. Applications à la C.A.O. *collection RMA(12), Masson, Paris 1989*

[20] **J.C. Fiorot, P. Jeannin, S. Taleb** B-Rational curves and Reparametrization : the Quadratic Case. *Soumis à publication*

[21] **J. Fitch.** Solving algebraic problems with Reduce. *Journal of Symbolic Computation. Vol 1. 1985, p. 211-227*

[22] **M. Fliess.** Séries de Volterra et séries formelles non commutatives. *C.R. académie des sciences Paris 1975*

[23] **M. Fliess.** Un outil algébrique; les séries formelles non commutatives. *Notes Econom. Math. Syst. 131, p. 122-148 Springer-Verlag Berlin 1976*

[24] **M. Fliess.** Fonctionnelles causales non linéaires et indéterminées non commutatives. *Bull. soc. math. France, 109 1981*

[25] **M. Fliess, M. Lamnabhi, F. Lamnabhi-Lagarrigue.** An algebraic approach to nonlinear functional expansions. *IEEE Trans. Circ. Syst., CAS-30, 1983, pp. 554-570*

[26] **C. Guyon, G. Jacob, M. Petitot** Motion Planning and Symbolic Computation. *Rapport technique du L.I.F.L. No 1992*

[27] **C.Hespel, G.Jacob.** Calcul des approximations locales bilinéaires de systèmes analytiques. *APII, 23, p.331-349 1989*

[28] **V. Hoang Ngoc Minh.** Evaluation transform with kernel function. *à paraître dans "Algebraic and Computing Treatment of Noncommutative power series" G. Duchamp, D. Krob & G. Jacob ed., Theoret. Comput. Sci. special issue, 1992*

[29] **V. Hoang Ngoc Minh.** Contribution au développement d'outils informatiques pour résoudre des problèmes d'automatique nonlinéaire. *Thèse de doctorat de l'université de Lille 1990*

[30] **V. Hoang Ngoc Minh, G.Jacob, N.E. Oussous.** Comportement entrée/sortie des systèmes analytiques non linéaires : approximations rationnelles, approximations structurelles nilpotentes. *Analysis of Controlled dynamical Systems, Lyon, Juillet 1990*

[31] **G. Jacob.** Lyndon discretization and exact motion planning. *European Control Conference Grenoble 1991*

[32] **G. Jacob, N.E. Oussous.** Local and minimal realization of nonlinear dynamical systems. *IFAC symposium, Capri 1989*

[33] **G.Jacob et N.Oussous.**. Mots de Lyndon et bases associées : traitement en Macsyma et applications, *Publication du LIFL Lille, IT-204, 1991.*

[34] **G.Jacob, N.E.Oussous and M.Petitot**. The Scratchpad implementation of the minimal analytic realization, *"IMACS SYMPOSIUM MCTS", Villeneuve d'Ascq-France, Mai 1991.*

[35] **A. Kaufmann.** Algorithmes distribués pour l'intersection de courbes et surfaces de Bézier. *Thèse de doctorat de l'université de Grenoble 1990*

[36] **P.V. Koseleff.** Jeux de mots dans les algèbres de Lie libres : quelques bases et formules. *Theoret. Comput. Sci. vol. 79 p 241-257 1991*

[37] **D. Krob**. Some examples of formal series used innon-commutative algebra. *Theoret. Comput. Sci. vol. 79 p 111-137 1991*

[38] **F. Lamnabhi-Lagarrigue.** Séries de Volterra et commande optimale singulière. *Thèse d'état de l'Université de Paris sud 1985*

[39] **F. Lamnabhi-Lagarrigue, P. Leroux, X.G. Viennot.** Combinatorial approximations of Volterra Series by bilinear systems. *Analysis of controlled dynamical systems. Ed. Bonnard, Bridc, Gauthier et Kupka. Birkaüser Boston. 1991.*

[40] **M. Lamnabhi.** A new symbolic calculus for the response of nonlinear systems. *Systems & Control Letters. Vol.2 Num.3 p.154-163 1982*

[41] **M. Lamnabhi.** Functional analysis of nonlinear circuits : a generating power series approach. *J.E.E. Proceedings vol 133 num.5 p 375-384 1986*

[42] **M. Lamnabhi.** Séries de Volterra et Séries génératrices non commutatives. *Thèse de Docteur Ingénieur. Université de Paris XI, Orsay. 1980.*

[43] **M. Lamnabhi.** Analyse des systèmes non linéaires par les méthodes de développements fonctionnels. *Thèse d'Etat. Université de Paris XI, Orsay. 1986.*

[44] **D. Marchepoil, P. Chenin.** Algorithmes de recherche de zéros d'une fonction de Bézier. *Rapport de recherche du LMC-Imag 1990*

[45] **A. Martin.** Calcul d'approximations de la solution d'un système non linéaire utilisant le logiciel Scratchpad. *Algebraic Computing in Control. Ed G. Jacob et F. Lamnabhi-Lagarrigue. Lec. Notes Contr. Inf. Sci. num. 165 1991. Springer-Verlag.*

[46] **G. Melançon, C. Reteunauer.** Lyndon words, free algebras and shuffles. *Publication du LITP et de l'UQAM Québec 1980*

[47] **Lafferriere, H.J. Sussmann.** Motion planning for controllable systems without drift : a preliminary report. *Rapport Sycon 1990*

[48] **J.M. Lane, R.F. Riesenfeld.** A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and machine Intelligence 2, p. 35-46 1980*

[49] **B. Leguy.** Ada, guide d'utilisation. *Eyrolles 1989*

[50] **M. Lothaire.** Combinatorics on words. *Reading Massachusetts 1983*

[51] **R. Ogor, R. Rannou.** Langage Ada et Algorithmique. *Hermès 1990*

[52] **N.E. Oussous.** Etude et traitement des séries formelles non commutatives pour la représentation minimale des systèmes dynamiques non linéaires. *Thèse de doctorat de l'université de Lille 1988*

[53] **N.E. Oussous, M. Petitot.** Polynômes non commutatifs: représentation et traitement par les systèmes de calcul formel. *Formal power series and algebraic combinatorics. Bordeaux mai 1991*

[54] **N.E.Oussous and M.Petitot.** Scratchpad implementation of the local minimal realization of dynamic systems, *ACC "Algebraic Computing in Control", Paris-France, Mars 1991.*

[55] **R. Pavelle, P.S. Wang.** Macsyma from F to G. *Journal of Symbolic Computation. Vol 1. 1985, p. 69-100*

[56] **M. Petitot.** Algebre non commutative en Scratchpad : application au problème de la réalisation minimale analytique. *Thèse de doctorat de l'université de Lille 1992*

[57] **C. Reteunauer.** Séries rationnelles et algèbres syntaxiques. *Thèse d'état de l'Université de Paris VI 1980*

[58] **W.J. Rugh.** Nonlinear system theory. *Baltimore : The Johns Hopkins Univ. Press. 1981.*

[59] **P.Sablonnière.** Bases de Bernstein et approximants splines. *Thèse d'état, Lille, 1982*

[60] **M. Schetzen.** The Volterra and Wiener theories of nonlinear systems. *Wiley. New York. 1980*

[61] **E.D.Sontag.** Bilinear realizability is equavalent to existence of a singular affine differential i/o equation. *Systems and Control Letters 11 1988, p. 181-187*

[62] **E.D.Sontag, Y. Wang.** Realization and Input/Output Relation : the Analytic case. *proceeding of C.D.C., December 1989*

[63] **S. Steinberg, P.J. Roache.** Using Macsyma to write Fortran subroutines. *Journal of Symbolic Computation. Vol 2. 1986, p. 213-216*

[64] **H.J. Sussmann.** Semigroup representations, bilinear approximation of input-output maps and generalized inputs. *Mathematical Systems Theory. Lect. notes econom. math. syst. 131, p. 172-191 Springer-Verlag Berlin 1976*

[65] **H.J. Sussman.** Two new methods for motion planning for controllable systems without drift. *European Control Conference Grenoble 1991*

[66] **S. Taleb.** Incidence du paramétrage sur le contrôle des courbes rationnelles. *Thèse de doctorat de l'université de Lille, 1991*

[67] **X.G. Viennot.**. Algèbres de Lie Libres et Monoïdes Libres, *Lecture Notes In Mathematics, Springer-Verlag. 691,1978.*

A

Les domaines relatifs aux courbes de Bézier

A.1 Les polynômes de Bernstein

```
)abb domain UBP UnivariateBernsteinPolynomial

UnivariateBernsteinPolynomial(t:Expression,R:Ring): pub == priv where

  NNI           ==> NonNegativeInteger
  I             ==> Integer
  E             ==> Expression
  L             ==> List
  C             ==> IntegerCombinatoricFunctions
  UP            ==> UnivariatePoly(t,R)

  pub == Join(UnivPolyCat R,RetractableTo R,DifferentialRing) with

        bdegree       :    $          -> NNI
        b             :    (NNI,NNI)  -> $
        listCoef      :    $          -> L R
        if R has Field then
          elevate     :   ($,NNI)     -> $
          ajust       :   ($,NNI)     -> $
          integrate   :    $          -> $
          listeP      :  ($,I)        -> L $
        eval          :    ($,R)      -> R
        listsub       :    ($,R)      -> L L R
        bpoly         :    L R        -> $
        coerce        :    UP         -> $
        coerce        :    $          -> UP
        bform         :   (UP,R,R)    -> $
        minform       :    $          -> $

  priv == add

-- REPRESENTATION EN MEMOIRE
```

```
   Rep := L R

-- DOMAINES VISIBLES

   import PrintableForm
   import OutputForm

-- FONCTIONS DE BASE

   b(i,n) ==
             i > n => error "does not exist"
             n = 0 => []
             [if i = j then 1 else 0 for j in 0..n]

   bdegree p ==
             null(p) => 0
             MAXINDEX(p)$Lisp

   coef(p,n) ==
             null(p) => 0
             p.n

   coerce(c:R):$ ==
             c = 0$R => []
             [c]

   Zero == []

   One == [1$R]

   (k:R) * (p:$) ==
             k = 0 => []
             [k *$R c for c in p]

   (n:I) * (p:$) ==
             n = 0 => []
             [n *$R c for c in p]

   listCoef(p) == p:L R

   lc p  == p.first

   red p == p.rest

   varPol == [0,1]

   var == t
```

```
-- EVALUATION EN UN POINT PAR L ALGO DE DE CASTELJAU

   difdiv:($,R) -> $
   difdiv(p,k) ==
             [(e1+e2)*k for e1 in p for e2 in p.rest]

   decast:($,R) -> R
   decast(p,k) ==
             #p =1  => lc p
             decast( difdiv(p,k), k)

   eval(p,k) ==
             null p => 0$R
             k =0   => p.first
             k = 1  => p.last
             decast(p,k)

-- ALGORITHME DE SUBDIVISION

   listsub(p,k) ==
             null p => [[]]$L(L(R))
             #p =1 => [listCoef p]
             cons( listCoef p,listsub( difdiv(p,k),k ) )

   if R has Field then
     listeP(p,nbsub) ==
             nbsub = 0 => [p]
             lpt1 := listsub( p,inv(2::R)$R )
             pol1 := [l.first for l in lpt1] :: $
             pol2 := [l.last for l in reverse lpt1] :: $
             [ :listeP(pol1,nbsub-1), :listeP(pol2,nbsub-1)]

-- CONVERSION EN EXPRESSION

   outTerm : (R,NNI,NNI) -> E
   outTerm(k,i,n) ==
             e := mkUnary(scripts(form("B"),
                                  [form(i::E),
                                  form(n::E)])::E,t)
             k=1 => e
             formTimes(k::E,e)

   coerce(p:$):E ==
             null(p) => "0"::E
             n := bdegree(p)
             n = 0 => p.0 :: E
             l1: L E := [k::E for k in p | not(k=0)]
             null(l1)$L(E) => "0"::E
             l2 := [outTerm(p.i,i,n) for i in 0..n | not(p.i = 0)]
```

```
            formPlus 12

-- FONCTIONS UTILES POUR LA MANIPULATION DE POLYNOMES DANS LA BASE DE BERNSTEIN

   if R has Field then

     incremente: $ -> $
     incremente(p) ==
               n := bdegree p
               append(cons(p.0,
                      [((i*p.(i-1)+(n+1-i)*p.i)) / (n+1)::R for i in 1..n]),
                               [p.n])

     elevate(p,n) ==
                null(p) => elevate([0],n)
                n = 0 => p
                elevate(incremente(p),(n-1)::NNI)

     ajust(p,n) ==
                null(p) => ajust([0],n)
                elevate(p,(n - bdegree p )::NNI)

-- ARITHMETIQUE

     p:$ + q:$ ==
                null(p) => q
                null(q) => p
                n := max(bdegree p,bdegree q)
                p1 := ajust(p,n)
                q1 := ajust(q,n)
                [e1 + e2 for e1 in p1 for e2 in q1]

     p:$ - q:$ ==
               null p => -q
               null q => p
               n := max(bdegree p,bdegree q)
               p1 := ajust(p,n)
               q1 := ajust(q,n)
               [e1 - e2 for e1 in p1 for e2 in q1]


     p:$ / c:R ==
               null p => []
               [e/c for e in p]

     integ(p:$,base:R,n:R):$ ==
               q:R := base + (p.first / n)
               #p = 1 => [q]
               cons(q,integ(p.rest,q,n))
```

```
    integrate(p:$) ==
              null p => []
              n := bdegree p
              cons(0,integ(p,0$R,(n+1)::R))

    coefk:($,$,I) -> R
    coefk(p,q,k) ==
              m := bdegree p
              n := bdegree q
              r:R := 0$R
              for j in max(0,k-n)..min(m,k) repeat
                        b1 := binomial(m,j)$C ::R
                        b2 := binomial(n,k-j)$C ::R
                        b3 := binomial(m+n,k)$C ::R
                        f := (b1*b2)/b3
                        r := r + (f * p.(j::NNI) * q.((k-j)::NNI))
              r

    p:$ * q:$ ==
              null p => 0
              null q => 0
              n := bdegree p
              m := bdegree q
              [coefk(p,q,i) for i in 0..(n+m)]

  der(p:$):$ ==
              #p = 2 => [p.1 - p.0]
              cons(p.1 - p.0, der p.rest)

  deriv p ==
              #p < 2 => []
              (q:= der p) = 0 => []
              bdegree(p) * q

  - p:$ ==    [-c for c in p]

-- CONVERSION DES POLYNOMES

-- Fonctions locales

  if R has Field then

    firstcol:UP -> L R
    firstcol(pol) ==
              n := degree pol
              [coef(pol,k)/binomial(n,k)$C::R for k in 0..n]

    delta:L R -> L R
```

```
    delta(l) == [e1 + e2 for e1 in l for e2 in rest l]

    allcol:L R -> L L R
    allcol(l) ==
              # l = 1 => [l]
              [l,:allcol(delta(l))]

-- Fonctions exportees

    coerce(pol:UP) ==
              pol = 0$UP => []
              ll:L L R := allcol firstcol pol
              bpoly [e.first for e in ll]

    coerce(p:$) ==
              null p => 0$UP
              n := bdegree p
              l := [coef(p,i)::UP for i in 0..n]
              for j in 1..n repeat
                 for i in 0..(n-j) repeat
                      l.i := l.i * (1-varPol()$UP) +
                                   l.(i+1) * varPol()$UP
              l.0

    bform(p,debut,fin) ==
              pr:UP := (fin-debut)*varPol()$UP + debut :: UP
              p := eval(p,pr)$UP
              p::$

    minform p == p::UP::$

    p:$ = q:$ == p::UP =$UP q::UP

    degree p == degree(p::UP)$UP

  bpoly(l) == l:$
```

A.2 Les courbes de Bézier

```
)abb domain BC BCurve

 BCurve(t:Expression,F:Field): pub == priv where

    UBP ==> UnivariateBernsteinPolynomial(t,F)
    L ==> List
    I ==> Integer
    NNI ==> NonNegativeInteger
```

```
E ==> Expression
UP ==> UnivariatePoly(t,F)
VM ==> VecteurMassique(F)

pub == with

  coerce : L L F -> $
  coerce : $ -> L L F
  coerce : $ -> E
  coerce : L UBP -> $
  coerce : $ -> L UBP
  coerce : L UP -> $
  coerce : $ -> L UP

priv == add

  Rep := L L F

  import UBP
  import PrintableForm
  import OutputForm

  coerce(c:L L F):$ == c:$

  coerce(bc:$):L L F == bc:L L F

  outTerm(k:L F,i:I,n:I):E ==
        e := mkUnary(scripts(form("B"),[form(i::E),form(n::E)])::E,t)
        v := matrix [[form(h::E)] for h in k]
        formTimes(v::E,e)

  coerce(c:$):E ==
        n := maxIndex c
        l2 := [outTerm(c.i,i,n) for i in 0..n]
        formPlus l2

  coerce(l: L UBP):$ ==
        n := 0$NNI
        for p in l repeat n := max(n,bdegree p)
        lp := [ajust (p,n) for p in l]
        ll := [listCoef p for p in lp]
        [[ll.i.j for i in 0..maxIndex(ll)] for j in 0..n]

  coerce(bc:$):L UBP ==
        l:L L F:=[[bc.i.j for i in 0..maxIndex(bc)]
                          for j in 0..maxIndex(bc.0)]
        [bpoly p for p in l]

  coerce(l: L UP):$ ==
```

```
        lp := [p::UBP for p in l]
        lp::$

    coerce(bc:$):L UP ==
        lp := bc::L UBP
        l := [p::UP for p in lp]
```

B

Les domaines relatifs aux courbes B-Rationnelles

B.1 Les vecteurs massiques

```
)abb domain VM VecteurMassique

  VecteurMassique(F:Field) : pub == priv where

    L ==> List

    E ==> Expression

    pub == VectorSpace(F) with

      pp : (L F,F) -> $
      v? : $ -> Boolean
      p? : $ -> Boolean
      vect : L F -> $
      vm : L F -> $
      coord : $ -> L F
      poids : $ -> F

    priv == add

      V := L F
      PP := Record(cf : V,pds : F)
      Rep := Union(V,PP)

      PF := PrintableForm
      OF := OutputForm

      v1,v2 : $


      vect(l : L F) == autoCoerce(l)
```

```
pp(l:L F,p:F) == construct (l,p)

v?(v1) == v1 case V

p?(v1) == v1 case PP

vm(l:L F) ==
  poids:= l.last
  l := [l.i for i in 0..(maxIndex(l)-1)]
  poids = 0 =>
    vect l
  pp([e/poids for e in l],poids)

outv(l:V):E ==
  le := [e::E for e in l]
  form(formMatrix [[ev] for ev in le])::E

outp(pt:PP):E ==
  poids := form(pt.pds::E)
  le := [e::E for e in pt.cf]
  hconcat (poids,form(formMatrix [[ev] for ev in le]))::E

coerce(v1):E ==
  v? v1 => outv (v1::V)
  outp (v1::PP)

coord(v1) ==
  v? v1 => (CDR$Lisp v1)::V
  v1.cf : L F

poids(v1) ==
  v? v1 => 0$F
  v1.pds

c * v1 ==
  p? v1 => construct(v1.cf,v1.pds*c)
  l:V := [c * e for e in (v1::V)]
  autoCoerce l

ad1(p1:PP,p2:PP):$ ==
   l:V
   p := p1.pds + p2.pds
   l1 := p1.cf
   l2 := p2.cf
   p ^= 0 =>
          l := [((p1.pds*e1)+(p2.pds*e2))/p for e1 in l1 for e2 in l2]
          construct(l,p)
   l := [(p1.pds*e1)+(p2.pds*e2) for e1 in l1 for e2 in l2]
   autoCoerce l
```

```
    ad2(p1:PP,v:V):$ ==
        p1.pds ^= 0 =>
          l:V := [(p1.pds*p1.cf.i)+(v.i / p1.pds)
                        for i in 0..maxIndex(p1.cf)]
          construct(l,p1.pds)
        autoCoerce v

    ad3(v:V,vv:V):$ ==
        l:V := [e + ee for e in v for ee in vv]
        autoCoerce l

    v1 + v2 ==

      p? v1 =>
         p? v2 => ad1(v1::PP,v2::PP)
         ad2(v1::PP,v2::V)
      p? v2 => ad2(v2::PP,v1::V)
      ad3(v1::V,v2::V)
```

B.2 Les courbes B-Rationnelles

```
)abb Domain BRC RationalCurve

  RationalCurve(t:Expression,F:Field) : pub == priv where

    VM       ==> VecteurMassique(F)
    E        ==> Expression
    U        ==> UnivariateBernsteinPolynomial(t,F)
    UP       ==> UnivariatePoly(t,F)
    QF       ==> QuotientField UP
    L        ==> List
    I        ==> Integer
    NNI      ==> NonNegativeInteger
    PF       ==> PrintableForm
    OF       ==> OutputForm
    BC       ==> BCurve(t,F)

    pub == with

      coerce : $       -> E
      coerce : L VM    -> $
      coerce : $       -> L VM
      coerce : $       -> L QF
      coerce : $       -> L U
      coerce : $       -> BC
      coerce : BC      -> $
```

```
      br     : L QF    -> $
      poly?  : $       -> Boolean

    priv == add

--  Representation interne

     V := VM
     Rep := L V

--  Packages visibles

     import PF
     import OF
     import U
     import L
     import BC

--  Declaration des variables globales
     lvm : L VM
     brp : $

--  Test polynomial

     poly?(brp) ==
        b:Boolean := true
        for v in brp repeat if poids(v)$VM = 0 then b:= false
        b

--  Fonctions de coercition

     transform:VM -> L F
     transform(v) ==
       poids(v) = 0 => coord(v)
       [poids(v) * c for c in coord(v)]

     coerce(brp):L U ==
       ll := coord(brp.first)$VM
       n := maxIndex(ll)
       lc:L L F := [transform(v) for v in brp]
       lp:L L F := [[poids(v)] for v in brp]
       l0:L L F := [append(ec,ep) for ec in lc for ep in lp]
       l1:L L F := [[p.i for p in l0] for i in 0..maxIndex l0.0]
       [bpoly(z)$U for z in l1]

     coerce(brp):L QF ==
       lu := brp :: L U
       lp := [u::UP for u in lu]
       lq := [p/lp.last for p in lp]
```

```
      reverse rest reverse lq

    coerce(brp):BC ==
      poly? brp => error "not polynomial curve"
      lu := brp :: L U
      (reverse rest reverse lu) :: BC

    coerce(bc:BC):$ ==
          [pp(e,1)$VM for e in bc::L L F] :: $

    coerce(lvm) == lvm:$

    coerce(brp):L VM == brp:L VM

--  Transformation en expression : fonctions internes

    outB(k:VM,i:I,n:I):E ==
      e := mkUnary(scripts(form("B"),[form(i::E),form(n::E)])::E,t)
      poids(k) = 1 => e
      formTimes(poids(k)::E,e)

    outTerm(k:VM,i:I,n:I):E ==
      e := mkUnary(scripts(form("B"),[form(i::E),form(n::E)])::E,t)
      formTimes(k::E,e)

    outA(l : L E):E ==
      #l=1 => first l
      formPlus l

--  Transformation en expression : fonction exportee

    coerce(brp) ==
      n := maxIndex(brp)
      pnum := [outTerm(brp.i,i,n) for i in 0..n | p? brp.i]
      vnum := [outTerm(brp.i,i,n) for i in 0..n | v? brp.i]
      denom := [outB(brp.i,i,n) for i in 0..n | p? brp.i]
      null pnum => outA vnum
      null vnum => formQuotient(outA pnum,outA denom)
      formPlus(formQuotient(outA pnum,outA denom),
               formQuotient(outA vnum,outA denom))

--  Calcul des vecteurs massiques a partir d'une liste de fractions rationnelles

--  fonctions internes

    gcdl(lp:L UP):UP ==
         n:=maxIndex lp
         n=0 => lp.first
         gcdl [gcd(lp.i,lp.(n-i)) for i in 0..(n quo 2)]
```

```
    hom(l:L QF):L UP ==
        ld := [denom e for e in l]
        delta := gcdl ld
        ldp := [p/delta for p in ld]
        beta := delta * (*/ldp)
        nconc([numer(e*beta) for e in l],[numer(beta)])

--  Fonction exportee

    br(l:L QF) ==
        ll := hom l
        lbp := [e::U for e in ll]
        m:NNI := 0$NNI
        for u in lbp repeat m := max(m,bdegree(u))
        lbp := [ajust(u,m) for u in lbp]
        [vm [coef(u,i) for u in lbp] for i in 0..m]::$
```

B.3 La paramétrisation des courbes B-Rationnelles

```
)abb package PARAM ParametrizationPackage

ParametrizationPackage(t:Expression,K:Field):pub == priv where

  UP     ==>  UnivariatePoly(t,K)
  QF     ==>  QuotientField UP
  BC     ==>  BCurve(t,K)
  BR     ==>  RationalCurve(t,K)
  UBP    ==>  UnivariateBernsteinPolynomial(t,K)
  L      ==>  List

  pub == with

    chgvar      : UP -> QF
    chgvar2     : UP -> QF
    halfspace   : QF -> QF
    allspace    : QF -> QF
    halfspace   : BC -> BR
    allspace    : BC -> BR
    halfspace   : BR -> BR
    allspace    : BR -> BR

  priv == add

    chgvar(p) ==
      constant?(p)$UP => p::QF
      q:QF := ((varPol()$UP)/(1-varPol()$UP))**(degree(p)$UP)
```

```
    c:UP := leadingCoef(p)$UP :: UP
    q1:QF:= c*q + chgvar(reductum(p)$UP)

  chgvar2(p) ==
    constant?(p)$UP => p::QF
    q:QF := ( (1-2*(varPol()$UP)) /
                (varPol()$UP * (1-varPol()$UP)) )**(degree(p)$UP)
    c:UP := leadingCoef(p)$UP :: UP
    q1:QF:= c*q + chgvar2(reductum(p)$UP)

  halfspace(q:QF) ==
    chgvar(numer(q)$QF) / chgvar(denom(q)$QF)

  allspace(q:QF) ==
    chgvar2(numer(q)$QF) / chgvar2(denom(q)$QF)

  halfspace(rc:BR) ==
    lq1 := rc :: L QF
    lq2 :=[halfspace q for q in lq1]
    br(lq2)$BR

  allspace(rc:BR) ==
    lq1 := rc :: L QF
    lq2:=[allspace q for q in lq1]
    br(lq2)$BR

  halfspace(bc:BC) == halfspace(bc::BR)

  allspace(bc:BC) == allspace(bc::BR)
```

C

Les polynômes-exponentiels

```
)abb domain POLEXP PolynomialExponential


PolynomialExponential(t:E,R,POLY,integ): pub == priv where

  integ: POLY -> POLY
  R : Ring
  POLY : Join(UnivPolyCat R ,RetractableTo R ,DifferentialRing)

  E        ==>          Expression
  UB       ==>          UnivariateBernsteinPolynomial(t,R)
  FE       ==>          FunctionalExpression R
  SE       ==>          SortedExpressions
  INTDOM   ==>          IntegralDomain

  pub ==  Join(Algebra(POLY),Algebra(R),DifferentialRing) with

    if R has Field then
      integrate    : $                    -> $
    if R has INTDOM then
      coerce       :  $                   -> FE
    coerce         : POLY                 -> $
    exp            : R                    -> $
    leadingCoef    : $                    -> POLY
    listcoef       : $                    -> List POLY
    listOfTerm     : $                    -> List R
    leadingMonomial:$                     -> $
    map            : ((POLY -> POLY),$)   -> $
    reductum       : $                    -> $
    leadingTerm    : $                    -> R
    cte            : $                    -> R
    coef           : ($,R)                -> POLY
    delete         : (R,$)                -> $

  priv == add

------------------------------------------------------------------
--  Representation interne
```

```
-----------------------------------------------------------------------

    Term := Record(k:R,c:POLY)
    Rep  := List Term

-----------------------------------------------------------------------
--  Importation de paquetages
-----------------------------------------------------------------------

    import OutputForm
    import PrintableForm

-----------------------------------------------------------------------
--  Fonctions de manipulation
-----------------------------------------------------------------------

    reductum x     == (null x => 0; rest x)
    leadingCoef x  == (null x => 0; x.0.c)
    leadingTerm x  == (null x => error "No terms"; x.0.k)

    delete(n,x) ==
      null x => []
      leadingTerm x = n => x.rest
      [x.first,:delete(n,x.rest)]

    listcoef(px) == [p.c for p in px]

    leadingMonomial(x) == (null x => error "No terms"; [first x])

    listOfTerm(px) == [p.k for p in px]

    exp(e:R) == [construct(e,1$POLY)]

    coef(pe,n) ==
      null pe             => 0$POLY
      leadingTerm pe = n => leadingCoef pe
      coef(reductum pe,n)

-----------------------------------------------------------------------
--  operations arithmetiques
-----------------------------------------------------------------------

    -x == [[u.k,-u.c] for u in x]


    if R has OrderedSet then

       x = y ==
         while not null x and not null y repeat
```

```
        not x.first.k = y.first.k => return false
        not x.first.c = y.first.c => return false
        x:=x.rest
        y:=y.rest
      null x and null y

   x + y  ==
      null x => y
      null y => x
      y.first.k > x.first.k => [y.first,:(x + y.rest)]
      x.first.k > y.first.k => [x.first,:(x.rest + y)]
      r:= x.first.c + y.first.c
      r = 0 => x.rest + y.rest
      [[x.first.k,r],:(x.rest + y.rest)]

   x - y  ==
      null x => - y
      null y => x
      y.first.k > x.first.k => [[y.first.k,-y.first.c],:(x - y.rest)]
      x.first.k > y.first.k => [x.first,:(x.rest - y)]
      r:= x.first.c - y.first.c
      r = 0 => x.rest - y.rest
      [[x.first.k,r],:(x.rest - y.rest)]

else
   x = y ==
     null x and null y => true
     #x ^= #y => false
     r:= coef(y,x.first.k)
     r ^= x.first.c => false
     x.rest = delete(x.first.k,y)

   x + y  ==
      null x => y
      null y => x
      co:= coef(y,x.first.k)
      co = 0 => [x.first,:(x.rest + delete(x.first.k,y))]
      r:= x.first.c + co
      r = 0 => x.rest + delete(x.first.k,y)
      [[x.first.k,r],:(x.rest +delete(x.first.k,y))]

   x - y  ==
      y := -y
      x+y

Zero == []
One == [construct(0$R,1$POLY)]

mult: (Term,$) -> $
```

```
    mult(t1,q) ==
      [construct(t1.k + q.i.k,t1.c * q.i.c)
                        for i in 0..maxIndex q]

    p:$ * q:$ ==
      (p = 0) or (q = 0) => 0
      p = 1              => q
      q = 1              => p
      r:$ := 0
      for t1 in p repeat
        r := r + mult(t1,q)
      r

    cte p ==
      null p => 0$R
      coef(leadingCoef p,0)$POLY + cte reductum p

    n:R * x:$ ==
      (n=0) or null(x) => []
      [construct(t1.k,n*t1.c) for t1 in x]

    n:Integer * x:$ ==
      (n=0) or null(x) => []
      [construct(t1.k,n*t1.c) for t1 in x]

    p:POLY * x:$ ==
      (p=0) or null(x) => []
      [construct(t1.k,p*t1.c) for t1 in x]

    i:NonNegativeInteger * x:$ ==
      (i=0) or null(x) => []
      [construct(t1.k,i*t1.c) for t1 in x]

-------------------------------------------------------------------
-- conversion en expression fonctionnelle
-------------------------------------------------------------------

    if R has INTDOM then
      convert : E  -> FE
      convert(t)  == t::SE::FE

      pTofe : POLY -> FE
      pTofe(p) ==
        (degree p = 0) => lc(p) :: FE
        ^(POLY has canonicalUnitNormal) =>
                    error " not canonicalUnitNormal"
        lc(p)*(convert(t)**degree p) + pTofe(red p)

      coerce(px:$):FE ==
```

```
        px = 0 => 0$FE
        leadingTerm px = 0 => pTofe(leadingCoef(px))
        p:=leadingCoef(px)
        pTofe(p)*exp(leadingTerm(px)*convert(t))$FE + reductum(px)::FE

-----------------------------------------------------------------------
-- conversion en expression
-----------------------------------------------------------------------

    outTerm: Term -> E
    outTerm(p) ==
      p.k = 0$R => p.c :: E
      if p.k = 1$R
        then term := form(var()$POLY)
        else term := form(p.k :: E) * form(var()$POLY)
      f := super(form("%e"),term)
      p.c = 1$POLY => f :: E
      (form(p.c ::$POLY E) * f) :: E

    coerce(pe:$):E ==
      null pe => (0$POLY) :: E
      le := [outTerm(p) for p in pe]
      formPlus le

-----------------------------------------------------------------------
-- conversion des entiers et des polynomes
-----------------------------------------------------------------------

    coerce(i:Integer) ==
      i = 0 => []
      [construct(0$R,i::POLY)]

    coerce(p:POLY) ==
      p = 0 => []
      [construct(0$R,p)]

-----------------------------------------------------------------------
--  Derivation & Integration
-----------------------------------------------------------------------

    tderiv: Term -> Term
    tderiv t1 ==
      pol := deriv(t1.c)
      t1.k = 0    => construct(0,pol)
      construct(t1.k,pol + (t1.k * t1.c) )

    deriv(p:$) ==
      p2 := [tderiv t1 for t1 in p]
      [t1 for t1 in p2 | not(t1.c = 0)]
```

```
if R has Field then

  tinteg: Term -> Term
  tinteg t1 ==
    t1.c = 0 => t1
    t1.k = 0 => construct(0,integ(t1.c))
    pol:POLY := deriv(t1.c)$POLY
    p1 := tinteg(construct(t1.k,pol/(t1.k)))
    p2:POLY := ( (t1.c)/(t1.k) ) - p1.c
    construct(t1.k,p2)

  integrate p ==
    p2:$ := [tinteg t1 for t1 in p]
    p2 := p2 - (cte p2)::POLY::$
    [t1 for t1 in p2 | not(t1.c = 0)]
```

D

Les paquetages pour le tracé des courbes

D.1 Les courbes de Bézier

```
)abb package B2D BezierDraw2D

BezierDraw2D(): pub == priv where

    SF          ==> SmallFloat
    I           ==> Integer
    Pt          ==> Record(xCoord:SF,yCoord:SF)
    L           ==> List
    VIEW2D      ==> TwoDimensionalViewport
    P           ==> Palette
    C           ==> Color
    PI          ==> PositiveInteger
    E           ==> Expression
    t:E
    BC          ==> BCurve(t,SF)
    UBP         ==> UnivariateBernsteinPolynomial(t,SF)
    UP          ==> UnivariatePoly(t,SF)

    nbCOLORS    ==> 4

    pub == with

       internPolygon:   L Pt              -> L Pt
       subdiv:          L Pt              -> L L Pt
       resultPolygon:   L Pt              -> L L Pt
       bcurve:         (L Pt,I)           -> L L Pt
       drawB2D:        (L Pt,I,String)    -> VIEW2D
       drawB2D:        (L Pt,I)           -> VIEW2D
       drawB2D:         L Pt              -> VIEW2D
       drawB2D:        (L BC,I,String)    -> VIEW2D
       drawB2D:        (BC,I,String)      -> VIEW2D
       drawB2D:        (BC,String)        -> VIEW2D
```

```
        drawB2D:          BC                  -> VIEW2D
        drawB2D:         (L UBP,I,String)     -> VIEW2D
        drawB2D:         (L UBP,String)       -> VIEW2D
        drawB2D:          L UBP               -> VIEW2D
        drawB2D:         (L UP,I,String)      -> VIEW2D
        drawB2D:         (L UP,String)        -> VIEW2D
        drawB2D:          L UP                -> VIEW2D

    priv == add

-- PACKAGES VISIBLES

        import TwoDimensionalViewport
        import GraphImage
        import Color
        import Palette
        import BC
        import UBP
        import UP
        import Pt

-- FONCTIONS RELATIVES A L'ALGORITHME DE SUBDIVISION

        internPolygon(l) ==
              [[(e1.xCoord+e2.xCoord)/2,(e1.yCoord+e2.yCoord)/2]
                                   for e1 in l for e2 in rest l]

        subdiv(l) ==
              #l = 1 => [l]
              cons(l,subdiv(internPolygon(l)))

        resultPolygon(l) ==
              lp := subdiv(l)
              [[p.first for p in lp],[p.last for p in reverse lp]]

        bcurve(l,n) ==
              n = 0 => [l]
              p := resultPolygon(l)
              append(bcurve(p.first,n-1),bcurve(p.last,n-1))


-- FONCTIONS DE TRACE EFFECTIF

-- Trace a partir de listes de points

        drawB2D(l:L Pt,n:I,titre:String) ==
              l1:L L Pt := bcurve(l,n)
              lp:L P := [bright(red()),bright(yellow()),
                         pastel(green()),light(blue())]
```

```
            if ((nbdup := (# l1) quo nbCOLORS) > 1) then
               lp := [:[lp.j for i in 1..nbdup]
                             for j in 0..maxIndex(lp)]
            graph1 := makeGraphImage(l1,[],lp,[])
            makeViewport2D(graph1,titre)

        drawB2D(l:L Pt,n:I) ==
            drawB2D(l,n,"Bezier curve")

        drawB2D(l:L Pt) ==
            nbsub : Integer := 1
            nbp := (2 * #l) :: Integer
            while ( (nbsub * nbp) < 96 ) repeat
              nbsub := nbsub+1
              nbp := 2 * nbp
            drawB2D(l,nbsub,"Bezier curve")

-- Trace a partir d'expressions polynomiales

-- Fonction locale de conversion

        convert:L UBP -> L Pt
        convert(lu) ==
            # lu ^= 2 => error "not 2 dimensional curve"
            p1 := lu.first
            p2 := lu.last
            if ((n1 :=  bdegree p1) ^= (n2 := bdegree p2)) then
               m:= max(n1,n2)
               if m > n1 then
                  p1 := ajust(p1,m)
               else
                  p2 := ajust(p2,m)
            lx := listCoef p1
            ly := listCoef p2
            [construct(lx.i,ly.i)$Pt
                        for i in 0 .. maxIndex lx]

-- Fonctions exportees

        drawB2D(lu:L UBP,n:I,titre:String) ==
            l1:L Pt := convert lu
            drawB2D(l1,n,titre)

        drawB2D(lu:L UBP,titre:String) ==
            l1:L Pt := convert lu
            nbsub : Integer := 1
            nbp := (2 * #l1) :: Integer
            while ( (nbsub * nbp) < 96 ) repeat
              nbsub := nbsub +1
```

```
            nbp := 2 * nbp
          drawB2D(l1,nbsub,titre)

    drawB2D(lu:L UBP) ==
          drawB2D(lu,"Bezier curve")

    drawB2D(b:BC,n:I,titre:String) ==
          lu : L UBP :=  b::L UBP
          drawB2D(lu,n,titre)

    drawB2D(lu:L BC,n:I,titre:String) ==
          courbes:L L Pt := [:bcurve(convert u::L UBP,n) for u in lu]
          drawM2D(courbes,titre)$MultiDraw2D

    drawB2D(b:BC,titre:String) ==
          lu : L UBP :=  b::L UBP
          drawB2D(lu,titre)

    drawB2D(b:BC) ==
          drawB2D(b,"Bezier curve")

    drawB2D(lp:L UP,n:I,titre:String) ==
          lu:L UBP := [e::UBP for e in lp]
          drawB2D(lu,n,titre)

    drawB2D(lp:L UP,titre:String) ==
          lu:L UBP := [e::UBP for e in lp]
          drawB2D(lu,titre)

    drawB2D(lp:L UP) ==
          drawB2D(lp,"Bezier curve")
```

D.2 Les courbes B-Rationnelles

```
)abb package R2D Rationaldraw2D

Rationaldraw2D() : pub == priv where

  t       :   Expression
  SF      ==> SmallFloat
  UBP     ==> UnivariateBernsteinPolynomial(t,SF)
  Pt      ==> Record(xCoord:SF,yCoord:SF)
  RC      ==> RationalCurve(t,SF)
  UP      ==> UnivariatePoly(t,SF)
  QF      ==> QuotientField UP
  S       ==> String
  L       ==> List
```

```
  L1        ==> L SF
  L2        ==> L L SF
  L3        ==> L L L SF
  I         ==> Integer
  VIEW2D    ==> TwoDimensionalViewport
  PR        ==> ParametrizationPackage(t,SF)

  pub == with

    internPolygon :   L SF          -> L SF
    subdiv        :   L SF          -> L L SF
    resultPolygon :   L SF          -> L L SF
    bcurve        :  (L SF,I)       -> L L SF
    ratToTrace    :  (RC,I)         -> L L Pt
    drawR2D       :  (RC,I,S)       -> VIEW2D
    drawR2D       :  (L QF,I,S)     -> VIEW2D
    drawR2D       :  (L QF,I,S,S)   -> VIEW2D

  priv == add

-- PACKAGES VISIBLES

    import UBP
    import RC
    import VIEW2D
    import GraphImage
    import L1
    import L2
    import L3

    point:(SF,SF,SF)    -> Pt
    point(x,y,p) ==
      p = 0 => construct(x,y)$Pt
      construct(x/p,y/p)$Pt

    coordonnee:L3 -> L L Pt
    coordonnee(lp) ==
      l1:L2 := first(lp)$L3
      l0:L1 := first(l1)$L2
      n:I := maxIndex(l0)$L1
      m:I := maxIndex(l1)$L2
      i,j : I
      lpds:L2:= last(lp)$L3
      lx:L2  := first(lp)$L3
      ly:L2  := lp(1)
      [[point(lx.i.j,ly.i.j,lpds.i.j) for j in 0..n | lpds.i.j ^=0]
                                                    for i in 0..m]

-- FONCTIONS RELATIVES A L'ALGORITHME DE SUBDIVISION
```

```
    internPolygon(l) ==
      [( l.i + l.(i+1) )/2 for i in 0..(maxIndex(l)-1)]

    subdiv(l) ==
      #l = 1 => [l]
      [l,:subdiv(internPolygon(l))]

    resultPolygon(l) ==
      lp := subdiv(l)
      n := maxIndex(lp)
      [[lp.i.first for i in 0..n],[lp.(n-i).last  for i in 0..n]]

    bcurve(l,n) ==
      n = 0 => [l]
      p := resultPolygon(l)
      append(bcurve(p.first,n-1),bcurve(p.last,n-1))

-- FONCTIONS DE TRACE EFFECTIF

    ratToTrace(rc:RC,n:I) ==
      lu := rc:: L UBP
      lp:L L L SF := [bcurve(listCoef(b)$UBP,n) for b in lu]
      coordonnee(lp)


    drawR2D(rc:RC,n:I,titre:S) ==
      lpt := ratToTrace(rc,n)
      graph := makeGraphImage(lpt,[],[],[])
      makeViewport2D(graph,titre)


    drawR2D(lq:L QF,n:I,titre:S) ==
      rc:=br(lq)$RC
      drawR2D(rc,n,titre)

    drawR2D(lq:L QF,n:I,titre:S,mode:S) ==
      mode = "h" =>
        lq1 := [halfspace(q)$PR for q in lq]
        drawR2D(lq1,n,titre)
      lq1 := [allspace(q)$PR for q in lq]
      drawR2D(lq1,n,titre)
```

D.3 Les courbes polynomiales-exponentielles

```
)abb package PX2D PolxDraw2D
```

```
PolxDraw2D(integ): pub == priv where

    SF          ==> SmallFloat
    E           ==> Expression
    t:E
    UBP         ==> UnivariateBernsteinPolynomial(t,SF)
    integ:      UBP -> UBP
    PX          ==> PolynomialExponential(t,SF,UBP,integ)

    I           ==> Integer
    Pt          ==> Record(xCoord:SF,yCoord:SF)
    L           ==> List
    S           ==> String
    VIEW2D      ==> TwoDimensionalViewport
    P           ==> Palette
    C           ==> Color
    PI          ==> PositiveInteger
    BC          ==> BCurve(t,SF)
    UP          ==> UnivariatePoly(t,SF)

    nbCOLORS    ==> 4

    pub == with

       points     :    (PX,I)        -> L Pt
       points     :      PX          -> L Pt
       drawX2D    :   (PX,I,S)       -> VIEW2D
       drawX2D    :    (PX,I)        -> VIEW2D
       drawX2D    :     (PX)         -> VIEW2D
       drawX2D    :  (L PX,I,S)      -> VIEW2D
       drawX2D    :   (L PX,I)       -> VIEW2D
       drawX2D    :    (L PX)        -> VIEW2D

    priv == add

-------------------------------------------------------------------------
-- fonctions locales
-------------------------------------------------------------------------

-- expression des poly dans la meme base
       ajustement: PX ->  PX
-- liste des temps pour le calcul des exponentielles
       listtime:(I,L SF) -> L SF
-- calcul de points exacts par subdivision pour une expr Pk.exp(kt)
       exactTerm:(PX,I,L SF) -> L SF
-- calcul de points exacts par subdivision pour Sum( Pk.exp(kt) )
       exactpts:(PX,I,L SF) -> L SF

-- PACKAGES VISIBLES
```

```
        import TwoDimensionalViewport
        import GraphImage
        import Color
        import Palette
        import BC
        import UBP
        import UP
        import Pt

------------------------------------------------------------------------
-- trace d'une courbe polynomiale-exponentielle: fonctions locales
------------------------------------------------------------------------

        ajustement(px) ==
          lcf := (listcoef px)::BC::L UBP
          lk := listOfTerm px
          ppx:PX := 0$PX
          for pol in lcf for k in lk repeat
            ppx := ppx + pol*exp(k)$PX
          ppx

        listtime(nbsub,l) ==
          nbsub = 0 => l
          mid := (l.first+l.last)/2
          l1:L SF := [l.first,mid]
          l2:L SF := [mid,l.last]
          removeDuplicates append(listtime(nbsub-1,l1),listtime(nbsub-1,l2))

        exactTerm(term,nbsub,ltps) ==
          lu: L UBP := listeP(leadingCoef term,nbsub)
          ls:L L SF := [listCoef e for e in lu]
          lx:L SF := [e.first for e in ls]
          lx := append(lx,[ls.last.last])
          k := leadingTerm term
          [x * exp(k*tps)$SF for x in lx for tps in ltps]

        exactpts(px,nbsub,ltps) ==
          rd := reductum px
          rd = 0 => exactTerm(px,nbsub,ltps)
          l1: L SF := exactTerm(leadingMonomial(px),nbsub,ltps)
          l2 : L SF := exactpts(rd,nbsub,ltps)
          [e1 + e2 for e1 in l1 for e2 in l2]


        points(px:PX,nbsub:I) ==
          pxad:PX := ajustement(px)
          ltps :L SF:= listtime(nbsub,[0$SF,1$SF])
          lpt: L SF := exactpts(pxad,nbsub,ltps)
```

```
          [[tps,pt]$Pt for tps in ltps for pt in lpt]

        points(px:PX) ==
          points(px,6)

-- FONCTIONS DE TRACE EFFECTIF
-- Une seule courbe

        drawX2D(px:PX,n:I,titre:S) ==
          graph1 := makeGraphImage([points(px,n)],[],[],[])
          makeViewport2D(graph1,titre)

        drawX2D(px:PX,n:I) ==
          drawX2D(px,n,"Ex-po curve")

        drawX2D(px:PX) ==
          drawX2D(px,6)

-- Plusieurs courbes

        drawX2D(lpx:L PX,n:I,titre:S) ==
          lp:= [points(px) for px in lpx]
          graph1 := makeGraphImage(lp,[],[],[])
          makeViewport2D(graph1,titre)

        drawX2D(lpx:L PX,n:I) ==
          drawX2D(lpx,n,"Ex-po curve")

        drawX2D(lpx:L PX) ==
          drawX2D(lpx,6)
```

E

Les programmes externes pour le tracé des courbes

E.1 Le programme *C*

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>

#include <stdio.h>

#define NB_COORD  1024
#define MARGE_X  10
#define MARGE_Y  5

Display *display ;
int screen ;
main(argc,argv)

  int argc;
  char **argv;
{
Window win;
unsigned int width,height;
int x =0, y = 0;
int i;
unsigned int border_width = 2;
unsigned int display_width,display_height;
char *window_name ="BEZIER";
char *icon_name = "BEZIER";

XSizeHints size_hints;
XEvent report;
GC gc;
XFontStruct *font_info;


XPoint points[NB_COORD];
int fd,j;
```

```
char buf[8*NB_COORD],car[1],nb[4];
unsigned n,k;

  if ( (fd = open(argv[1],0)) == -1 )
    printf("impossible d'ouvrir le fichier %s\n",argv[1]);
  else {
    printf("ok \n");
    n=0;
    read(fd,car,1);
    do {
         n = n * 10 + (car[0] - '0');
         read(fd,car,1);
       }
    while(car[0] != ' ');

    read(fd,buf,n*8);
    for (i=0; i<n ;i++) {
      for (j=0;j<4;j++) nb[j] = buf[8*i+j];
      sscanf(nb,"%u",&k);
      points[i].x = k+MARGE_X;
      for (j=0;j<4;j++) nb[j] = buf[8*i+4+j];
      sscanf(nb,"%u",&k);
      points[i].y = k+MARGE_Y;
      }
    }


if ( (display = XOpenDisplay( NULL )) == NULL )
  {
    (void) fprintf( stderr ,
      " prog. fenetre : connection imposiiiible avec X-serveur %s\n",
     XDisplayName( NULL ));
    exit( -1 );
  }
screen = DefaultScreen( display );
width  =  512+2*MARGE_X;
height =  512+2*MARGE_Y;
win = XCreateSimpleWindow( display , RootWindow( display, screen ),
                  x , y , width , height , border_width ,
                  BlackPixel( display , screen ),
                  WhitePixel( display , screen ));
size_hints.flags = PPosition | PSize | PMinSize ;
size_hints.x     = x ;
size_hints.y     = y ;
size_hints.width = width ;
size_hints.height= height ;
size_hints.min_width = 350 ;
size_hints.min_height= 250 ;
XSetStandardProperties( display , win , window_name , icon_name ,
```

```
                        NULL, argv , argc , &size_hints );
XSelectInput( display , win , ExposureMask | ButtonPressMask |
              StructureNotifyMask );
load_font ( &font_info );
get_GC( win , &gc , font_info );
XMapWindow( display , win );
while( 1 ) {
  XNextEvent( display , &report );
  switch (report.type) {
    case Expose : XDrawLines(display,win,gc,points,n,CoordModeOrigin);
break;      case ConfigureNotify:break;
    case ButtonPress:exit(1);break;
    default :break;
    }
  }
 }
load_font(font_info)
XFontStruct **font_info;
{
char *fontname="Rom14.500";
if ((*font_info=XLoadQueryFont(display,fontname))==NULL)
 {printf("ERREUR FONT");
  exit(-1);
 }
}
get_GC( win , gc , font_info , mode )
Window win ;
GC *gc ;
XFontStruct *font_info ;
int mode;
{
 unsigned long valuemask = 0 ;
     /* ignorer XGCvalues , utiliser les valeurs par defaults */
 XGCValues values ;

 /* creer le contexte graphique par default  */

*gc=XCreateGC(display,win,valuemask,&values);

XSetFont(display,*gc,font_info->fid);
  XSetForeground(display,*gc,BlackPixel(display,screen));
}
```

E.2 Le programme *PostScript*

Le fichier exécutable *PostScript* se compose d'un entête contenant en particulier une instruction *translate* pour cadrer la figure, et une instruction *scale* pour sa réduction. Ces commandes

son facilement modifiables "à la main" suivant le résultat recherché.

```
gsave
  newpath

  1 setlinejoin
  100 100 translate

  gsave
    0.5 setlinewidth

    0.26 0.26 scale
    0 0 moveto
```

On trouve ici la liste des points à tracer, ceux-ci étant reliés par des segments de droite. Enfin, on trouve l'ensemble du programme permettant l'affichage des échelles et du cadre.

```
%=======================================================
% Definition des constantes
%=======================================================
/longueur_taquet 4 def      % tiret pour les graduations
/fleche_x 14 def            % une fleche sur les axes
/fleche_y 4 def             %
/marge_x 50 def             % un offset pour le cadre
/marge_y 50 def             %
/width 1024 def             % la taille de la figure
/height 1024 def            %

%=======================================================
% Definition des fontes
%=======================================================
/textfont{
  /Times-Roman findfont 12 scalefont setfont }def

/numfont{
  /Times-Roman findfont 18 scalefont setfont }def

%=======================================================
% width height PROC --> rectangle autour point courant
%=======================================================
/cadre{
  /h exch def
  /w exch def
  w 0  rlineto
  0 h rlineto
  w neg 0 rlineto
  closepath
}def
```

```
%=======================================================
% nombre_taquet decalage_entre_taquets x_1er_point PROC
%=======================================================
/taquetenx{
  axe_x_x add
  axe_x_y
  moveto
  /decalage exch def
  /decalage_reel exch def
  /pas_reel exch def

  /chaine 8 string def
  {
    0 longueur_taquet rmoveto
    0 longueur_taquet 2 mul neg
    rlineto
    gsave
      decalage_reel chaine cvs
      dup stringwidth pop 2 div neg
      -15 rmoveto
      /decalage_reel decalage_reel pas_reel add def
      show
    grestore
    decalage longueur_taquet rmoveto
 }repeat
}def

/taqueteny{
  axe_y_y add
  axe_y_x exch
  moveto
  /decalage exch def
  /decalage_reel exch def
  /pas_reel exch def
  {
    longueur_taquet 0 rmoveto
    longueur_taquet 2 mul neg 0
    rlineto
    gsave
      decalage_reel chaine cvs
      dup stringwidth pop neg 2 rmoveto
      show
      /decalage_reel decalage_reel pas_reel add def
    grestore
    longueur_taquet decalage rmoveto
  }repeat
}def

%=======================================================
```

```
% angle_degre PROC ->fleche noire au pt courant
%=======================================================
/fleche{
  rotate
  fleche_x neg fleche_y neg rlineto
  0 fleche_y 2 mul rlineto
  closepath
  0 setgray
  fill
}def

%=======================================================
% des axes au milieu de la figure
%=======================================================
/axe_y{
  moveto
  0 height rlineto
}def

/axe_x{
  moveto
  width 0 rlineto
}def


%=======================================================
% centrer une chaine
%=======================================================
/centrer{
  dup
  stringwidth
  pop
  width exch sub
  2 div
  0 rmoveto
  show
}def



%=======================================================
%                    DEBUT
%=======================================================

    numfont

    %--------------la courbe  ----------------------------
    /axe_x_x 0 def
    /axe_x_y exch def
```

```
/axe_y_x exch def
/axe_y_y 0 def

%-------------- les axes  ----------------------------
newpath
0 0 moveto

axe_y_x axe_y_y
axe_y
axe_x_x axe_x_y
axe_x
stroke

%-------------- fleche verte -------------------------
newpath
gsave
  axe_y_x height moveto
  90
  fleche
grestore
%-------------- fleche horiz -------------------------
newpath
gsave
  width axe_x_y moveto
  0
  fleche
grestore

%-------------- les taquets --------------------------
newpath
gsave
  taquetenx
  stroke
  taqueteny
  stroke
grestore

%-------------- le cadre  ----------------------------
gsave
  newpath
  2 setlinewidth
  marge_x neg marge_y neg
  moveto
  width marge_x 2 mul add
  height marge_y 2 mul add
  cadre
  stroke
grestore
```

```
  grestore
  %-------------- les comments -------------------------
  gsave
    newpath
    0 30 moveto
    textfont
  grestore

grestore
%-------------- on imprime ----------------------------
showpage
```

Résumé

Le but de ce travail est de développer, pour le système de calcul formel Scratchpad-Axiom, des outils de simulation graphique pour l'étude du comportement des systèmes dynamiques non linéaires.

Nous codons de tels systèmes par une série formelle en variables non commutatives selon la technique proposée par M. Fliess. Leur comportement est alors déterminé par une transformation d'évaluation appliquée ici à des approximations polynomiales ou rationnelles des séries pour des entrées polynomiales ou polynomiales-exponentielles.

Les entrées et sorties sont définies par des courbes de Bézier, permettant une modification interactive des entrées par déplacement des points de contrôle, puis étendues aux courbes rationnelles sous forme (BR) et aux courbes polynomiales exponentielles.

Enfin, l'implantation en Scratchpad-Axiom de l'algorithme d'intégration numérique de Runge-Kutta a permis la simulation d'approximations nilpotentes et ainsi la visualisation du problème du Motion planning, résolu dans des cas simples.

Ces différentes implantations ont permis en outre des comparaisons qualitatives des différentes méthodes d'approximation et de simulation.

Mots clé : Systèmes dynamiques non linéaires, Séries formelles, Transformation d'évaluation, Motion planing, Courbes de Bézier, Courbes B-Rationnelles, Calcul formel.