Faculteit Wetenschappen
Vakgroep Toegepaste wiskunde en informatica

Automatische Differentiatie in Matlab

Tineke Goessens

Promotor: Prof. Dr. W. Govaerts

Begeleiders: Prof. Dr. W. Govaerts en V. De Witte

Masterproef ingediend tot het behalen van de academische graad van master in de wiskunde, afstudeerrichting toegepaste wiskunde

Academiejaar 2008-2009

Faculteit Wetenschappen
Vakgroep Toegepaste wiskunde en informatica

Automatische Differentiatie in Matlab

Tineke Goessens

Promotor: Prof. Dr. W. Govaerts

Begeleiders: Prof. Dr. W. Govaerts en V. De Witte

Masterproef ingediend tot het behalen van de academische graad van master in de wiskunde, afstudeerrichting toegepaste wiskunde

Academiejaar 2008-2009

Dankwoord

Toelating tot bruikleen

De auteur geeft de toelating deze masterproef voor consultatie beschikbaar te stellen en delen van de masterproef te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze masterproef.

Gent, 1 juni 2009 Tineke Goessens

Inhoudsopgave

Inleiding

Tot het midden van de 17de eeuw was de algebraïsche wiskunde voornamelijk gebasseerd op algoritmen, stap-voor-stap recepten voor oplossingen van problemen. Een belangrijk voorbeeld hiervan is Euclides algoritme voor de grootste gemene deler van twee gehele getallen. Daarnaast is het zo dat als het menselijk brein denkt in opeenvolgende stappen, zoals het optellen van lange kolommen met getallen, dit traag en soms onnauwkeurig is. Daardoor waren eenvoudige berekeningen soms ondoorzichtig en onbegrijpbaar als alleen de algoritmen beschikbaar waren.
Meetkundigen aan de andere kant maakten gebruik van intuïtieve symbolische visualisaties voor rechten, cirkels, driehoeken enzovoort. Vanaf de 17de eeuw begon men ook in de algebraïsche wiskunde te werken met zulke symbolische voorstellingen, wat leidde tot een snelle ontwikkeling van de toenmalige wiskunde. Voorbeelden hiervan zijn I. Newton en G. Leibniz die differentiaal- en integraalrekening introduceerden in deze eerste jaren van formalisme. Deze formules liggen nog steeds aan de basis van automatische differentiatie.
De kracht van de keuze voor een geschikte notatie in formules en manipulatie van die formules om problemen op te lossen stond centraal bij de moderne wiskundigen van de laatste 350 jaar, en zal altijd een blijven bestaan in de toekomst. Anderzijds heeft de komst van de computer ervoor gezorgt dat vanaf het midden van de 20ste eeuw algoritmen terug aan belang wonnen, nu echter niet meer uitgevoerd door het traag en onnauwkeurig menselijk brein maar door in de vorm van computerprogramma's. Zo wordt nu voortdurend gezocht naar methoden die algoritmen kunnen optimaliseren en even efficiënt maken als de formules voorhanden. In zo'n algortime worden problemen uitgesplitst tot hun kleinste bestanddelen: de elementaire bewerkingen zoals optellen, aftrekken, vermenigvuldigen en delen en lineaire combinaties

van deze elementaire operators. Deze notie dat problemen kunnen worden beschreven aan de hand van ingewikkelde functies die op hun beurt kunnen ontbonden worden tot een opeenvolging van elementaire bewerkingen ligt aan de basis van alle automatische differentiatie tools.

De laatste 30 jaar werd de ontwikkeling van automatische differentiatie tools gedreven door de nood aan efficiënte evaluatie van afgeleiden en gevoed door onze voortdurend verbeterde mogelijkheden om zulke tools te creëren. Een van de principiële motivaties voor automatische differentiatie (AD) is de nood voor exacte berekening van waarden van afgeleiden van functies gebruikt in wetenschappelijke modellering. Wetenschappers gebruiken namelijk steeds complexere modellen waarin steeds hogere afgeleiden worden gebruikt. In bijvoorbeeld atmosferisch en oceanografisch onderzoek is de nood aan exacte waarden voor afgeleiden bijzonder groot. Samen met deze nood aan nauwkeurigheid, heeft de vooruitgang in de moderne programmeertalen en computeromgevingen zoals Matlab [1] de ontwikkeling van AD tools vereenvoudigd.

Ook in vele wiskundige toepassingen is het nodig, afgeleiden van hogere orde van een gegeven functie nauwkeurig te kunnen berekenen. Voor orde hoger dan 2 zijn benaderingen met eindige differenties meestal onvoldoende nauwkeurig. Het gebruik van symbolische afgeleiden is niet altijd mogelijk en bovendien in de regel traag voor hogere orde afgeleiden. Automatische differentiatie is hier een alternatief; het is even nauwkeurig als symbolische afleiding maar vereist geen speciale toolboxen.

In het bijzonder denken we in het kader van deze thesis aan het Matlab softwarepakket MatCont voor numerieke wiskundige modellering van dynamische systemen, d.w.z. systemen van differentiaalvergelijkingen met parameters. Dit pakket heeft behoefte aan afgeleiden tot en met orde 5 voor de berekening van de coëfficiënten van de normaalvormen. Deze coëfficiënten zijn belangrijk want ze bepalen bijvoorbeeld of een oplossing stabiel is, of er nieuwe oplossingen ontstaan enzoverder.

Het doel van deze scriptie was enerzijds de studie van algoritmen voor automatische differentiatie als alternatief voor symbolische afgeleiden, en ander-

[1] De software developer The MathWorks beschrijft Matlab als volgt: Matlab is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.

zijds de codering hiervan in Matlab. Codes voor de berekening van Taylorreeksen van functies van één variabele waren reeds beschikbaar in het softwarepakket Cl_MatcontM, maar werkten nog niet volledig naar behoren en nog niet voor meerdimensionele afbeeldingen. Voor de implementatie van deze algoritmen werden de object-georienteerde methoden in Matlab gebruikt. Aan de hand van deze code kunnen de directionele afgeleiden van functies van meerdere variabelen worden bekomen uit de hogere orde afgeleiden van functies van een variabele ([KUZ04], Sectie 10.3.4) en dit is de sleutel voor de berekening van de coëfficiënten van de normaalvormen.

Om inzicht te krijgen in de gebruikte algoritmes in de ADTayl-klasse was het nuttig om een deel van hoofdstuk 13 "Taylor and Tensor Coefficients" uit het boek "Evaluating Derivatives" van A. Griewank [GRIE08] uit te werken. Dit vindt u terug in het eerste hoofdstuk.

Het tweede hoofdstuk behandeld automatische differentiatie en geeft een overzicht van codim 1 en codim 2 bifurcaties die detecteerbaar zijn bij dynamische systemen van afbeeldingen. We geven er een overzicht van de polarisatieidentiteiten die nuttig zijn bij het berekenen van de normaalvormcoëfficiënten bij bifurcaties. Aan de hand van deze identiteiten kan men namelijk de berekening van partiële afgeleiden in multilineaire vormen herleiden tot de berekening van directionele afgeleiden.

In hoofdstuk drie gaan we dieper in op de implementatie van AD in het Matlab softwarepakket Cl_MatcontM. We maken er een vergelijking tussen symbolisch afleiden en automatisch afleiden aan de hand van o.a. een test case en een voorbeeld van symbolisch afleiden in Maple. De belangrijkste correcties die werden aangebracht in de oorspronkelijk code en opmerkelijke ontdekkingen hierbij worden eveneens in dit hoofdstuk vermeld.

Uiteindelijk geven we in hoofdstuk 4 een overzicht wat mogelijkheden voor toekomstig werk betreft.

Hoofdstuk 1

Taylor- en Tensorcoëfficiënten

In dit deel volgt een uitwerking van hoofdstuk 13 "Taylor and Tensor Coefficients" van het boek "Evaluating Derivatives" van A. Griewank [GRIE08]. In dit hoofdstuk wordt de berekening van derde- en hogere-orde directionele afgeleiden van functies herleid tot de berekening van Taylorreeksontwikkelingen van univariate functies en in het bijzonder van de coëfficiënten die hierin voorkomen. Ook wordt het belang daarvan aangetoond voor het berekenen van ontaarde minima en andere singuliere punten. Onder andere in de numerieke bifurcatietheorie berekent men deze singulariteiten door bepaalde parameters in het probleem variabel te laten en andere vast te houden. De sleutel voor de interpretatie van bifurcatiepunten ligt in het berekenen van de normaalvormcoëfficiënten, waarvoor het berekenen van directionele hogere orde afgeleiden eveneens noodzakelijk is.

1.1 Motiverend voorbeeld

We beschouwen een scalaire functie $f(x, y, \lambda, \mu)$ en onderstellen dat we geïnteresseerd zijn in het berekenen van een ontaard minimum van deze functie met betrekking tot x en y. We beschouwen λ en μ als de bifurcatieparameters in het probleem.

Als we parametriseren

$$\begin{cases} x(t) &= x_0 + t\Delta x \\ y(t) &= y_0 + t\Delta y, \end{cases}$$

dan

$$\begin{cases} \dot{x} &= \Delta x \\ \dot{y} &= \Delta y. \end{cases}$$

Voor gegeven waarden van λ en μ kunnen we de functie f als volgt ontwikkelen rond (x_0, y_0)

$$\begin{aligned} f(x,y) &= f(x_0,y_0) + f_x(x_0,y_0)\Delta x + f_y(x_0,y_0)\Delta y \\ &+ \frac{1}{2}\left[f_{xx}(x_0,y_0)(\Delta x)^2 + 2f_{xy}(x_0,y_0)\Delta x\Delta y\right. \\ &\quad \left. + f_{yy}(x_0,y_0)(\Delta y)^2\right] \qquad (1.1) \\ &+ \frac{1}{6}\left[f_{xxx}(x_0,y_0)(\Delta x)^3 + 3f_{xxy}(x_0,y_0)(\Delta x)^2\Delta y\right. \\ &\quad \left. + 3f_{xyy}(x_0,y_0)\Delta x(\Delta y)^2 + f_{yyy}(x_0,y_0)(\Delta x)^3\right] \qquad (1.2) \\ &+ O(|\Delta x|^4, |\Delta y|^4). \qquad (1.3) \end{aligned}$$

De nodige voorwaarden voor een ontaard minimum zijn dan:

1. De gradiënt is 0 of dus $0 = f_x = f_y$.
2. De kwadratische vorm in (1.1) moet niet-negatief definiet zijn en $f_{xy}^2 - f_{xx}f_{yy} = 0$ moet gelden.
3. De derde-orde term in (1.2) moet nul zijn in de $(\Delta x, \Delta y)$-richting waarin de term (1.1) nul wordt.

Voorwaarde 2 drukt uit dat de Hessiaan van f, $\begin{pmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{pmatrix}$, singulier moet zijn, of dus een nulvector moet hebben. We kunnen als nulvector $(\dot{x}, \dot{y}) = (\Delta x, \Delta y)$ kiezen waarbij

$$\begin{pmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

en dus

$$\begin{pmatrix} \Delta x & \Delta y \end{pmatrix} \begin{pmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = 0$$

waarbij het linkerlid juist het dubbel van (1.1) is.
Om normalisatieredenen stellen we deze $(\Delta x, \Delta y)$ gelijk aan $(\cos(\alpha), \sin(\alpha))$ voor een zekere $\alpha \in [0, 2\pi[$. Opdat de derde voorwaarde zou gelden moet de derde directionele afgeleide van f langs $(\dot{x}, \dot{y})$ nul worden.

Op die manier komen we tot het volgende systeem van derde-orde optimalisatiecondities in de volledig variabele vector $(x, y, \lambda, \mu, \alpha)$ met

$$\begin{cases} \dot{x} &= \cos(\alpha) \\ \dot{y} &= \sin(\alpha) \end{cases}$$

$0 = f_x = f_y$	eerste-orde noodzakelijke voorwaarde
$0 = f_{xx}\dot{x} + f_{yx}\dot{y} = f_{xy}\dot{x} + f_{yy}\dot{y}$	tweede-orde ontaardingsvoorwaarde
$0 = f_{xxx}\dot{x}^3 + 3f_{xxy}\dot{x}^2\dot{y} + 3f_{xyy}\dot{x}\dot{y}^2 + f_{yyy}\dot{y}^3$	derde-orde noodzakelijke voorwaarde

Deze drie voorwaarden zijn noodzakelijk om een ontaard minimum te hebben in (x, y). Opdat deze ook voldoende zouden zijn is de extra eis een vierde-orde niet-ontaardingsvoorwaarde die inhoudt dat de vierde directionele afgeleide langs $(\dot{x}, \dot{y})$ niet-negatief definiet moet zijn (zie [GRI80]). We moeten dus niet enkel de gradiënt en Hessiaan van f berekenen, maar ook de directionele afgeleiden langs $(\dot{x}, \dot{y})$ van derde en vierde orde.

We kunnen nu algemeen een functie $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ beschouwen, die $d + 1$ keer continu afleidbaar is. Dan is de Taylorreeksontwikkeling van $F(x + ts)$ rond $t = 0$:

$$
\begin{aligned}
F(x+ts) &= F(x(t)) \\
&= F_0(x,s) + F_1(x,s)t + F_2(x,s)t^2 + ... + F_d(x,s)t^d + O(t^{d+1}) \\
&= F(x+ts)|_{t=0} + \frac{\partial}{\partial t}F(x+ts)\bigg|_{t=0} t + \frac{1}{2}\frac{\partial^2}{\partial t^2}F(x+ts)\bigg|_{t=0} t^2 + \\
&\quad ... + \frac{1}{d!}\frac{\partial^d}{\partial t^d}F(x+ts)\bigg|_{t=0} t^d + O(t^{d+1}) \\
&= F(x(0)) + \nabla F(x(0))st + \frac{1}{2}\nabla^2 F(x(0))s^2t^2 + ... \\
&\quad + \frac{1}{d!}\nabla^d F(x(0))s^d t^d + O(t^{d+1}).
\end{aligned}
$$

Hierin is ∇F de gradiënt F_x van F en is $\nabla^2 F$ de Hessiaan HF van F. Verder zijn de factoren F_k de directionele afgeleiden van F in het punt x, nodig om bijvoorbeeld het ontaarde minimum te vinden in het voorbeeld hierboven. Dit zijn dus de Taylorcoëfficiënten :

$$
F_k(x,s) = \frac{1}{k!}\frac{\partial^k}{\partial t^k}F(x+ts)\bigg|_{t=0} = \frac{1}{k!}\nabla^k F(x(0))s^k \in \mathbb{R}^m \text{ voor } k \leq d \quad (1.4)
$$

1.2 Taylorpolynomen en het compositieprobleem

1.2.1 Algemeen

Beschouwen we nu opnieuw de algemene functie $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Het is dan interessant om te kijken hoe de Taylorcoëfficiënten van $F(x(t))$ kunnen geschreven worden voor een gegeven vectorpolynoom $x(t)$, als we de Taylorcoëfficiënten kennen van deze $x(t)$. In de literatuur wordt naar dit probleem verwezen als het Compositieprobleem

Zij gegeven een polynoom

$$x(t) = x_0 + tx_1 + t^2x_2 + ... + t^dx_d \in \mathbb{R}^n. \tag{1.5}$$

Dan wordt de Taylorreeksontwikkeling van $F(x(t))$

$$\begin{aligned} F(x(t)) &= y(t) + O(t^{d+1}) \\ &= y_0 + ty_1 + t^2y_2 + ... + t^dy_d + O(t^{d+1}) \in \mathbb{R}^m. \end{aligned} \tag{1.6}$$

De indices geven hier de orde van de Taylorcoëfficiënt. De vectoren y_j worden op unieke wijze bepaald door de coëfficiëntvectoren x_i, met $i \leq j$ als gevolg van de kettingregel:

$$\begin{aligned} F(x(t)) &= F(x(t))|_{t=0} + \nabla F(x(t))|_{t=0}\, t + \frac{1}{2}\nabla^2 F(x(t))\bigg|_{t=0} t^2 \\ &\quad + \frac{1}{6}\nabla^3 F(x(t))\bigg|_{t=0} t^3 + ... \end{aligned}$$

met

$$\begin{aligned} \nabla F(x(t)) &= F'(x(t))(x_1 + 2x_2t + 3x_3t^2 + 4x_4t^3 + ...) \\ \nabla^2 F(x(t)) &= F''(x(t))(x_1 + 2x_2t + 3x_3t^2 + 4x_4t^3 + ...)(x_1 + 2x_2t + 3x_3t^2 \\ &\quad +4x_4t^3 + ...) + F'(x(t))(2x_2 + 6x_3t + 12x_4t^2 + ...) \\ \nabla^3 F(x(t)) &= F'''(x(t))(x_1 + 2x_2t + 3x_3t^2 + 4x_4t^3 + ...)(x_1 + 2x_2t + 3x_3t^2 \\ &\quad +4x_4t^3 + ...)(x_1 + 2x_2t + 3x_3t^2 + 4x_4t^3 + ...) + 3F''(x(t)) \\ &\quad (2x_2 + 6x_3t + 12x_4t^2 + ...)(x_1 + 2x_2t + 3x_3t^2 + 4x_4t^3 + ...) \\ &\quad +F'(x(t))(6x_3 + 24x_4t + ...). \end{aligned} \tag{1.7}$$

We krijgen dus voor $t = 0$

$$\begin{aligned} F(x(t))|_{t=0} &= F(x_0) \\ \nabla F(x(t))|_{t=0} &= F'(x_0)x_1 \\ \nabla^2 F(x(t))\big|_{t=0} &= F''(x_0)x_1x_1 + 2F'(x_0)x_2 \\ \nabla^3 F(x(t))\big|_{t=0} &= F'''(x_0)x_1x_1x_1 + 6F''(x_0)x_2x_1 + 6F'(x_0)x_3, \end{aligned}$$

zodat

$$\begin{aligned} y_0 &= F(x_0) \\ y_1 &= F'(x_0)x_1 \\ y_2 &= F'(x_0)x_2 + \frac{1}{2}F''(x_0)x_1x_1 \\ y_3 &= F'(x_0)x_3 + F''(x_0)x_1x_2 + \frac{1}{6}F'''(x_0)x_1x_1x_1 \\ &... \end{aligned} \tag{1.8}$$

Het is duidelijk dat het aantal termen die voorkomen in deze "symbolische" uitdrukkingen voor y_j, exponentieel groeit met j. Het is namelijk zo dat bij het berekenen van de tensor van j-de orde, mn^j termen te berekenen zijn. Dit wil dus zeggen dat om y_j te berekenen eveneens mn^j termen moeten berekend worden. Voor polynomen van hoge graad wordt dit dus een tijdrovende zaak.
Door gebruik te maken van automatische differentiatie kunnen we deze snelle groei vermijden. We zullen dit bekomen door het combineren van bepaalde termen op zo'n manier dat de nodige opslagruimte en het aantal bewerkingen slechts kwadratisch is in d, de bovengrens voor j.

We maken nu de volgende veronderstelling voor elementaire functies φ_i, zoals $+$, $*$, $-$, $/$, exp, sin,...

Veronderstelling (EA)**.** ELEMENTAIRE AFLEIDBAARHEID
Alle elementaire functies φ_i zijn d keer continu afleidbaar in hun open domein $\mathcal{D}_i$, i.e. $\varphi_i \in \mathbb{C}^d(\mathcal{D}_i)$, $0 \leq d \leq \infty$.

Definitie 1.1. Taylorcoëfficiëntfuncties
Onder de onderstelling (EA) noemen we

$$y_k = F_k(x_0, x_1, x_2, ..., x_k) \qquad met \qquad F_k : \mathbb{R}^{n\times(k+1)} \longrightarrow \mathbb{R}^m$$

de Taylorcoëfficiëntfunctie voor $k \leq d$ gedefinieerd door relaties (1.5) en (1.6).

Deze definitie is een veralgemening van $F_k(x, s)$ gegeven door (1.4). Om de Taylorcoëfficiëntfunctie F_k te berekenen met automatische differentiatie,

hebben we ook de overeenkomstige Taylorcoëfficiënten van alle intermediaire variabelen nodig (zie verder bij Hoofdstuk 2: Automatische Differentiatie).

Definitie 1.2. Taylorpolynoom van Intermediairen
Voor een gegeven d keer continu afleidbaar inputpad $x(t) : (-\epsilon, \epsilon) \longrightarrow \mathbb{R}^n$ *met* $\epsilon > 0$*, zijn de resulterende waarden van een intermediaire variabele* v *de waarden* $v(x(t)) : (-\epsilon, \epsilon) \longrightarrow \mathbb{R}$ *en is het overeenkomstige Taylorpolynoom*

$$v(t) = v_0 + tv_1 + t^2v_2 + ... + t^dv_d = v(x(t)) + o(t^d).$$

We hebben met andere woorden het symbool v overschreven zodat we drie verschillende begrippen kunnen voorstellen, nl. de waarde van de intermediaire variabele zelf als men geen argument vermeldt, het exacte waardenpad van $v(x(t))$ en een schatting van de taylorpolynoom van graad d door $v(t)$. De v_k stellen hierbij de Taylorcoëfficiënten voor in $t = 0$ met $0 \leq k \leq d$. Deze kunnen berekend worden aan de hand van de rekenkundige recursiebewerkingen die we uitwerken in sectie 1.2.2. De bekomen algoritmen worden vermeld in Tabel 1.1.

In deze tabel vindt men in de derde kolom het aantal elementaire rekenkundige bewerkingen nodig om de volledige recursie met k van 0 tot d uit te werken. Opmerkelijk hierbij is dat het berekenen van de vierkantswortel van een Taylorpolynoom minder rekenkundige bewerkingen vraagt dan het vermenigvuldigen van twee Taylorpolynomen, iets wat we op het eerste zicht niet zouden verwachten. De uitwerking van deze aantallen rekenkundige bewerkingen wordt eveneens gegeven in de volgende sectie.

Tabel 1.1: Algoritmen voor Taylorcoëfficiënten via recursie

$v =$	Recursie voor $k = 0..d$	Aantal bewerkingen
$u + cw$	$v_k = u_k + cw_k$	$2d$
$u * w$	$v_k = \sum_{j=0}^{k} u_j w_{k-j}$	d^2
u/w	$v_k = \frac{1}{w_0}\left[u_k - \sum_{j=0}^{k-1} v_j w_{k-j}\right]$	d^2
u^2	$v_k = \sum_{j=0}^{k} u_j u_{k-j}$	$\frac{1}{2}d^2$
$\sqrt{u}$	$v_k = \frac{1}{2v_0}\left[u_k - \sum_{j=1}^{k-1} v_j v_{k-j}\right]$	$\frac{1}{2}d^2$

1.2.2 Algoritmes voor Taylorcoëfficiënten van elementaire functies

Stel $v(t) = \varphi(u(t), w(t))$ met φ een elementaire operator, dan kunnen we de Taylorcoëfficiënten v_1, v_2, ..., v_k berekenen uit deze van u en w, of dus uit u_1, u_2, ..., u_k en w_1, w_2, ..., w_k, via eenvoudige veeltermbewerkingen.

1. $\underline{v(t) = u(t) + cw(t)}$

   We hebben

   $$v_0 + v_1 t + v_2 t^2 + v_3 t^3 + ...$$
   $$\begin{aligned} &= (u_0 + u_1 t + u_2 t^2 + u_3 t^3 + ...) + c(w_0 + w_1 t + w_2 t^2 + w_3 t^3 + ...) \\ &= (u_0 + cw_0) + (u_1 + cw_1)t + (u_2 + cw_2)t^2 + (u_3 + cw_3)t^3 + ... \end{aligned}$$

   Gelijkstellen van coëfficiënten bij gelijke machten van t geeft dan

   $$\begin{array}{rcll} v_0 & = & u_0 + cw_0 & 2 \text{ bewerkingen} \\ v_1 & = & u_1 + cw_1 & 2 \text{ bewerkingen} \\ v_2 & = & u_2 + cw_2 & 2 \text{ bewerkingen} \\ v_3 & = & u_3 + cw_3 & 2 \text{ bewerkingen} \\ & \vdots & & \\ v_k & = & u_k + cw_k & 2 \text{ bewerkingen} \end{array} \tag{1.9}$$

   In totaal zijn er dus voor de recursie met $k = 0...d$, $2(d+1)$ bewerkingen nodig. Slechts rekening houdend met de hoogste-ordeterm wordt dit orde $2d$ bewerkingen.

2. $\underline{v(t) = u(t) * w(t)}$

   We hebben

$v_0 + v_1 t + v_2 t^2 + v_3 t^3 + ...$

$$\begin{aligned} &= (u_0 + u_1 t + u_2 t^2 + u_3 t^3 + ...) * (w_0 + w_1 t + w_2 t^2 + w_3 t^3 + ...) \\ &= (u_0 w_0) + (u_0 w_1 + u_1 w_0) t + (u_0 w_2 + u_1 w_1 + u_2 w_0) t^2 \\ &\quad + (u_0 w_3 + u_1 w_2 + u_2 w_1 + u_3 w_0) t^3 + ... \end{aligned}$$

Gelijkstellen van coëfficiënten bij gelijke machten van t geeft dan

$$\begin{aligned} v_0 &= u_0 w_0 && \text{1 bewerking} \\ v_1 &= u_0 w_1 + u_1 w_0 && \text{3 bewerkingen} \\ v_2 &= u_0 w_2 + u_1 w_1 + u_2 w_0 && \text{5 bewerkingen} \\ v_3 &= u_0 w_3 + u_1 w_2 + u_2 w_1 + u_3 w_0 && \text{7 bewerkingen} \\ &\vdots \\ v_k &= \sum_{j=0}^{k} u_j w_{k-j} && 2k+1 \text{ bewerkingen} \end{aligned} \qquad (1.10)$$

In totaal zijn er dus voor de recursie met $k = 0...d$,

$$1 + 3 + 5 + 7 + ... + (2d + 1)$$

bewerkingen nodig. De som van de eerst n oneven getallen is n^2, dus zijn er $(d+1)^2$ bewerkingen voor de recursiebetrekking. Slechts rekening houdend met de hoogste-ordeterm wordt dit orde d^2 bewerkingen.

3. $\underline{v(t) = u(t)/w(t)}$

We kunnen dit ook anders uitdrukken, nl. $w(t) * v(t) = u(t)$ en hebben dan

$u_0 + u_1 t + u_2 t^2 + u_3 t^3 + ...$

$$\begin{aligned} &= (w_0 + w_1 t + w_2 t^2 + w_3 t^3 + ...) * (v_0 + v_1 t + v_2 t^2 + v_3 t^3 + ...) \\ &= (w_0 v_0) + (w_0 v_1 + w_1 v_0) t + (w_0 v_2 + w_1 v_1 + w_2 v_0) t^2 \\ &\quad + (w_0 v_3 + w_1 v_2 + w_2 v_1 + w_3 v_0) t^3 + ... \end{aligned}$$

Gelijkstellen van coëfficiënten bij gelijke machten van t geeft dan

$$u_k = \sum_{j=0}^{k} v_j w_{k-j},$$

zoals voorheen.
Hieruit vinden we dan

$$\begin{aligned}
v_0 &= \frac{u_0}{w_0} && \text{1 bewerking} \\
v_1 &= \frac{u_1}{w_0} - \frac{v_0 w_1}{w_0} && \text{3 bewerkingen} \\
v_2 &= \frac{u_2}{w_0} - \frac{v_0 w_2}{w_0} - \frac{v_1 w_1}{w_0} && \text{5 bewerkingen} \\
v_3 &= \frac{u_3}{w_0} - \frac{v_0 w_3}{w_0} - \frac{v_1 w_2}{w_0} - \frac{v_2 w_1}{w_0} && \text{7 bewerkingen} \\
&\vdots \\
v_k &= \frac{1}{w_0}\left[u_k - \sum_{j=0}^{k-1} v_j w_{k-j}\right] && 2k+1 \text{ bewerkingen}
\end{aligned}$$

In totaal zijn er dus voor de recursie met $k = 0...d$,

$$1 + 3 + 5 + 7 + ... + (2d + 1)$$

bewerkingen nodig. De som van de eerst n oneven getallen is n^2, dus zijn er $(d+1)^2$ bewerkingen voor de recursiebetrekking. Slechts rekening houdend met de hoogste-ordeterm wordt dit orde d^2 bewerkingen.

4. $\underline{v(t) = u(t)^2}$

We zitten dan opnieuw in geval 2 met $w(t) = u(t)$ en krijgen dan

$$v_k = \sum_{j=0}^{k} u_j u_{k-j}.$$

Er zijn voor de recursie met $k = 0...d$ maar half zoveel bewerkingen nodig dan bij de vermenigvuldiging uit geval 2 omwille van symmetrie. We vinden hier dus orde $\frac{1}{2}d^2$ rekenkundige bewerkingen.

5. $\underline{v(t) = \sqrt{u(t)}}$

   We kunnen dit ook anders uitdrukken, nl. $v(t) * v(t) = u(t)$ en hebben dan

   $$u_k = \sum_{j=0}^{k} v_j v_{k-j}.$$

   Hieruit vinden we dan

   $$\begin{aligned}
   v_0 &= \sqrt{u_0} \\
   v_1 &= \frac{u_1}{2v_0} \\
   v_2 &= \frac{u_2}{2v_0} - \frac{v_1 v_1}{2v_0} \\
   v_3 &= \frac{u_3}{2v_0} - \frac{v_1 v_2}{2v_0} - \frac{v_2 v_1}{2v_0} \\
   &\vdots \\
   v_k &= \frac{1}{2v_0}\left[u_k - \sum_{j=1}^{k-1} v_j v_{k-j}\right]
   \end{aligned}$$

   Er zijn voor de recursie met $k = 0...d$ maar half zoveel bewerkingen nodig dan bij de deling uit geval 3 omwille van symmetrie. We vinden hier dus orde $\frac{1}{2}d^2$ rekenkundige bewerkingen.

1.2.3 Niet-lineaire univariate elementairen

Nemen we nu een elementaire functie $v = \varphi(u)$ dan komen we via symbolisch afleiden tot de uitdrukkingen

$$\begin{aligned}
v_0 &= \varphi(u_0) \\
v_1 &= \varphi_1(u_0)u_1 \\
v_2 &= \varphi_2(u_0)u_1u_1 + \varphi_1(u_0)u_2 \\
v_3 &= \varphi_3(u_0)u_1u_1u_1 + 2\varphi_2(u_0)u_1u_2 + \varphi_1(u_0)u_3 \\
v_4 &= \varphi_4(u_0)u_1u_1u_1u_1 + 3\varphi_3(u_0)u_1u_1u_2 + \varphi_2(u_0)(u_2u_2 + 2u_1u_3) + \varphi_1(u_0)u_4 \\
v_5 &= \ldots,
\end{aligned}$$

met $\varphi_i = \varphi^{(i)}/i!$ de i-de Taylorcoëfficiënt van φ, vermits

$$\begin{aligned}\varphi(u(t)) &= \varphi(u(t))|_{t=0} + \nabla\,\varphi(u(t))|_{t=0}\,t + \frac{1}{2}\,\nabla^2\varphi(u(t))\big|_{t=0}\,t^2 + \\ &\quad \frac{1}{6}\,\nabla^3\varphi(u(t))\big|_{t=0}\,t^3 + \frac{1}{24}\,\nabla^4\varphi(u(t))\big|_{t=0}\,t^4 + ...\end{aligned}$$

Zoals eerder vermeld in (1.8) voor een algemene functie $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ vinden we met behulp van de kettingregel onmiddellijk v_0, v_1, v_2 en v_3. Voor v_4 hebben we $\nabla^4\varphi(u(t))|_{t=0} = \nabla(\nabla^3\varphi(u(t)))|_{t=0}$ nodig. We weten uit (1.7) dat

$$\begin{aligned}\nabla^3\varphi(u(t)) &= \varphi'''(u(t))(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 \\ &\quad +4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...) + 3\varphi''(u(t)) \\ &\quad (2u_2 + 6u_3t + 12u_4t^2 + ...)(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...) \\ &\quad +\varphi'(u(t))(6u_3 + 24u_4t + ...).\end{aligned}$$

Dus

$\nabla(\nabla^3\varphi(u(t)))$

$$\begin{aligned}&= \varphi^{(4)}(u(t))(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 + \\ &\quad 4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 + \\ &\quad 4u_4t^3 + ...) \\ &\quad +3\varphi'''(u(t))(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 + \\ &\quad 4u_4t^3 + ...)(2u_2 + 6u_3t + 12u_4t^2 + ...) \\ &\quad +3\varphi'''(u(t))(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(u_1 + 2u_2t + 3u_3t^2 + \\ &\quad 4u_4t^3 + ...)(2u_2 + 6u_3t + 12u_4t^2 + ...) \\ &\quad +3\varphi''(u(t))(2u_2 + 6u_3t + 12u_4t^2 + ...)(2u_2 + 6u_3t + 12u_4t^2 + ...) \\ &\quad +3\varphi''(u(t))(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(6u_3 + 24u_4t + ...) \\ &\quad +\varphi''(u(t))(u_1 + 2u_2t + 3u_3t^2 + 4u_4t^3 + ...)(6u_3 + 24u_4t + ...) \\ &\quad +\varphi'(u(t))(24u_4 + ...).\end{aligned}$$

In $t = 0$ hebben we bijgevolg

$$\begin{aligned}\nabla^4\varphi(u(t)) &= \varphi^{(4)}(u_0)u_1u_1u_1u_1 + 12\varphi'''(u_0)u_2u_1u_1 + 12\varphi''(u_0)u_2u_2 \\ &\quad +24\varphi''(u_0)u_1u_3 + 24\varphi'(u_0)u_4.\end{aligned}$$

We vinden dus

$$v_4 = \varphi_4(u_0)u_1u_1u_1u_1 + 3\varphi_3(u_0)u_2u_1u_1 + \varphi_2(u_0)(u_2u_2 + 2u_1u_3) + \varphi_1(u_0)u_4.$$

Door op deze manier verder te redeneren kunnen we ook v_5, ..., v_d vinden. We zien opnieuw in dat het aantal termen in de berekening exponentieel groeit met de graad van de afgeleide. Dit effect wordt zelfs nog duidelijker in het multivariate geval, waarbij u_2u_2, u_1u_3 en u_3u_1 niet gewoon kunnen opgeteld worden.

De opgave om de coëfficiënten v_i voor $i = 0...d$ te berekenen van de koppels (u_i, φ_i) voor $i = 1...d$ is dus opnieuw het compositieprobleem. Men heeft bewezen dat dit probleem van dezelfde rekenkundige complexiteit is als het probleem om een machtreeks $u(t)$ te inverteren, wat vraagt naar het vinden van d afgeleiden van φ zodat $\varphi(u(t)) = 1 + o(t^d)$. De klassieke methoden om beide problemen op te lossen eisen orde d^3 rekenkundige bewerkingen zolang er geen voorwaarden aan de coëfficiënten van φ of u worden opgelegd.
Gelukkig komt zo'n probleem met compleet ongerelateerde Taylorcoëfficiënten φ_0, φ_1, ...,φ_d nooit voor in de praktijk. Alle elementaire functies $\varphi(u)$ zijn in de praktijk namelijk oplossingen van ODE's. Meer specifiek voldoen zij allen aan een identiteit van de vorm

$$b(u)\varphi'(u) - a(u)\varphi(u) = c(u). \tag{1.11}$$

Hierbij worden de coëfficiëntenfuncties $a(u)$, $b(u)$ en $c(u)$ verondersteld "gekend" te zijn, i.e. uitdrukbaar in termen van rekenkundige bewerkingen en univariate functies waarvan de afgeleiden zonder problemen of zonder omslachtige uitwerkingen kunnen berekend worden. We kunnen dus de Taylorcoëfficiënten a_k, b_k en c_k berekenen uit de coëfficiënten u_k van u.
Daarvoor leiden we de identiteit $v(x(t)) = \varphi(u(x(t)))$ af en vermenigvuldigen

dan met de functie $b(u(x(t)))$ om tot de volgende relatie te komen

$$b(u(x(t)))\frac{d}{dt}v(x(t)) = \left[c(u(x(t))) + a(u(x(t)))v(x(t))\right]\frac{d}{dt}u(x(t)). \quad (1.12)$$

Wegens veronderstelling (EA) zijn $v(x(t))$ en $u(x(t))$ d keer continu afleidbaar. Als we uitgaan van de ontwikkelingen

$$u(x(t)) = u_0 + u_1 t + ... + u_d t^d + O(t^{d+1})$$

en

$$v(x(t)) = v_0 + v_1 t + ... + v_d t^d + O(t^{d+1}),$$

kunnen we dus ook voor de afgeleiden van $v(x(t))$ en $u(x(t))$ ontwikkelingen van volgende vorm beschouwen

$$\begin{aligned}
\frac{d}{dt}u(x(t)) &= \tilde{u}_1 + \tilde{u}_2 t + ... + \tilde{u}_d t^{d-1} + O(t^d) \text{ met } \tilde{u}_j = j u_j \text{ voor } j > 0, \\
\frac{d}{dt}v(x(t)) &= \tilde{v}_1 + \tilde{v}_2 t + ... + \tilde{v}_d t^{d-1} + O(t^d) \text{ met } \tilde{v}_j = j v_j \text{ voor } j > 0.
\end{aligned}$$

Als we deze twee ontwikkelingen en de ontwikkelingen van $a(u)$, $b(u)$ en $c(u)$ in (1.12) invullen krijgen we

$$\begin{aligned}
&(b_0 + b_1 t + b_2 t^2 + b_3 t^3 + ...)(\tilde{v}_1 + \tilde{v}_2 t + \tilde{v}_3 t^2 + \tilde{v}_4 t^3 + ...) \\
&\qquad = \big[(c_0 + c_1 t + c_2 t^2 + c_3 t^3 + ...) + (a_0 + a_1 t + a_2 t^2 + a_3 t^3 + ...) \\
&\qquad\quad (v_0 + v_1 t + v_2 t^2 + v_3 t^3 + ...)\big]\,(\tilde{u}_1 + \tilde{u}_2 t + \tilde{u}_3 t^2 + \tilde{u}_4 t^3 + ...)
\end{aligned}$$

of

$$\begin{aligned}
\text{LL} &= b_0\tilde{v}_1 + (b_0\tilde{v}_2 + b_1\tilde{v}_1)t + (b_0\tilde{v}_3 + b_1\tilde{v}_2 + b_2\tilde{v}_1)t^2 + (b_0\tilde{v}_4 + b_1\tilde{v}_3 \\
&\quad + b_2\tilde{v}_2 + b_3\tilde{v}_1)t^3 + ... \\
\text{RL} &= c_0\tilde{u}_1 + (c_0\tilde{u}_2 + c_1\tilde{u}_1)t + (c_0\tilde{u}_3 + c_1\tilde{u}_2 + c_2\tilde{u}_1)t^2 + (c_0\tilde{u}_4 + c_1\tilde{u}_3 \\
&\quad + c_2\tilde{u}_2 + c_3\tilde{u}_1)t^3 + ... \\
&\quad + a_0v_0\tilde{u}_1 + (a_0v_0\tilde{u}_2 + a_0v_1\tilde{u}_1 + a_1v_0\tilde{u}_1)t + (a_0v_0\tilde{u}_3 + a_0v_1\tilde{u}_2 + a_1v_0\tilde{u}_2 \\
&\quad + a_0v_2\tilde{u}_1 + a_1v_1\tilde{u}_1 + a_2v_0\tilde{u}_1)t^2 + (a_0v_0\tilde{u}_4 + a_0v_1\tilde{u}_3 + a_1v_0\tilde{u}_3 + a_0v_2\tilde{u}_2 \\
&\quad + a_1v_1\tilde{u}_2 + a_2v_0\tilde{u}_2 + a_0v_3\tilde{u}_1 + a_1v_2\tilde{u}_1 + a_2v_1\tilde{u}_1 + a_3v_0\tilde{u}_1)t^3 + ...
\end{aligned}$$

Gelijkstellen van de coëfficiënten bij gelijke machten van t geeft met $v_0 = \varphi(u(x(0)))$

$$\begin{aligned}
\tilde{v}_1 &= \frac{1}{b_0}\left[(c_0 + a_0 v_0)\tilde{u}_1\right] \\
\tilde{v}_2 &= \frac{1}{b_0}\left[(c_0 + a_0 v_0)\tilde{u}_2 + (c_1 + a_0 v_1 + a_1 v_0)\tilde{u}_1 - b_1\tilde{v}_1\right] \\
\tilde{v}_3 &= \frac{1}{b_0}\Big[(c_0 + a_0 v_0)\tilde{u}_3 + (c_1 + a_0 v_1 + a_1 v_0)\tilde{u}_2 + (c_2 + a_0 v_2 + a_1 v_1 + a_2 v_0)\tilde{u}_1 \\
&\quad -b_1\tilde{v}_1 - b_2\tilde{v}_1\Big] \\
&\vdots
\end{aligned}$$

En we weten dat

$$v_j = \frac{\tilde{v}_j}{j},$$

voor $j > 0$.

We krijgen dan volgende propositie:

Propositie 1.1. (TAYLORPOLYNOMEN VAN OPLOSSINGEN VAN ODE'S)
Zijn a_k, b_k en c_k de Taylorcoëfficiënten van coëfficiëntenfuncties $a(u)$, $b(u)$ en $c(u)$ die voldoen aan (1.11).
Op voorwaarde dat $b_0 \equiv b(u_0) \neq 0$ geldt dan dat

$$\tilde{v}_k = \frac{1}{b_0}\left[\sum_{j=1}^{k}(c_{k-j} + e_{k-j})\tilde{u}_j - \sum_{j=1}^{k-1} b_{k-j}\tilde{v}_j\right] \quad \textit{voor } k = 1...d$$

met

$$e_k = \sum_{j=0}^{k} a_j v_{k-j} \ \textit{voor } k = 1..(d-1).$$

Door het herschalen van de coëfficiënten u_j en v_j tot $\tilde{u}_j = ju_j$ en $\tilde{v}_j = jv_j$, respectievelijk, halveert het totaal aantal vermenigvuldigingen in deze berekening. Nadien kan men gewoon opnieuw herschalen naar de oorspronkelijke

coëfficiënten zonder veel rekenwerk te moeten uitvoeren.
Als $a(u) \equiv 0$ in de ODE (1.11) dan wordt de elementaire functie φ een rationale kwadratuur van de vorm

$$\varphi(u) = \int \frac{c(u)}{b(u)} du.$$

Vervolgens kunnen we dan in de bovenstaande formule a_j en dus ook e_j nul stellen, wat het aantal rekenkundige bewerkingen reduceert met $d^2 + O(d)$.
Deze reductie van de bewerkingen krijgen we eveneens in het geval waarbij ofwel $a(u) = a_0$, ofwel $b(u) = b_0$, ofwel beide constant zijn. Het aantal overblijvende rekenkundige bewerkingen is dan respectievelijk $3d^2 + O(d)$, $2d^2 + O(d)$ of $d^2 + O(d)$.
In elk van deze gevallen is de groei van de bewerkingen kwadratisch in d, wat vanzelfsprekend de voorkeur geniet vergeleken bij de exponentiële groei bij symbolisch afleiden van een algemene elementaire functie $\varphi(u)$.

Voor de in de ODE's meest voorkomende elementairen met coëfficiëntfuncties $a(u)$, $b(u)$ en $c(u)$ staan de recursieresultaten gebaseerd op Propositie 1.1 opgesomd in Tabel 1.2.

Merk op dat de coëfficiënten voor s_k en c_k van $\sin(u)$ en $\cos(u)$ samen moeten berekend worden door de recursierelaties toe te passen in alternerende volgorde. In Matlab bijvoorbeeld kunnen beide recursies efficiënt geïmplementeerd worden zodat sin en cos telkens samen worden berekend en opgeslagen in het geheugen, ook als slechts één van beide vereist is. Men gaat er namelijk van uit dat deze twee elementairen in praktische problemen meestal samen voorkomen. Eens er sprake is van $\sin(u)$ in de oplossing van een vraagstuk, duikt $\cos(u)$ vaak ook op in het vervolg van de oplossing. Door een efficiënte implementatie kan dan op een later tijdstip eventueel teruggegrepen worden naar de voorgaande berekeningen van sin of cos zodat een nieuwe berekening kan vermeden worden.

Tabel 1.2: Taylorcoëfficiënten van univariate elementaire functies in ODE's

$v =$	a	b	c	Recursie voor $k = 1..d$	Aantal bewerkingen
$\ln(u)$	0	u	1	$\tilde{v}_k = \frac{1}{u_0}\left[\tilde{u}_k - \sum_{j=1}^{k-1} u_{k-j}\tilde{v}_j\right]$	d^2
$\exp(u)$	1	1	0	$\tilde{v}_k = \left[\sum_{j=1}^{k} v_{k-j}\tilde{u}_j\right]$	d^2
u^r	r	u	0	$\tilde{v}_k = \frac{1}{u_0}\left[r\sum_{j=1}^{k} v_{k-j}\tilde{u}_j - \sum_{j=1}^{k-1} u_{k-j}\tilde{v}_j\right]$	$2d^2$
$\sin(u)$	0	1	$\cos(u)$	$\tilde{s}_k = \sum_{j=1}^{k} \tilde{u}_j c_{k-j}$	$2d^2$
$\cos(u)$	0	-1	$\sin(u)$	$\tilde{c}_k = \sum_{j=1}^{k} -\tilde{u}_j s_{k-j}$	$2d^2$

Hoofdstuk 2

Automatische Differentiatie

2.1 Inleiding

Als alternatief voor de twee klassieke manieren om afgeleiden van een functie in een bepaald punt te evalueren gebruiken we nu automatische differentiatie. De twee klassieke manieren zijn

1. **Symbolische differentiatie** (de functie afleiden als een expressie, en daarna evalueren in het punt)
2. **Numerieke differentiatie** (eindige differenties)

waarbij beide hun voor- en nadelen hebben.
De eerste manier, waarbij men symbolisch gaat afleiden, leidt soms tot complexe berekeningen om tot de uiteindelijke expressie te komen, vooral als er hogere orde afgeleiden berekend dienen te worden. Dit is nog meer het geval als het gaat om de afgeleide van iteraties van een functie. Deze volledige expressie wordt in zijn geheel opgeslagen en daarna wordt deze geëvalueerd in het gegeven punt. Dit vraagt een grote opslagruimte in het computergeheugen en heeft anderzijds ook een lage snelheid tot gevolg. Verder zijn dikwijls speciale en vaak dure toolboxen nodig om aan symbolische afleiding te kunnen doen.

Anderzijds kan numeriek afleiden leiden tot grote onnauwkeurigheid door afkappingsfouten. Dit probleem wordt nog groter doordat de nauwkeurigheid nog meer daalt als het gaat om een orde van afleiden hoger dan 2 van (geïtereerde) functies.

In vele wiskundige toepassingen is het echter noodzakelijk afgeleiden van hogere orde van een gegeven functie nauwkeurig te berekenen. Automatische differentiatie (AD) is dan een alternatief. Deze methode is even nauwkeurig als symbolisch afleiden, maar vraagt minder opslagruimte en vereist geen specifieke toolboxen. Met AD is het mogelijk om controlestructuren (lussen, takken en subfuncties) eigen aan moderne programmeertalen in te bouwen, wat niet op een doorzichtige manier haalbaar is met symbolische differentiatie. Verder worden er geen afkappingsfouten gemaakt omdat discretizatie niet nodig is en is AD in de regel sneller (kleinere CPU-time).
Concreet zal de verlopen tijd in de berekeningen met AD lineair groeien met het aantal iteraties J van de gegeven functie, terwijl men bij symbolisch afleiden, zoals reeds besproken in Hoofdstuk 1, kan spreken van een exponentiële groei met J^d voor afgeleiden van orde d. Voor kleine J blijft symbolische differentiatie echter wel sneller dan automatische differentiatie.

In het bijzonder denken we in het kader van deze scriptie aan het Matlab softwarepakket CL_MatContM (zie [MatCont]), voor numerieke wiskundige modellering van discrete dynamische systemen, d.w.z. systemen van afbeeldingen met parameters.
Dit pakket heeft behoefte aan afgeleiden tot en met orde 5 van geïtereerde functies voor de berekening van de normaalvormcoëfficiënten . Uit deze coëfficiënten kan bijvoorbeeld besloten worden of een oplossing al dan niet stabiel is, of er nieuwe oplossingen ontstaan enzoverder.

We beschouwen in deze scriptie een geïtereerde functie

$$f^{(J)}(x,\alpha) = \underbrace{f(f(f(...f}_{J \text{ keer}}(x,\alpha)...,\alpha),\alpha),\alpha)$$

waarbij $x \in \mathbb{R}^n$ een vector van toestandsvariabelen, $\alpha \in \mathbb{R}^p$ een vector van parameters en $f : \mathbb{R}^n \times \mathbb{R}^p \longrightarrow \mathbb{R}^n$ een niet-lineaire functie is.

We definiëren nu een J-cykel als een geordend J-tal $(x_1, ..., x_J)$ waarvoor $f(x_i) = x_{i+1} \quad (i = 1, ..., J-1)$ en $f(x_J) = x_1$.

Hoge waarden voor het aantal iteraties J zijn vaak nodig omdat herhaalde periodeverdubbelingsbifurcaties algemeen voorkomen bij functies die afhangen van minstens één parameter. Met andere woorden is de kans groot dat een voldoend gladde afbeelding van een eindigdimensionale reële ruimte op zichzelf afhangend van één parameter J-cykels heeft met een arbitrair hoge J, voor geschikte parameterwaarden.

Zoals reeds aangegeven zullen we de multilineaire vormen van een geïtereerde functie f^J moeten bepalen. Dit wordt gedaan aan de hand van Taylorreeksontwikkelingen tot 5de orde. De Taylorcoëfficiënten in deze reeksontwikkeling zijn namelijk geschaalde afgeleiden waar men makkelijker mee kan rekenen dan de afgeleiden zelf.

2.2 Automatische differentiatie concreet

We gaan ervan uit dat elke functie f die in een computerprogramma kan voorkomen opgebouwd is uit een eindige reeks van elementaire functies (waarbij men onder elementaire functies ook de basisbewerkingen rekent). Het basisprincipe van AD, zie ook [GRIE08] en [GRIE00], is om de bekende afleidingsregels voor elementaire functies te gebruiken, samen met de kettingregel, om de gevraagde afgeleide van de functie f in kwestie op te bouwen.

We onderstellen dat f een reële vectorfunctie is met n inputs, of dus n onafhankelijke variabelen $x = (x_1, ..., x_n)$ en m outputs, of dus m afhankelijke variabelen $y = (y_1, ..., y_m)$. De code voor f kan zoals eerder gesteld lussen en takken bevatten, maar tegelijkertijd kan elke evaluatie van f in de gegeven inputvariabelen x geschreven worden als een lijst van codes. Deze codelijst is een eindige reeks van toewijzingen van de eenvoudige vorm

$$v_i = e_i(\text{eerder gedefinieerde } v_j\text{'s, of constanten}), \quad i = 1, 2, .., (p + m)$$

waarbij elke e_i een elementaire functie is. De v_i worden variabelen genoemd.

AD kan op twee manier geïmplementeerd worden, de *forward mode* en de *reverse mode*. De *forward mode* van AD is het gemakkelijkst, en is geschikt voor de toepassingen in deze thesis. Zoals in de inleiding van dit hoofdstuk vermeld maken we gebruik van Taylorcoëfficiënten om de hogere afgeleiden

van een gegeven functie f te berekenen. Elke v_i wordt in deze toepassing dan ook voorgesteld als een object $\mathtt{v}_i$ van een datatype `adtayl` dat naast de waarde van de variabele ook de nodige afgeleiden bevat in de vorm van Taylorcoëfficiënten . Het is niet slecht om een `adtayl` object te zien als een object dat een oneindige machtreeks voorstelt gekend tot orde p. We hebben voor deze toepassing:

- Op elke ogenblik wordt slechts één variabele als onafhankelijk beschouwd, we noemen deze variabele t. Op die manier wordt elke v_i gezien als een functie van t.
- De datastructuur bevat Taylorcoëfficiënten . Dit zijn de coëfficiënten van een getrunceerde Taylorreeksontwikkeling van v_i, tot een bepaalde orde p en ontwikkeld rond een bepaald punt $t = a$. We krijgen dan een nieuwe onafhankelijk variabele $s = t - a$.
  v_i zal dan een matrix $(v_{i,0}, v_{i,1}, ..., v_{i,p})$ bevatten waarbij

$$v(a + s) = v_{i,0} + v_{i,1}s + ... + v_{i,p}s^p + O(s^{p+1}).$$

Wanneer zo'n codelijst wordt overlopen zal elke elementaire bewerking toegepast op een reëel argument vervangen worden door een overeenkomstige elementaire bewerking toegepast op een `adtayl` object, waarbij dit object wordt gezien als een oneindige machtreeks gekend tot een bepaalde orde.
Beschouw hierbij eerst de vier rekenkundige bewerkingen, $+, -, \times, \div$. Zij `a` het object dat $(a_0, a_1, ...)$ bevat en idem voor de andere variabelen die volgen. Zij verder a gedefinieerd tot orde p en `b` tot orde q. Definieer nu `c` en `d` als volgt:

$$\begin{aligned} \mathtt{c} &= \mathtt{a} + \mathtt{b} \\ \mathtt{d} &= \mathtt{a} \times \mathtt{b} \end{aligned}$$

Dan zijn zowel `c` als `d` gekend tot orde $r = \min(p, q)$ en bevatten deze objecten $(c_0, c_1, ..., c_r)$ en $(d_0, d_1, ..., d_r)$ met wegens (1.9) en (1.10)

$$\begin{aligned} c_i &= a_i + b_i, \\ d_i &= a_0b_i + a_1b_{i-1} + ... + a_ib_0 \end{aligned}$$

en analoog voor $\texttt{a} - \texttt{b}$ en $\texttt{a} \div \texttt{b}$, voor de laatste op voorwaarde dat $b_0 \neq 0$.

Nemen we bijvoorbeeld $y = (2+t)(3+t^2)$ en willen we de machtreeks tot orde $p = 2$ ontwikkelen rond $t = 1$, dan gaan we als volgt te werk. We maken eerst een object van het type `adtayl` dat de Taylorreeksontwikkeling tot orde 2 van de onafhankelijke variabele t voorstelt in de nieuwe onafhankelijke variabele $s = t - 1$:

$\texttt{t} = (t_0, t_1, t_2) = (1, 1, 0)$ wat de voorstelling is voor $1 + 1s + 0s^2 = 1 + s$.

Daarna hebben we twee constante `adtayl`objecten `c2` en `c3` nodig die respectievelijk de constanten 2 en 3 vasthouden:

$\texttt{c2} = (c_{2,0}, c_{2,1}, c_{2,2}) = (2, 0, 0)$ wat de voorstelling is voor $2+0s+0s^2 = 2$,

$\texttt{c3} = (c_{3,0}, c_{3,1}, c_{3,2}) = (3, 0, 0)$ wat de voorstelling is voor $3+0s+0s^2 = 3$.

Verder hebben we de Taylorreeksontwikkeling nodig van $\texttt{v1} = (2+t)$, $\texttt{v2} = t^2$ en $\texttt{v3} = (3 + t^2)$ wat opnieuw wegens (1.9) en (1.10) wordt:

$$\begin{aligned}
\texttt{v1} &= (v_{1,0}, v_{1,1}, v_{1,2}) = \texttt{c2} + \texttt{t} = (3, 1, 0) \\
&\quad \text{wat de voorstelling is voor } 3 + 1s + 0s^2 + O(s^3) = 3 + s + O(s^3), \\
\texttt{v2} &= (v_{2,0}, v_{2,1}, v_{2,2}) = \texttt{t} \times \texttt{t} = (1, 2, 1) \\
&\quad \text{wat de voorstelling is voor } 1 + 2s + 1s^2 + O(s^3), \\
\texttt{v3} &= (v_{3,0}, v_{3,1}, v_{3,2}) = \texttt{c3} + \texttt{v2} = (4, 2, 1) \\
&\quad \text{wat de voorstelling is voor } 4 + 2s + 1s^2 + O(s^3).
\end{aligned}$$

Als resultaat van een laatste berekening krijgen we dan de uiteindelijke output $y = (2 + t)(3 + t^2)$ wegens (1.10):

$$\begin{aligned}
\texttt{y} &= (y_0, y_1, y_2) = \texttt{v1} + \texttt{v3} = (12, 10, 5) \\
&\quad \text{wat de voorstelling is voor } 12 + 10s + 5s^2 + O(s^3)
\end{aligned}$$

2.3 Multilineaire vormen

Voor een geïtereerde functie f kan men een bifurcatie-analyse uitvoeren. Numeriek wordt dit gedaan door een curve van vaste punten, i.e. oplossingen van de vergelijking

$$F(x, \alpha) = f^J(x, \alpha) - x = 0$$

te continueren met een controleparameter. Door de parameterwaarde te laten variëren, kan men bifurcatiepunten tegenkomen. Dit zijn kritieke parameterwaarden waar de stabiliteit van de vaste punten veranderen en waar takken van nieuwe oplossingen kunnen ontstaan, bijvoorbeeld takken van cykels met een andere periode dan J, of gesloten krommen die invariant zijn onder de inwerking van f^J.

De eigenwaarden van de Jacobiaanmatrix van F^J worden multiplicatoren genoemd. Een vast punt x kan worden geclassificeerd volgens het baangedrag van de punten in de nabijheid van het vaste punt. Zo wordt een vast punt asymptotisch stabiel (of een aantrekkingspunt van f^J) genoemd, als voor stijgende J, de paden van de punten in de nabijheid van het vaste punt er naartoe neigen. We kunnen deze classificering ook maken door naar de absolute waarde van de multiplicatoren te kijken. Een vast punt zal dan asymptotisch stabiel zijn als voor alle multiplicatoren λ, $|\lambda| < 1$. De multiplicatoren van f^J maken dus het onderscheid tussen de aantrekkende en afstotende punten van de afbeelding f^J in de buurt van x. Eigenwaarden groter in absolute waarde dan 1 geven afstotende punten, terwijl eigenwaarden kleiner in absolute waarde dan 1 overeenkomen met aantrekkingspunten. Van zodra er een multiplicator λ bestaat waarvoor $|\lambda| > 1$, is het vaste punt x onstabiel. Als alle multiplicatoren zich buiten de eenheidscirkel bevinden, is x een afstotend punt. Als enkele multiplicatoren binnen en enkele buiten de eenheidscirkel liggen, dan is x een zadelpunt.

Om in de bifurcatietheorie voor discrete dynamische systemen van afbeeldingen de kwalitatieve eigenschappen van deze afbeeldingen te bestuderen zijn er coördinaattransformaties nodig. Deze coördinaattransformateis zijn op hun beurt afhankelijk van de afgeleiden van de afbeeldingen in kwestie. Bijgevolg is automatische differentiatie een aantrekkelijke methode om bifurcaties van discrete dynamische systemen te bestuderen.

In de volgende secties bekijken we de mogelijke bifurcaties van codimensie 1 en codimensie 2 die kunnen optreden bij discrete dynamische systemen van afbeeldingen. Om deze te detecteren zijn testfuncties nodig en om het gedrag van de bifurcatiecurve vervolgens te analyseren bestuderen we normaalvormcoëfficiënten . Om duidelijk te maken dat we hier in aanraking komen met afgeleiden tot hoge orde bespreken we enkele voorbeelden van deze normaalvormcoëfficiënten .

2.3.1 Detectie, localisatie en analyse van codim 1 bifurcaties

Men kan drie codim 1 bifurcatiepunten detecteren op een curve van vaste punten van een geïtereerde functie f^J, namelijk een limietpunt (fold, LP), een periodeverdubbelingspunt (flip, PD) en een Neimark-Sackerpunt (NS). Daarnaast zijn er vertakkingspunten (BP). Om deze singulariteiten te detecteren worden vier testfuncties, ϕ_1, ϕ_2, ϕ_3 en ϕ_4, gedefinieerd (zie [KHOS08]). De respectieve singulariteiten komen dan overeen met de nulwaarden van de overeenkomstige testfunctie.

Wanneer een LP, PD of NS gedetecteerd wordt op een curve van vaste punten is het nodig de overeenkomstige normaalvormcoëfficiënten te berekenen om deze punten te analyseren. Wat volgt is de bespreking van de normaalvormcoëfficiënten voor codim 1 bifurcaties. We doen dit voor de eenvoud voor het geval van een functie

$$f(x,\alpha) = \underbrace{f(f(f(...f}_{J \text{ keer}}(x,\alpha)...,\alpha),\alpha),\alpha) \text{ met } J = 1$$

of dus een niet-geïtereerde functie $f(x,\alpha)$, waarbij $x \in \mathbb{R}^n$ een vector van toestandsvariabelen, $\alpha \in \mathbb{R}^p$ een vector van parameters en $f : \mathbb{R}^n \times \mathbb{R}^p \longrightarrow \mathbb{R}^n$ een niet-lineaire functie is. Later maken we dan de uitbreiding voor een geïtereerde afbeelding $f^J, J > 1$.

1. Limietpuntbifurcatie

   In een limietpuntbifurcatie heeft de Jacobiaan f_x een eigenwaarde (multiplicator) $\lambda = 1$ en geen andere multiplicatoren op de eenheidscirkel.

Om een limietpuntbifurcatie te analyseren is het nodig om de normaalvormcoëfficiënt

$$a = \frac{1}{2}\langle p, B(q,q)\rangle$$

te berekenen.
Hierbij geldt dat $p, q \in \mathbb{R}^n$ respectievelijk de linkse en rechtse eigenvectoren van de Jacobiaan zijn en dus voldoen aan

$$Aq = q, \quad A^T p = p,$$

en genormaliseerd zijn volgens

$$\langle p, q\rangle = 1.$$

Om deze vectoren volledig (op een teken na) vast te leggen nemen we verder aan dat

$$\langle q, q\rangle = 1.$$

De functie $A : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ en de bilineaire functie $B : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}^n$ worden gedefinieerd door

$$A_i(x) = \sum_{j=1}^{n} \frac{\partial f_i(x_0, \alpha_0)}{\partial \xi_j} x_j \tag{2.1}$$

$$B_i(x,y) = \sum_{j,k=1}^{n} \frac{\partial^2 f_i(x_0, \alpha_0)}{\partial \xi_j \partial \xi_k} x_j y_k, \quad i = 1, 2, ..., n. \tag{2.2}$$

We kunnen de normaalvormcoëfficiënt b nu berekenen door eenvoudigweg de tweede-orde afgeleide van een scalaire functie f te berekenen. Inderdaad, we kunnen narekenen dat (zie ook 1.4)

$$B(q,q) = \frac{d^2}{d\tau^2} f(x_0 + \tau q, \alpha_0)\bigg|_{\tau=0}$$

en dus,

$$\langle p, B(q,q)\rangle = \frac{d^2}{d\tau^2}\langle p, f(x_0 + \tau q, \alpha_0)\rangle\bigg|_{\tau=0}.$$

Deze tweede afgeleide kan berekend worden door gebruik te maken van automatische differentiatie, meer bepaald door te werken met `adtayl` objecten.

2. Periodeverdubbelingsbifurcatie

In een periodeverdubbelingspunt heeft de Jacobiaan f_x een multiplicator $\lambda = -1$ en geen andere multiplicatoren op de eenheidscirkel.

Eens we een periodeverdubbelingspunt gedetecteerd hebben, kunnen we een verdere analyse van het gedrag van de curve bij de kritische parameterwaarde afleiden door de normaalvormcoëfficiënt b te berekenen, gegeven door

$$b = \frac{1}{6}\langle p, C(q,q,q) + 3B(q,(I-A)^{-1}B(q,q))\rangle, \tag{2.3}$$

waarbij I de $n \times n$-eenheidsmatrix is en p en q voldoen aan

$$Aq = -q, \quad A^T p = -p,$$

en genormaliseerd zijn volgens

$$\langle p, q\rangle = 1.$$

Verder veronderstellen we dat

$$\langle q, q\rangle = 1.$$

De bilineaire functie $B(x,y)$ werd gedefinieerd in (2.2) en de multilineaire functie $C : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}^n$ wordt gegeven door

$$C_i(x,y,z) = \sum_{j,k,l=1}^{n} \frac{\partial^3 f_i(x_0,\alpha_0)}{\partial\xi_j \partial\xi_k \partial\xi_l} x_j y_k z_l, \quad i = 1,2,...,n. \tag{2.4}$$

Afhankelijk van het teken van b in de normaalvorm kunnen we nu besluiten of we te maken hebben met een super- of subkritische periodeverdubbelingsbifurcatie. Bij de kritische parameterwaarde krijgen we een lymietcykel van periode 2. Als $b > 0$ dan is deze limietcykel stabiel en wordt deze gevonden voor die waarden van de controleparameter waarvoor het vaste punt van de afbeelding onstabiel is. Dit wordt een superkritische PD genoemd. Als $b < 0$ dan is de limietcykel onstabiel en wordt deze gevonden voor die waarden van de controleparameter waarvoor het vaste punt van de afbeelding stabiel is. Dit noemt men een subkritische PD.

3. Neimark-Sackerbifurcatie

In een Neimark-Sackerpunt heeft de Jacobiaan f_x een paar toegevoegd complexe multiplicatoren $e^{\pm i\theta_0}$ met $0 < \theta < \pi$, dus met modulus 1.

Wanneer we een Neimark-Sackerpunt detecteren kunnen we een verdere analyse van het gedrag van de bifurcatiecurve uitvoeren door de normaalvormcoëfficiënt d te berekenen. Als $c = \Re(d) \neq 0$ verschijnt er een unieke gesloten invariante curve rond het vaste punt van de afbeelding bij de gevonden kritische parameterwaarde. De normaalvormcoëfficiënt d is hierbij als volgt gedefinieerd

$$d = \frac{1}{2} e^{-i\theta_0} \langle p, C(q, q, \bar{q}) + 2B(q, h_{11}) + B(\bar{q}, h_{20}) \rangle$$

met

$$h_{11} = (I_n - A)^{-1} B(q, \bar{q}), \quad h_{20} = (e^{2i\theta_0} I_n - A)^{-1} B(q, q).$$

De linkse en rechtse eigenvectoren p en $q \in \mathbb{R}^n$ voldoen aan

$$Aq = e^{i\theta_0} q, \quad A^T p = e^{-i\theta_0} p,$$

en zijn genormaliseerd volgens

$$\langle p, q \rangle = 1,$$

en waarbij q daarnaast ook voldoet aan

$$\langle q, q \rangle = 1.$$

Als nu $c < 0$ dan ontstaat er een vertakking volgens een stabiele invariante curve in het NSpunt en dit voor controleparameterwaarden waarvoor het vaste punt van de afbeelding onstabiel is. De bifurcatie is dan superkritisch. Als anderzijs $c > 0$ dan zal de vertakking een onstabiele invariante curve zijn en dit voor waarden van de controleparameter waarvoor het vaste punt van de afbeelding stabiel is. In dit geval spreekt men van een subkritische bifurcatie.

Het berekenen van de derde-orde partiële afgeleiden bij de periodeverdubbeling en de Neimark-Sackerbifurcatie kan opnieuw herleid worden tot het berekenen van directionele afgeleiden van derde orde wegens de polarisatie-identiteiten (zie sectie 2.3.3). Door gebruik te maken

van de `adtayl` klasse in het Matlabpakket CL_MatContM kan men de tweede- en derde-orde directionele afgeleiden, nodig voor de normaalvormcoëfficiënt in (2.3), nu gemakkelijk berekenen zonder gebruik te maken van de eindige differentiemethode of symbolisch afleiden. Op die manier worden opnieuw geen afkappingsfouten gemaakt en wordt de symbolische toolbox overbodig.

2.3.2 Detectie, localisatie en analyse van codim 2 bifurcaties

Door een bifurcatiecurve corresponderend met een codim 1 bifurcatie te volgen is het mogelijk om codim 2 bifurcaties te detecteren en te localiseren. Daarvoor moeten testfuncties opgesteld worden. We zullen hier de bifurcatiecurves bij een LP-, PD- en NSpunt continueren en de mogelijke codim 2 bifurcaties overlopen.

Veronderstel dat we een limietpuntcurve continueren. We kunnen dan, net zoals voor codim 1 bifurcaties, door één van de testfuncties $\phi_1^{(1)}$, $\phi_2^{(1)}$, $\phi_3^{(1)}$ of $\phi_4^{(1)}$ te definiëren (zie [KHOS08]) één van de volgende codim 2 bifurcaties detecteren als nulwaarden van deze testfuncties

- 1:1 resonantie R1,
- Fold+Flip punt LPPD,
- Fold+Neimark-Sackerpunt LPNS,
- Cusp punt CP.

Analoog kunnen we vier codim 2 bifurcaties langs een periodeverdubbelingscurve detecteren door de juiste testfuncties $\phi_i^{(2)}, i = 1...4$ te gebruiken. Dit zijn dan de bifurcaties

- 1:2 resonantie R2,
- Fold+Flip punt LPPD,

- Flip+Neimark-Sackerpunt PDNS,
- Veralgemeend Flippunt GPD.

Ook door een Neimark-Sackercurve te continueren kunnen we codim 2 bifurcaties detecteren. Daarvoor moeten acht testfuncties $\phi_i^{(3)}, i = 1...8$ worden gedefinieerd (zie [KHOS08]). We kunnen bijgevolg acht codim 2 bifurcaties detecteren als nulwaarden van deze testfuncties

- Chencinerpunt CH,
- Flip+Neimark-Sackerpunt PDNS,
- Fold+Neimark-Sackerpunt LPNS,
- 1:1 Resonantie R1,
- Dubbel Neimark-Sackerpunt NSNS,
- 1:2 Resonantie R2,
- 1:3 Resonantie R3,
- 1:4 Resonantie R4.

We kunnen dus elf verschillende codim-2 bifurcaties detecteren voor afbeeldingen f, nl. een cusppunt, een veralgemeende flip, een Chenciner bifurcatie, een 1:1 resonantie, een 1:2 resonantie, een 1:3 resonantie, een 1:4 resonantie, een fold-flippunt, een fold-Neimark-Sackerpunt, een flip-Neimark-Sackerpunt en een dubbel Neimark-Sackerpunt.

Om singulariteiten te detecteren en dus m.a.w. de gedetecteerde codim 2 bifurcaties verder te analyseren, is het nodig om de corresponderende normaalvormcoëfficiënten te berekenen, zoals eveneens het geval was bij de codim 1 bifurcaties in de vorige sectie 2.3.1.

Hiervoor definiëren we eerst de multilineaire functies

$$D_i(x,y,z,u) = \sum_{j,k,l,m=1}^{n} \frac{\partial^3 f_i(x_0,\alpha_0)}{\partial\xi_j\partial\xi_k\partial\xi_l\partial\xi_m} x_j y_k z_l u_m, \tag{2.5}$$

$$E_i(x,y,z,u,v) = \sum_{j,k,l,m,s=1}^{n} \frac{\partial^3 f_i(x_0,\alpha_0)}{\partial\xi_j\partial\xi_k\partial\xi_l\partial\xi_m\partial\xi_s} x_j y_k z_l u_m v_s, \tag{2.6}$$

$$i = 1, 2, ..., n.$$

Zo zal het verdere gedrag van de bifurcatiecurve bij een Chencinerpunt, een veralgemeende flipbifurcatie en andere codim 2 bifurcaties afhangen van de multilineaire vormen $D(q1, q2, q3, q4)$ en $E(q1, q2, q3, q4, q5)$. Deze multilineaire vormen zijn op hun beurt afhankelijk zijn van vierde- en vijfde-orde partiële afgeleiden van het rechterlid van het dynamisch systeem in het kritische punt. De berekening van deze partiële afgeleiden kan herleid worden tot het evalueren van bepaalde directionele afgeleiden wegens de polarisatie-identiteiten (zie sectie 2.3.2). We kunnen dan opnieuw gebruik maken van automatische differentiatie in de `adtayl` klasse van Cl_MatContM om deze directionele afgeleiden van vierde en vijfde orde nauwkeurig te berekenen.

Als voorbeelden geven we hier de normaalvormcoëfficiënten voor de analyse van de bifurcatiecurve bij een cusppunt, een veralgemeende flipbifurcatie en een Chencinerpunt. Voor de normaalvormcoëfficiënten bij andere bifurcaties van codimensie 2 wordt verwezen naar [MEIJ06].

Cusp

Om verdere analyse van de bifurcatiecurve bij het detecteren van een cuspbifurcatie mogelijk te maken berekenen we de normaalvormcoëfficiënt c. Hiervoor nemen we eerst de rechtse eigenvector q en vector p als toegevoegde linkse eigenvector die voldoen aan

$$Aq = q, \quad A^T p = p.$$

Daarnaast schalen we p als volgt

$$\langle q, q\rangle = \langle p, q\rangle = 1.$$

De normaalvormcoëfficiënt c wordt dan gegeven door

$$c = \frac{1}{6}\langle p, C(q,q,q) + 3B(q,h_2),\rangle$$

waarbij

$$h_2 = (I_n - A)^{-1}B(q,q).$$

Als $c \neq 0$ dan is de cuspbifurcatie niet-gedegenereerd.

Veralgemeende flip

Ook hier kunnen we besluiten trekken uit een normaalvormcoëfficiënt voor het gedrag van de bifurcatiecurve bij de gedetecteerde veralgemeende flip. We kunnen opnieuw twee eigenvectoren p en q, resp. de linkse en rechtse, vinden die voldoen aan

$$Aq = -q, \quad A^T p = -p$$

en waarvoor

$$\langle q, q\rangle = \langle p, q\rangle = 1.$$

Als we nu achtereenvolgens coëfficiënten h_2, h_3 en h_4 definiëren als volgt

$$\begin{aligned} h_2 &= (I_n - A)^{-1}B(q,q), \\ h_3 &= -(A + I_n)^{-1}(C(q,q,q) + 3B(q,h_2)), \\ h_4 &= (I_n - A)^{-1}(4B(q,h_3) + 3B(h_2,h_2) + 6C(q,q,h_2) + D(q,q,q,q)), \end{aligned}$$

dan wordt de kritieke normaalvormcoëfficiënt

$$\begin{aligned} c &= \frac{1}{120}\langle p, 5B(q,h_4) + 10C(q,q,h_3) + 10B(h_2,h_3) + 15C(q,h_2,h_2) \\ &\quad +10D(q,q,q,h_2) + E(q,q,q,q,q)\rangle. \end{aligned}$$

Als $c \neq 0$ dan is de veralgemeende flip niet-gedegenereerd.

Chencinerpunt

Eens we een Chencinerpunt gedetecteerd hebben, kunnen we het verdere verloop van de curve bij de kritische parameterwaarde analyseren door twee normaalvormcoëfficiënten c_1 en c_2 te berekenen en te interpreteren. We kiezen eerst de complexe eigenvectoren p en $q \in \mathbb{R}^n$ die voldoen aan

$$\begin{aligned} Aq &= e^{i\theta_0}q, \quad A\bar{q} = e^{-i\theta_0}\bar{q}, \\ A^T p &= e^{-i\theta_0}p, \quad A^T\bar{p} = e^{i\theta_0}\bar{p}, \end{aligned}$$

en waarvoor verder ook geldt dat

$$\langle q, q\rangle = \langle p, q\rangle = 1.$$

De normaalvormcoëfficiënten c_1 en c_2 worden dan als volgt gedefinieerd (zie [KUZ04] en [MEIJ06])

$$\begin{aligned} c_1 &= \frac{1}{2}\langle p, C(q,q,\bar{q}) + B(\bar{q}, (e^{2i\theta_0}I_n - A)^{-1}B(q,q)) + 2B(q, (I_n - A)^{-1}B(q,\bar{q}))\rangle \\ c_2 &= \frac{1}{12}\langle p, E(q,q,q,\bar{q},\bar{q}) + D(q,q,q,h_{02}) + 6D(q,q,\bar{q},h_{11}) + 3D(q,\bar{q},\bar{q},h_{20}) \\ &\quad +3C(q,h_{20},h_{02}) + 6C(q,h_{11},h_{11}) + 3C(q,q,h_{12}) + 6C(q,\bar{q},h_{21}) \\ &\quad +6C(\bar{q},h_{11},h_{20}) + C(\bar{q},\bar{q},h_{30}) + 3B(h_{20},h_{12}) + 6B(h_{11},h_{21}) \\ &\quad +3B(q,h_{22}) + B(h_{02},h_{30}) + 2B(\bar{q},h_{31})\rangle. \end{aligned}$$

Hierin zijn de vectoren $h_{20}, h_{11}, h_{02}, h_{21}, h_{12}, h_{22}, h_{31}$ en h_{31} oplossingen van lineaire stelsel van vergelijking waarin opnieuw de multilineaire vormen A, B, C, D en E voorkomen.

De Chenciner bifurcatie is niet-gedegenereerd als de tweede Lyapunovcoëfficiënt $l_2 = \Re(e^{-i\theta_0}c_2) + \frac{1}{2}\Im(e^{-i\theta_0}c_1)^2$ niet-nul is.

2.3.3 Polarisatie-identiteiten

In de hierboven gestelde voorbeelden werd het berekenen van de partiële afgeleiden in de multilineaire functies telkens herleid tot het berekenen van

directionele afgeleiden van corresponderende orde. Hiervoor moeten we eerst opmerken dat we de multilineaire functies $B(x,y)$, $C(x,y,z)$, $D(x,y,z,u)$ en $E(x,y,z,u,v)$ kunnen evalueren als directionele afgeleiden in een reeks *gelijke* reële vectorargumenten. We kunnen namelijk, zoals eerder al vermeld, voor de vector $B(v,v)$, met $v \in \mathbb{R}^n$ de volgende formule neerschrijven

$$B(v,v) = \frac{d^2}{d\tau^2} f(x_0 + \tau q, \alpha_0)\Big|_{\tau=0}. \tag{2.7}$$

Ook voor vector $C(v,v,v)$ kunnen we een gelijkaardige formule opstellen

$$C(v,v,v) = \frac{d^3}{d\tau^3} f(x_0 + \tau q, \alpha_0)\Big|_{\tau=0}. \tag{2.8}$$

Deze afgeleiden kunnen we nauwkeurig berekenen door gebruik te maken van automatische differentiatie. Als we dus nu de multilineaire vormen in de normaalvormcoëfficiënten in bovenstaande voorbeelden kunnen herleiden tot uitdrukkingen zoals in (2.7) en (2.8) waarin gelijke vectorargumenten optreden worden de berekening stukken vereenvoudigd doordat er in dat geval gebruik kan gemaakt worden van de `adtayl` klasse in CL_MatContM.

Voor het berekenen van de normaalvormcoëfficiënten voor een Neimark-Sackerpunt en een Chencinerpunt is het nodig om een onderscheid te maken tussen het reële deel het imaginaire deel van de eigenvector q.

Noteren we de reële en imaginaire delen van eigenvector q als q_R en q_I respectievelijk

$$q = q_R + iq_I, \quad q \in \mathbb{C}^n, \quad q_R, q_I \in \mathbb{R}^n,$$

dan krijgen we

$$\begin{aligned}
B(q,q) &= B(q_R + iq_I, q_R + iq_I) = B(q_R,q_R) + 2iB(q_R,q_I) - B(q_I,q_I) \\
B(q,\bar{q}) &= B(q_R + iq_I, q_R - iq_I) = B(q_R,q_R) + B(q_I,q_I) \\
C(q,q,\bar{q}) &= C(q_R + iq_I, q_R + iq_I, q_R - iq_I) \\
&= C(q_R,q_R,q_R) + iC(q_R,q_R,q_I) + C(q_R,q_I,q_I) + iC(q_I,q_I,q_I).
\end{aligned}$$

De vectoren $B(q_R,q_R)$, $B(q_I,q_I)$, $C(q_R,q_R,q_R)$ en $C(q_I,q_I,q_I)$ kunnen onmiddelijk berekend worden aan de hand van formules (2.7) en (2.8). Voor de

andere vectoren $B(q_R, q_I)$, $C(q_R, q_R, q_I)$ en $C(q_R, q_I, q_I)$ moeten we dus een oplossing zoeken. Uiteindelijk is het voldoende om een methode te vinden die het mogelijk maakt om multilineaire functies $B(v, w)$ en $C(v, v, w)$ te berekenen voor reële vectoren $v, w \in \mathbb{R}^n$ gebruik makend van (2.7) en (2.8). Daarvoor beschouwen we de vectoren

$$\begin{aligned} B(v+w, v+w) &= B(v,v) + 2B(v,w) + B(w,w), \\ B(v-w, v-w) &= B(v,v) - 2B(v,w) + B(w,w). \end{aligned}$$

Door beide identiteiten van elkaar af te trekken vinden we een uitdrukking voor $B(v, w)$ die alleen afhangt van vectoren met gelijke vectorargumenten

$$B(v,w) = \frac{1}{4}[B(v+w, v+w) - B(v-w, v-w)]. \tag{2.9}$$

Identiteit (2.9) noemt met de eerste *polarisatie-identiteit*. Ook voor de multilineaire vorm $C(v, v, w)$ kunnen we een dergelijke uitdrukking vinden. We beschouwen hiertoe de vectoren

$$\begin{aligned} C(v+w, v+w, v+w) &= C(v,v,v) + 3C(v,v,w) + 3C(v,w,w) + C(w,w,w), \\ C(v-w, v-w, v-w) &= C(v,v,v) - 3C(v,v,w) + 3C(v,w,w) - C(w,w,w). \end{aligned}$$

Aftrekken van deze identiteiten leidt dan tot

$$C(v,v,w) = \frac{1}{6}[C(v+w, v+w, v+w) - C(v-w, v-w, v-w)] - \frac{1}{3}C(w,w,w). \tag{2.10}$$

Dit is nog niet de tweede polarisatie-identiteit maar is wel al bruikbaar voor de berekening van de normaalvormcoëfficiënt d bij een Neimark-Sackerbifurcatie. We kunnen dit dan doen aan de hand van automatische differentiatie van Taylorpolynomen. De tweede polarisatie-identiteit geeft een uitdrukking voor $C(u, v, w)$ met $u, v, w \in \mathbb{R}^n$ in termen van vectoren met gelijke vectorargumenten.

Stelling 2.1. *Voor de multilineaire functie $C(u, v, w)$ zoals gedefinieerd in*

(2.4) geldt dat

$$\begin{aligned} C(u,v,w) = \frac{1}{24}\Big[\ & C(u+v+w, u+v+w, u+v+w) \\ & - C(u+v-w, u+v-w, u+v-w) \\ & - C(u-v+w, u-v+w, u-v+w) \\ & + C(u-v-w, u-v-w, u-v-w)\Big]. \end{aligned} \tag{2.11}$$

Bewijs.

We beschouwen eerst de vectoren

$$\begin{aligned} & C(u+v+w, u+v+w, u+v+w) \\ & \quad = C(u,u,u) + C(v,v,v) + C(w,w,w) + 3C(u,u,v) + 3C(u,u,w) \\ & \quad + 3(u,v,v) + 3C(u,w,w) + 3C(v,v,w) + 3C(v,w,w) + 6C(u,v,w), \end{aligned} \tag{2.12}$$

$$\begin{aligned} & C(u+v-w, u+v-w, u+v-w) \\ & \quad = C(u,u,u) + C(v,v,v) - C(w,w,w) + 3C(u,u,v) - 3C(u,u,w) \\ & \quad + 3(u,v,v) + 3C(u,w,w) - 3C(v,v,w) + 3C(v,w,w) - 6C(u,v,w), \end{aligned} \tag{2.13}$$

$$\begin{aligned} & C(u-v+w, u-v+w, u-v+w) \\ & \quad = C(u,u,u) - C(v,v,v) + C(w,w,w) - 3C(u,u,v) + 3C(u,u,w) \\ & \quad + 3(u,v,v) + 3C(u,w,w) + 3C(v,v,w) - 3C(v,w,w) - 6C(u,v,w), \end{aligned} \tag{2.14}$$

en

$$\begin{aligned} & C(u-v-w, u-v-w, u-v-w) \\ & \quad = C(u,u,u) - C(v,v,v) - C(w,w,w) - 3C(u,u,v) - 3C(u,u,w) \\ & \quad + 3(u,v,v) + 3C(u,w,w) - 3C(v,v,w) - 3C(v,w,w) + 6C(u,v,w). \end{aligned} \tag{2.15}$$

Als we nu (2.12) en (2.13) van elkaar aftrekken dan vinden we

$$\begin{aligned} &C(u+v+w, u+v+w, u+v+w) - C(u+v-w, u+v-w, u+v-w) \\ &\quad = 2C(w,w,w) + 6C(u,u,w) + 6(v,v,w) + 12C(u,v,w). \end{aligned} \tag{2.16}$$

Aftrekken van (2.14) en (2.15) geeft anderzijds

$$\begin{aligned} &C(u-v+w, u-v+w, u-v+w) - C(u-v-w, u-v-w, u-v-w) \\ &\quad = 2C(w,w,w) + 6C(u,u,w) + 6(v,v,w) - 12C(u,v,w). \end{aligned} \tag{2.17}$$

Door (2.16) en (2.17) vervolgens van elkaar af te trekken vinden we uiteindelijk de tweede polarisatie-identiteit (2.11). □

Door dezelfde redeneringen te maken vinden we ook voor de andere multilineaire functies $D(u,v,w,y)$ en $E(u,v,w,y,z)$ polarisatie-indentiteiten.

Stelling 2.2. *Voor de multilineaire functie $D(u,v,w,y)$ zoals gedefinieerd in (2.5) geldt dat*

$$\begin{aligned} D(u,v,w,y) = \frac{1}{192}\Big[\; & D(u+v+w+y,...) - D(u+v+w-y,...) \\ & + D(u+v-w-y,...) - D(u+v-w-y,...) \\ & - D(u-v+w+y,...) + D(u-v+w-y,...) \\ & - D(u-v-w-y,...) + D(u-v-w+y,...)\Big]. \end{aligned} \tag{2.18}$$

Voor de multilineaire functie $E(u,v,w,y,z)$ zoals gedefinieerd in (2.6) geldt dat

$$
\begin{aligned}
&E(u,v,w,y,z) \\
&\quad = \frac{1}{1920}\Big[\ E(u+v+w+y+z,...) - E(u+v+w+y-z,...) \\
&\qquad + E(u+v+w-y-z,...) - E(u+v+w-y+z,...) \\
&\qquad - E(u+v-w+y+z,...) + E(u+v-w+y-z,...) \\
&\qquad - E(u+v-w-y-z,...) + E(u+v-w-y+z,...) \\
&\qquad - E(u-v+w+y+z,...) + E(u-v+w+y-z,...) \\
&\qquad - E(u-v+w-y-z,...) + E(u-v+w-y+z,...) \\
&\qquad + E(u-v-w+y+z,...) + E(u-v-w+y-z,...) \\
&\qquad + E(u-v-w-y-z,...) + E(u-v-w-y+z,...)\Big]\,.
\end{aligned}
\tag{2.19}
$$

Samen met uitdrukkingen (2.7), (2.8),

$$
D(v,v,v,v) = \frac{d^4}{d\tau^4} f(x_0+\tau q,\alpha_0)\bigg|_{\tau=0} \tag{2.20}
$$

en

$$
E(v,v,v,v,v) = \frac{d^5}{d\tau^5} f(x_0+\tau q,\alpha_0)\bigg|_{\tau=0} \tag{2.21}
$$

kunnen we de polarisatie-identiteiten dus gebruiken om normaalvormcoëfficiënten te berekenen aan de hand van automatische differentiatie van Taylorpolynomen. Op die manier kunnen we gedetecteerde bifurcaties verder analyseren. Het teken en de grootte van de normaalvormcoëfficiënten bepaalt namelijk het bifurcatiescenario bij een lokaal bifurcatiepunt, cf. [GOV07, KUZ04, KUZ05].

In deze thesis werd al eerder vermeld dat gewerkt zou worden met geïtereerde functies f^J. Om een bifurcatieanalyse van een geïtereerde functie uit te voeren moeten we analoog als hierboven de multilineaire vormen berekenen voor deze geïtereerde functie. Deze multilineaire vormen van een functie f^J

worden dan als volgt gedefinieerd

$$A^J(x) = \sum_{j=1}^{n} \frac{\partial f^J(x_0,\alpha_0)}{\partial \xi_j} x_j \tag{2.22}$$

$$B^J(x,y) = \sum_{j,k=1}^{n} \frac{\partial^2 f^J(x_0,\alpha_0)}{\partial \xi_j \partial \xi_k} x_j y_k, \tag{2.23}$$

$$C^J(x,y,z) = \sum_{j,k,l=1}^{n} \frac{\partial^3 f^J(x_0,\alpha_0)}{\partial \xi_j \partial \xi_k \partial \xi_l} x_j y_k z_l, \tag{2.24}$$

$$D^J(x,y,z,u) = \sum_{j,k,l,m=1}^{n} \frac{\partial^4 f^J(x_0,\alpha_0)}{\partial \xi_j \partial \xi_k \partial \xi_l \partial \xi_m} x_j y_k z_l u_m, \tag{2.25}$$

$$E^J(x,y,z,u,v) = \sum_{j,k,l,m,s=1}^{n} \frac{\partial^5 f^J(x_0,\alpha_0)}{\partial \xi_j \partial \xi_k \partial \xi_l \partial \xi_m \partial \xi_s} x_j y_k z_l u_m v_s, \tag{2.26}$$

voor $J = 1, 2, ...$
We kunnen de componenten van deze multilineaire vormen ook afzonderlijk beschouwen

$$(A^J)_{i,j} = \frac{\partial (f^J(x_0,\alpha_0))_i}{\partial \xi_j} \tag{2.27}$$

$$(B^J)_{i,j,k} = \frac{\partial^2 (f^J(x_0,\alpha_0))_i}{\partial \xi_j \partial \xi_k}, \tag{2.28}$$

$$(C^J)_{i,j,k,l} = \frac{\partial^3 (f^J(x_0,\alpha_0))_i}{\partial \xi_j \partial \xi_k \partial \xi_l}, \tag{2.29}$$

$$(D^J)_{i,j,k,l,m,s} = \frac{\partial^4 (f^J(x_0,\alpha_0)_i}{\partial \xi_j \partial \xi_k \partial \xi_l \partial \xi_m}, \tag{2.30}$$

$$(E^J)_{i,j,k,l,m,s,r} = \frac{\partial^5 (f^J(x_0,\alpha_0))_i}{\partial \xi_j \partial \xi_k \partial \xi_l \partial \xi_m \partial \xi_s}, \tag{2.31}$$

Veronderstellen we dat de functie f^J voldoende glad is dan hebben we de

uitdrukking

$$\begin{aligned} f^J(x_0+u,\alpha_0) &= x_0 + A^J(u) + \frac{1}{2}B^J(u,u) + \frac{1}{6}C^J(u,u,u) \\ &\quad + \frac{1}{24}D^J(u,u,u,u) + \frac{1}{120}E^J(u,u,u,u,u) + O(||u||^6). \end{aligned}$$

In deze uitdrukking zien we enkel gelijke vectorargumenten optreden bij de multilineaire vormen. Dankzij de hierboven besproken polarisatie-identiteiten kunnen we dus de normaalvormcoëfficiënten bepalen voor geïtereerde functies door AD op Taylorpolynomen en dus gebruik te maken van de `adtayl` klasse in Cl_MatContM.

Bij de voorbeelden van codim 1 en en codim 2 bifurcaties kunnen we in de overeenkomstige normaalvormcoëfficiënten de multilineaire vormen A, B, C, D en E vervangen door de hierboven gedefinieerde multilineaire vormen indien we te maken hebben met het geval van een geïtereerde functie f^J.

Hoofdstuk 3

Implementatie in Cl_MatContM

3.1 Inleiding

De techniek voor AD bestaat erin om een gegeven computercode eerst om te vormen naar een "straight-line" code, eerder de codelijst genoemd. Dit wil zeggen dat de code geschreven wordt als een eindige reeks elementaire bewerkingen zonder daarin lussen, voorwaarden, subroutines of takken.
Beschouwen we bijvoorbeeld een functie $f : \mathbb{R}^5 \longrightarrow \mathbb{R}^5$ die een vijfdimensionale vector x afbeeldt op een vector y met dezelfde dimensie. Neem voor de eenvoud aan dat de corresponderende straight-line code een vector $x(1:5)$ als input neemt en een output $y(1:5)$ produceert, door gebruik te maken van een achtdimensionale vector $t(1:8)$ die de intermediaire variabelen opslaat. De straight-line code van dit voorbeeld wordt gegeven in tabel 3.1. In deze tabel krijgen de onafhanklijke variabelen $x(i)$ initiële waarde c_i en stelt het symbool $\odot$ een binaire elementaire bewerking voor, zoals vermenigvuldiging of optelling. Om van de onafhankelijke inputvariabelen naar de afhankelijke outputvariabelen te stappen kan voor de toepassing waarvan sprake in deze thesis gebruik gemaakt worden van automatische differentiatie. Zo zullen de inputvariabelen vectoren $q_1, ...q_5$ of Taylorpolynomen zijn en de outputvariabelen de multilineaire vormen. We werken met Taylorpolynomen

Tabel 3.1: De straight-line code voor een eenvoudige functie met $n = 5$ onafhankelijke scalaire variabelen $x(1)$, $x(2)$,..., $x(5)$ en $m = 5$ afhankelijke scalaire variabelen $y(1)$, $y(2)$,..., $y(5)$.

(voorbeeld uit [BUCK05])

	$t(1) \longleftarrow x(1) \odot x(2)$	
$x(1) \longleftarrow c_1$	$t(2) \longleftarrow t(1) \odot x(3)$	$y(1) \longleftarrow t(2) \odot t(2)$
$x(2) \longleftarrow c_2$	$t(3) \longleftarrow x(2) \odot x(4)$	$y(2) \longleftarrow t(2) \odot t(7)$
$x(3) \longleftarrow c_3$	$t(4) \longleftarrow x(2) \odot x(5)$	$y(3) \longleftarrow y(2) \odot t(7)$
$x(4) \longleftarrow c_4$	$t(5) \longleftarrow t(3) \odot t(4)$	$y(4) \longleftarrow t(6) \odot t(8)$
$x(5) \longleftarrow c_5$	$t(6) \longleftarrow t(2) \odot t(5)$	$y(5) \longleftarrow t(5) \odot t(8)$
	$t(7) \longleftarrow t(2) \odot t(6)$	
	$t(8) \longleftarrow t(5) \odot t(7)$	

omdat Taylorcoëfficiënten makkelijker te manipuleren zijn dan de beoogde afgeleiden zelf. Om AD te kunnen toepassen op de Taylorpolynomen werd elke elementaire rekenkundige bewerking en elke operator overschreven. Dit houdt in dat we elke bewerking die we normaal toepassen op een reëel argument vervangen door zijn overeenkomstige bewerking toepasbaar op een Taylorpolynoom.

Deze overschreven codes kunnen worden teruggevonden in de `adtayl` klasse van het Matlabpakket Cl_MatContM. Een object van dit `adtayl` type kunnen we, zoals eerder aangehaald, zien als een object dat een oneindige machtsreeks tot orde p voorstelt. Een `adtayl` object bevat dan naast de waarde van de oorspronkelijke inputvariabele ook de Taylorcoëfficiënten van de functie in kwestie geëvalueerd in deze inputvariabelen. In de Matlabcodes voor de binaire elementaire bewerkingen $+, -, \times, \div$ vinden we verschillende van de gevonden algoritmes uit sectie 1.2.2 terug. Zo ook in het voorbeeld in sectie 2.2.
Om standaardoperatoren zoals sin en cos,... toe te passen op machtsreeksen zoals hier het geval is, zijn in de literatuur meerdere algoritmes beschikbaar, o.a. deze in tabel 1.2. We hebben echter gekozen voor een algoritme dat redelijke snel geïmplementeerd en uitgevoerd kan worden in Matlab, voornamelijk als het argument een vector van machtreeksen is. We geven hier enkele voorbeelden.

1. We kunnen voor de vermenigvuldiging van `adtayl`objecten gebruik maken van het feit dat dit een convolutie is en we dus de ingebouwde Matlabfunctie `filter` kunnen toepassen.

```
function y = times(x,y)
% Elementwise multiplication x.*y for ADTAYL objects.
% Either of x or y can be numeric, and either can be scalar
% (1 by 1).
% If neither is scalar, they must have the same size.
% In effect, a numeric argument is converted to ADTAYL,
% and a scalar argument is "spread" to the size of the other.
% The method uses the FILTER function. The case when either
% argument is a scalar runs fastest because one can do it
% with just one call to FILTER.

x = adtayl(x);
y = adtayl(y);
if isscalar(x) % X is a scalar ADTAYL, do a FILTER on all Y
               % at once
  [m,n,p] = size(y.tc);
  mn = m*n;
  x.tc = reshape(x.tc,1,p);
  y.tc = reshape(y.tc,mn,p);
  y.tc = filter(x.tc, 1, y.tc, [],2); %along the 2nd %
            % dimension
elseif isscalar(y) % Y is a scalar ADTAYL, do a FILTER on
            % all X at once
  [m,n,p] = size(x.tc);
  mn = m*n;
  x.tc = reshape(x.tc,mn,p);
  y.tc = reshape(y.tc,1,p);
  y.tc = filter(y.tc, 1, x.tc, [],2); % along the 2nd
            %   dimension
elseif ~isequal(size(x),size(y)) %ADTAYL size, just the
               % [m n]dimensions
  error('Arguments must have equal sizes')
else
  [m,n,p] = size(x.tc);
  mn = m*n;
  x.tc = reshape(x.tc,mn,p);
  y.tc = reshape(y.tc,mn,p);
  for i=1:mn
    y.tc(i,:) = filter(x.tc(i,:), 1, y.tc(i,:));
  end
end
```

```
42 y.tc = reshape(y.tc,m,n,p);
```

Deze ingebouwde `filter` functie werkt als volgt:

Beschouwen we een polynoom of getrunceerde machtreeks `A` opgeslagen als een lijst van coëfficiënten $a_0, a_1, ..., a_p$ in een vector van lengte $p+1$. Dan kunnen we `Y=filter(B,A,X)` eenvoudig zien als

$$\texttt{Y} = \frac{\texttt{polynoom B}}{\texttt{polynoom A}}(\texttt{matrix van polynomen X})$$

zodat output `Y` niet alleen dezelfde orde als `X` heeft, maar ook dezelfde dimensies.

In de Matlab-helpfile voor de `filter` functie wordt exacter vermeld dat:

`Y = filter(B,A,X)` filtert de data in vector `X` met de filter beschreven door de coëfficiëntenvector `B` in de teller en de coëfficiëntenvector `A` in de noemer. Als `A(1)` niet gelijk is aan 1, dan worden de filtercoëfficiënten genormaliseerd met `A(1)`. Als `A(1)` gelijk is aan 0, wordt er een foutmelding gegeven.

De filtering gebeurt volgens de betrekking

$$\begin{aligned} Y(n) =& B(1)X(n) + B(2)X(n-1) + ... + B(n_B+1)X(n-n_B) \\ & - A(2)Y(n-1) - ... - A(n_A+1)Y(n-n_A) \end{aligned}$$

waarbij $n-1$ de filterorde is, n_A de orde van het polynoom voorgesteld in vector `A` en n_B de orde van het polynoom voorgesteld in vector `B`. De output `Y(n)` is dan een lineaire combinatie van de huidige en vorige inputs `X(n), X(n-1),...` en de vorige outputs `Y(n-1), Y(n-2),...` en wordt bijgevolg recursief gedefinieerd en berekend.

Als `X` een matrix is, zal `filter` automatisch inwerken op de kolommen van `X`. Als `X` een multidimensionale matrix is, zal `filter` inwerken op de eerste niet-singletondimensie.

Men kan verder met `[...] = filter(b,a,X,[],dim)` ook de dimensie als inputargument toevoegen om `filter` op de gewenste dimensie te laten inwerken. Als inputvector `X` een vector of matrix is en dim is 1, dan werkt `filter` in op de kolommen van `X`. Als dim 2 is, zal filter inwerken op de rijen van `X`.

Het bovenstaande in beschouwing nemend is het dus duidelijk dat als `A=1`, de `filter` functie een convolutie uitvoert van de inputvectoren `X` en `B` zodanig dat de lengte van de outputvector dezelfde als de lengte van de inputvector `X`.
We kunnen de Matlabfunctie `filter` dan vergelijken met de Matlabfunctie `conv` die de twee vectoren gegeven als argument convoluteert.

Bijvoorbeeld als $X = [1, 2, 3, 4]$ en $B = [5, 6]$, dan:

`conv(X,B) = [5 16 27 38 24],`

`conv(B,X) = [5 16 27 38 24],`

`filter(B,1,X) = [5 16 27 38],`

`filter(X,1,B) = [5 16].`

2. Bij de methode voor exp(`a`) gebruiken we het feit dat als $c(t) = \exp(a(t))$ we dan de ODE $c'(t) = a'(t)c(t)$ krijgen. Definiëren we nu eerst de lineaire operator T als volgt

$$(Tc)(t) = \int_0^t a'(s)c(s)ds,$$

dan kunnen we deze ODE wegens de hoofdstelling van de integraalrekening in integraalvorm schrijven als

$$c(t) = c_0 + (Tc)(t), \quad \text{of dus} \quad (I - T)c(t) = c_0$$

met $c_0 = \exp(a_0)$.
Om de $c(t)$ te vinden, kunnen we deze integraalvorm in Matlab oplossen met de ingebouwde `mldivide` functie

$$c(t) = \texttt{mldivide}(I - T, c_0) = (I - T)\backslash c_0.$$

De functie werkt als volgt:

- Als A een $n \times n$-matrix is en B een kolomvector met n elementen, of een matrix met zulke kolommen, dan is $X = A\backslash B$ de oplossing van de vergelijking $AX = B$ berekend met de Gausseliminatiemethode met partieel pivoteren. Als A bijna singulier is, wordt een waarschuwing gegeven.

- Als A een $m \times n$-matrix is met $m \neq n$ en B is een kolomvector met m componenten of een matrix met zulke kolommen, dan is $X = A\backslash B$ de oplossing in kleinste-kwadraten-zin van onder- of overgedetermineerde systeem van vergelijkingen $AX = B$. M.a.w. zal X de norm of dus de lengte van de vector $AX - B$ minimaliseren. De rang k van A wordt hier bepaald door een QR-decompositie met kolompivoteren. De oplossing X heeft dan maximaal k niet-nul elementen per kolom.

Om deze procedure nog sneller te laten verlopen en het aantal berekening in Matlab te verminderen wordt de ODE eerst omgezet naar de vorm

$$c(t) = (K - L)\backslash(Ky_0)$$

met $K = diag([1, 1 : p])$ de diagonaalmatrix met als eerste element op de diagonaal 1 en als volgende diagonaalelementen de vector $[1, 2, 3, ..., p]$ met p de orde van Taylorreeks en L de strikt onder-triangulaire Toeplitz matrix met als eerste kolom $[0, 1a(1), 2a(2), ..., pa(p)]$.
Een Toeplitzmatrix is een matrix waarbij elk element van elke dalende diagonaal constant is. De matrix

$$\begin{bmatrix} a_0 & a_{-1} & a_{-2} & \dots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \dots & a_{-n} \\ a_2 & a_1 & a_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} \\ a_{n-1} & \dots & a_2 & a_1 & a_0 \end{bmatrix}$$

is een Toeplitz matrix.
Een matrixvergelijking van de vorm

$$\sum_{j=1}^{n} a_{i-j} x_j = y_i$$

waarbij de coëfficiëntenmatrix een Toeplitzmatrix is, kan worden opgelost met $O(n^2)$ bewerkingen.

```
function x = exp(x)
% Compute exponential of Taylor series, for each element of
% adtayl array.
[m,n,p1] = size(x.tc);
```

```
p = p1-1;
mn = m*n;
x.tc = reshape(x.tc, mn,p1);
exp0 = exp(x.tc(:,1)); %vector of exponentials of constant
% terms
Kdiag = [1  1:p];
z = zeros(p,1);
for i=1:mn
  % Form derivative of x.tc regarded as Taylor series:
  dx = x.tc(i,2:end).*(1:p); %of length p, not p+1.
  L = toeplitz([0 dx], zeros(1,p1));
  x.tc(i,:) = ((diag(Kdiag) - L) \ [exp0(i);z]).'; %want a
%row vector as output
end
x.tc = reshape(x.tc, m,n,p1);
```

3. Zoals reeds vermeld is het efficiënter om sin(a) en cos(a) tegelijkertijd te berekenen en op te slaan. In de literatuur zijn hier verschillende algoritmen voor beschikbaar zoals het algoritme in tabel 1.2. Voor de implementatie in deze toepassing werd dit gedaan door deze functies te zien als het reële en imaginaire deel van $\exp ia$. Op die manier kan men bij een nieuwe gegeven a eerst nagaan of één van beide functies reeds werd berekend voor a en deze weergeven indien dit het geval is zonder nieuwe berekeningen te hoeven maken. Dit werd als volgt geïmplementeerd

```
function x = sin(x)
% SIN for adtayl objects.

global adtayl_CSSAV adtayl_CSARG

if ~isreal(x.tc)
  error('COS, SIN for adtayl objects must not have
  imaginary part')
end

% Compute and store COS and SIN together; or check to see if
% already computed.
if isequal(x.tc, adtayl_CSARG)
  x.tc = adtayl_CSSAV;
else
  adtayl_CSARG = x.tc;
  x = exp(i*x); % a adtayl object
  adtayl_CSSAV = x.tc; % a complex array
```

```
19 end
20 x.tc = imag(x.tc);
```

3.2 Vergelijking Symbolische en Automatische Differentiatie

3.2.1 Recursieformules voor afgeleiden tot 5de orde

Als de symbolische toolbox beschikbaar is in Matlab, dan kunnen we de nodige afgeleiden van een geïtereerde functie f^J berekenen aan de hand van recursieve formules waarvan we hier een kort overzicht geven. Door een functie f K keer toe te passen op een beginargument x_1 bekomen we een K+1-tupel

$$(x_1, x_2, x_3, ..., x_{K+1}).$$

Veronderstel dat we in elk punt de symbolische afgeleiden van f tot orde 5 kunnen berekenen. De recursieve formules voor de afgeleiden worden dan bekomen door toepassing van de kettingregel. Voor $J = 1$ hebben we $A^1 = A(x_1)$, $B^1 = B(x_1)$, $C^1 = C(x_1)$, $D^1 = D(x_1)$ en $E^1 = E(x_1)$, dewelke gekend zijn.

Verder is

$$\begin{aligned}
A^J_{i,j} &= \sum_k \frac{\partial f_i}{\partial x_k}(f^{J-1}(x_1))\frac{\partial (f^{J-1}(x_1))_k}{\partial x_j} \\
&= \sum_k (A(x_J))_{i,k} A^{j-1}_{k,j} \\
&= \sum_k (A(x_J))_{i,k} \sum_l (A(x_{J-1}))_{k,l} A^{J-2}_{l,j} \\
&\vdots \\
&= (A(x_J)A(x_{J-1})...A(x_1))_{i,j}.
\end{aligned}$$

Bijgevolg is dus voor $J = K$

$$A^K q = A(x_K)A(x_{K-1})A(x_{K-2})...A(x_1)q. \tag{3.1}$$

Voor de multilineaire vorm B hebben we dat

$$\begin{aligned}
B^J_{i,j,k} &= \frac{\partial^2 f_i(f^{J-1}(x_1))}{\partial x_j \partial x_k} \\
&= \frac{\partial}{\partial x_j}\left(\frac{\partial}{\partial x_k} f_i(f^{J-1}(x_1))\right) \\
&= \frac{\partial}{\partial x_j}\left(\sum_l \frac{\partial f_i}{\partial x_l}(x_J)\frac{\partial (f^{J-1}(x_1))_l}{\partial x_k}\right) \\
&= \sum_{l,m} \frac{\partial^2 f_i}{\partial x_l \partial x_m}(x_J)\frac{\partial (f^{J-1}(x_1))_m}{\partial x_j}\frac{\partial (f^{J-1}(x_1))_l}{\partial x_k} + \sum_l \frac{\partial f_i}{\partial x_l}(x_J)\frac{\partial^2 (f^{J-1}(x_1))_l}{\partial x_j \partial x_k} \\
&= \sum_{l,m} B_{i,l,m}(x_J)A^{J-1}_{m,j}A^{J-1}_{l,k} + \sum_l A_{i,l}(x_J)B^{J-1}_{l,j,k} \\
&= (B(x_J)A^{J-1}A^{J-1} + A(x_J)B^{J-1})_{i,j,k}
\end{aligned}$$

Bijgevolg kunnen we voor twee vectoren q_1 en q_2 de vorige uitdrukking vermenigvuldigen met $(q_1)_j(q_2)_k$ en sommeren over (j,k) en krijgen we

$$B^J(q_1, q_2) = B(x_J)(A^{J-1}q_1, A^{J-1}q_2) + A(x_J)B^{J-1}(q_1, q_2). \tag{3.2}$$

We kennen $A(x_J)$ en $B(x_J)$ dus we kunnen de multilineaire vorm $B^K(q_1, q_2)$, $K = 1, ..., J$ recursief berekenen.

Voor de multilineaire vorm C hebben we

$$
\begin{aligned}
C^J_{i,j,k,d} &= \frac{\partial^3 f_i(f^{J-1}(x_1))}{\partial x_j \partial x_k \partial x_d} \\
&= \frac{\partial}{\partial x_d}\left(\frac{\partial}{\partial x_j}\left(\frac{\partial}{\partial x_k} f_i(f^{J-1}(x_1))\right)\right. \\
&= \frac{\partial}{\partial x_d}\left(\sum_{l,m} \frac{\partial^2 f_i}{\partial x_l \partial x_m}(x_J)\frac{\partial (f^{J-1}(x_1))_m}{\partial x_j}\frac{\partial (f^{J-1}(x_1))_l}{\partial x_k}\right. \\
&\quad \left.+ \sum_l \frac{\partial f_i}{\partial x_l}(x_J)\frac{\partial^2 (f^{J-1}(x_1))_l}{\partial x_j \partial x_k}\right) \\
&= \sum_{l,m,r} \frac{\partial^3 f_i}{\partial x_l \partial x_m \partial x_r}(x_J)\frac{\partial (f^{J-1}(x_1))_l}{\partial x_k}\frac{\partial (f^{J-1}(x_1))_m}{\partial x_j}\frac{\partial (f^{J-1}(x_1))_r}{\partial x_d} \\
&\quad + \sum_{l,m} \frac{\partial^2 f_i}{\partial x_l \partial x_m}(x_J)\frac{\partial^2 (f^{J-1}(x_1))_l}{\partial x_d \partial x_k}\frac{\partial (f^{J-1}(x_1))_m}{\partial x_j} \\
&\quad + \sum_{l,m} \frac{\partial^2 f_i}{\partial x_l \partial x_m}(x_J)\frac{\partial (f^{J-1}(x_1))_l}{\partial x_k}\frac{\partial^2 (f^{J-1}(x_1))_m}{\partial x_d \partial x_j} \\
&\quad + \sum_{l,m} \frac{\partial^2 f_i}{\partial x_l \partial x_m}(x_J)\frac{\partial^2 (f^{J-1}(x_1))_l}{\partial x_j \partial x_k}\frac{\partial (f^{J-1}(x_1))_m}{\partial x_d} \\
&\quad + \sum_l \frac{\partial f_i}{\partial x_l}(x_J)\frac{\partial^3 (f^{J-1}(x_1))_l}{\partial x_d \partial x_j \partial x_k}
\end{aligned}
$$

Wegens de definities van de multilineaire vormen in sectie 2.3 vinden we dan

$$\begin{aligned}
C^J_{i,j,k,d} &= \sum_{l,m,r} C_{i,l,m,r}(x_J) A^{J-1}_{l,k} A^{J-1}_{m,j} A^{J-1}_{r,d} \\
&\quad + \sum_{l,m} B_{i,l,m}(x_J) B^{J-1}_{l,d,k} A^{J-1}_{m,j} \\
&\quad + \sum_{l,m} B_{i,l,m}(x_J) A^{J-1}_{l,k} B^{J-1}_{m,d,j} \\
&\quad + \sum_{l,m} B_{i,l,m}(x_J) B^{J-1}_{l,j,k} A^{J-1}_{m,d} \\
&\quad + \sum_{l} A_{i,l}(x_J) C^{J-1}_{l,d,j,k} \\
&= \left(C(x_J) A^{J-1} A^{J-1} A^{J-1}\right)_{i,j,k,d} + \left(B(x_J) A^{J-1} B^{J-1}\right)_{i,j,k,d} \\
&\quad + \left(B(x_J) A^{J-1} B^{J-1}\right)_{i,j,k,d} + \left(B(x_J) A^{J-1} B^{J-1}\right)_{i,j,k,d} \\
&\quad + \left(A(x_J) C^{J-1}\right)_{i,j,k,d}
\end{aligned}$$

Beschouwen we vervolgens de vectoren $q_i, i = 1, 2, 3$. We kunnen dan de laatste uitdrukking met $(q_1)_j (q_2)_k (q_3)_d$ vermenigvuldigen en sommeren over (j, k, d). We krijgen dan

$$\begin{aligned}
C^J(q_1, q_2, q_3) &= C(x_J)(A^{J-1} q_1, A^{J-1} q_2, A^{J-1} q_3) \\
&\quad + B(x_J)(B^{J-1}(q_1, q_2), A^{J-1} q_3)^\star \\
&\quad + A(x_J)(C^{J-1}(q_1, q_2, q_3)),
\end{aligned} \tag{3.3}$$

waarbij $\star$ staat voor alle combinatorisch verschillende termen, d.w.z.

$$\begin{aligned}
B(x_J)(B^{J-1}(q_1, q_2), A^{J-1} q_3)^\star &= B(x_J)(B^{J-1}(q_1, q_2), A^{J-1} q_3) \\
&\quad + B(x_J)(B^{J-1}(q_1, q_3), A^{J-1} q_2) \\
&\quad + B(x_J)(B^{J-1}(q_2, q_3), A^{J-1} q_1).
\end{aligned}$$

Ook voor de multilineaire vormen D en E kunnen we een analoge berekening uitvoeren. Als we dan de vectoren $q_i, i = 1, 2, 3, 4, 5$ beschouwen vinden we dan de volgende uitdrukkingen, telkens gebruik makend van de kettingregel.

Voor de multilineaire vorm D hebben we

$$\begin{aligned} D^J(q_1,q_2,q_3,q_4) &= D(x_J)(A^{J-1}q_1, A^{J-1}q_2, A^{J-1}q_3, A^{J-1}q_4) \\ &\quad +C(x_J)(B^{J-1}(q_1,q_2), A^{J-1}q_3, A^{J-1}q_4)^\star \\ &\quad +B(x_J)(B^(J-1)(q_1,q_2), B^{J-1}(q_3,q_4))^\star \\ &\quad +B(x_J)(C^{J-1}(q_1,q_2,q_3), A^{J-1}q_4)^\star \\ &\quad +A(x_J)D^{J-1}(q_1,q_2,q_3,q_4). \end{aligned} \tag{3.4}$$

Voor de multilineaire vorm E hebben we

$$\begin{aligned} E^J(q_1,q_2,q_3,q_4,q_5) &= E(x_J)(A^{J-1}q_1, A^{J-1}q_2, A^{J-1}q_3, A^{J-1}q_4, A^{J-1}q_5) \\ &\quad +D(x_J)(B^{J-1}(q_1,q_2), A^{J-1}q_3, A^{J-1}q_4, A^{J-1}q_5)^\star \\ &\quad +C(x_J)(B^{J-1}(q_1,q_2), B^{J-1}(q_3,q_4), A^{J-1}q_5)^\star \\ &\quad +C(x_J)(C^{J-1}(q_1,q_2,q_3), A^{J-1}q_4, A^{J-1}q_5)^\star \\ &\quad +B(x_J)(C^(J-1)(q_1,q_2,q_3), B^{J-1}(q_4,q_5))^\star \\ &\quad +B(x_J)(D^{J-1}(q_1,q_2,q_3,q_4), A^{J-1}q_5)^\star \\ &\quad +A(x_J)E^{J-1}(q_1,q_2,q_3,q_4,q_5). \end{aligned} \tag{3.6}$$

We zien nu dat als de orde van afleiden stijgt, we in bovenstaande recursies steeds meer termen krijgen. Hieraan gekoppeld zullen we geconfronteerd worden met een niet-lineaire groei van de complexiteit en van de tijd nodig om de berekeningen uit te voeren als het aantal iteraties J stijgt. Om dit te verduidelijken gebruiken we e_1, e_2, e_3, e_4 en e_5 om de complexiteit van de berekeningen van de multilineaire functies weer te geven, m.a.w.

$$\begin{aligned} e_1 &= e(Aq), \\ e_2 &= e(B(q_1,q_2)), \\ e_3 &= e(C(q_1,q_2,q_3)), \\ e_4 &= e(D(q_1,q_2,q_3,q_4)), \\ e_5 &= e(E(q_1,q_2,q_3,q_4,q_5)), \end{aligned}$$

waarbij e het aantal nodige elementaire bewerkingen voorstelt.

Gebruik makend van de net afgeleide recursieformules (3.1), (3.2), (3.3), (3.4) en (3.6) vinden we dan dat

$$\begin{aligned}
e(A^J q) &= Je_1 && (3.7)\\
e(B^J(q_1,q_2)) &= (J-1)(J+1)e_1 + Je_2 \\
&= J^2e_1 + Je_2 + \text{l.o.t}, && (3.8)\\
e(C^J(q_1,q_2,q_3)) &= J^3e_1 + J^2e_2 + Je_3 + \text{l.o.t}, \\
e(D^J(q_1,q_2,q_3,q_4)) &= J^4e_1 + J^3e_2 + J^2e_3 + Je_4 + \text{l.o.t}, \\
e(E^J(q_1,q_2,q_3,q_4,q_5)) &= J^5e_1 + J^4e_2 + J^3e_3 + J^2e_4 + Je_5 + \text{l.o.t}, && (3.9)
\end{aligned}$$

waarbij l.o.t. staat voor lagere orde termen.
Hierbij is (3.7) onmiddellijk te vinden en kunnen we de andere aantalen bewijzen door inductie. Bewijzen we hier nu (3.8).

Voor $J = 1$ vinden we inderdaad

$$e(B^1(q_1,q_2)) = (1-1)(1+1)e_1 + 1e_2 = e_2 = e(B(q_1,q_2)).$$

Nemen we nu aan dat de formule geldt voor $J-1$

$$e(B^{J-1}(q_1,q_2)) = (J-2)(J)e_1 + (J-1)e_2,$$

dan vinden we, wegens (3.2)

$$\begin{aligned}
e(B^J(q_1,q_2) &= e_2 + (J-1)e_1 + (J-1)e_1 + e_1 + (J-2)Je_1 + (J-1)e_2 \\
&= (J^2-1)e_1 + Je_2,
\end{aligned}$$

en dit is juist (3.8).

We zien dat symbolisch afleiden niet altijd even efficiënt is voor hogere orde afgeleiden. We kunnen deze afgeleiden met CL_MatContM echter ook met automatische differentiatie berekenen.

3.2.2 Multilineaire afbeeldingen berekenen met AD

We definiëren hiervoor eerst een functie die een afbeelding f het gewenste aantal keren itereert. Hierbij is het argument `func` de (function-handle van) afbeelding f.

```
function y1= Bvv(mapsf,x0,hc,p1,taylorder,n)
   s = adtayl(0,taylorder); %Base point & Taylor order
   y1= x0 + s.*hc;
   for i=1:n
     y1 = mapsf(0, y1,p1{:});
   end
```

We geven nu de code voor `multilinear1AD` en `multilinear2AD` die respectievelijk $A^J q_1$ en $B^J(q_1, q_2)$ berekenen, gebruik makend van de iteratiefunctie `Bvv`. Als inputargumenten geven we `q1` en `q2`, de n-vectoren q_1 en q_2, waarbij n afhangt van de afbeelding. Daarnaast zijn `x0` en `par` respectievelijk de vectoren van toestandsvariabelen en van parameterwaarden in het bifurcatiepunt. `J` geeft het aantal iteraties. Er zijn analoge codes voor de hogere-orde multilineaire vormen tot orde 5.

```
function ytayl1 = multilinear1AD(func,q1,x0,p1,n)
        taylorder=1;
        y1=Bvv(func,x0,q1,p1,taylorder,n);
        ytayl1=tcs(y1);
```

```
function ytayl2 = multilinear2AD(func,q1,q2,x0,p1,n)
        taylorder=2;
        if q1==q2
        y1=Bvv(func,x0,q1,p1,taylorder,n);
        else
        y11=Bvv(func,x0,q1+q2,p1,taylorder,n);
        y12=Bvv(func,x0,q1-q2,p1,taylorder,n);
        y1=1/4.0*(y11-y12);
        end
        ytayl2=2*tcs(y1);
```

We krijgen zo $A^j q_1$ als de laatste kolom van `ytayl1`, namelijk als `ytayl1(:,end)`; en $B^j(q_1, q_2)$ als het dubbel van de laatste kolom van `ytayl2`, namelijk `2*ytayl2(:,end)`. Dit laatste is het geval omdat we te maken hebben Taylorreeksontwikkelingen en de tweede afgeleide daarin voorkomt met

een coëfficiënt $\frac{1}{2}$. We moeten dus opnieuw met 2 vermenigvuldigen, of met een andere overeenkomstige coëfficiënt voor hogere orde afgeleiden, om de juiste multilineaire vorm te bekomen. We zien dat in de code voor multilineaire vorm B^J gebruik gemaakt wordt van de polarisatie-identiteit (2.9).

Voorbeeld: Het Cod Stock model

Beschouwen we nu het volgende voorbeeld met de specifieke afbeelding

$$M_{CS} : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow \begin{pmatrix} Fe^{-\beta_1 x_2} x_2 + (1+\mu_1) e^{-\beta_2 x_2} x_1 \\ Pe^{-\beta_3 x_2} x_1 + (1+\mu_2) x_2 \end{pmatrix}, \tag{3.10}$$

het Cod Stock model. De achterliggende gronden van dit model kunnen worden teruggevonden in [CASW01, CUSH98, DENN97]. Dit model beschrijft de kabeljauwpopulatie en wordt beschreven door de twee-dimensionale differentievergelijking (3.10) met zeven parameters. Hierin zijn $x_{1,t}$ en $x_{2,t}$ respectievelijk de volwassen en onvolwassen vissen uit de populatie op tijdstip t. F stelt de vruchtbaarheid voor, d.w.z. het aantal geboortes per volwassen vis en P, $0 < P \leq 1$, is de fractie van de onvolwassen populatie die overleeft en één tijdstip later volwassen wordt. μ_2 is de natuurlijk sterftegraad, μ_1 stelt de combinatie van natuurlijke sterfte en volwassenheid voor, dus $\mu_1 \geq P$. De parameters $\beta_i, i = 1, 2, 3$ noemt men cannibalismeparameters. Zo stelt $e^{-\beta_1 x_2}$ de reductie van F voor ten gevolge van cannibalisme uitgeoefend door de volwassen dieren van de populatie, zal de overblijvende onvolwassen populatie $(1-\mu_1)x_1$ gereduceerd worden met een factor $e^{-\beta_2 x_2}$ en zal tenslotte het overlevende deel dat overgaat naar de volwassen fase gereduceerd worden met de factor $e^{-\beta_3 x_2}$ door cannibalisme van de vissen reeds in deze volwassen fase.

De differentievergelijking (3.10) moet eerst worden geïmplementeerd in Matlab. Dit wordt als volgt gedaan

```
function y = CodStockFunc(t,x,F,P,beta1,beta2,beta3,mu1,mu2)
x1 = x(1);
x2 = x(2);
y=[F*exp(-beta1*x2)*x2 + (1-mu1)*exp(-beta2*x2)*x1;
   P*exp(-beta3*x2)*x1 + (1-mu2)*x2 ];
```

Vervolgens kunnen we $A^J q_1$ en $B^J(q_1, q_2)$ evalueren voor specifieke vectoren q_1 en q_2 aan de hand van volgende code. Hierbij zijn `q1` en `q2` gewone kolomvectoren terwijl `par` een Matlab cell-array is en de argumenten F, P, β_1, β_2, β_3, μ_1 en μ_2 bevat, in die volgorde. `@CodstockFunc` is de function-handle voor de CodStock functie.

```
>>par = {399.5861,0.5,1,1,1,0.5,0.444715}
>>x0=[26;3]
>>q1=[1;2]
>>q2=[3;4]
>>J=15
>>ytayl1=multilinear1AD(@CodstockFunc,q1,x0,par,J)
>>ytayl2=multilinear2AD(@CodstockFunc,q1,q2,x0,par,J)
>>A=ytayl1(:,end)
>>B=ytayl2(:,end)
```

We krijgen zo de volgende resultaten

```
ytayl1 =
   2.3892e+001   7.7217e+001
   3.0478e+000  -3.1647e-001

ytayl2 =
             0   1.5525e+002   3.8664e+003
             0  -6.3639e-001   1.4503e+000

A =
   7.7217e+001
  -3.1647e-001

B =
   3.8664e+003
   1.4503e+000
```

3.2.3 Test case

In deze paragraaf voeren we een test uit om de nauwkeurigheid en snelheid van AD en SD te vergelijken voor het berekenen van normaalvormcoëfficiënten . Deze testen worden uitgevoerd met CL_MatContM.

We beschouwen de twee-dimensionale differentievergelijking met drie para-

meters

$$M_K : \begin{pmatrix} x \\ y \end{pmatrix} \longrightarrow \begin{pmatrix} Sx - y - (\epsilon y^2 + x^2) \\ Lx - (y^2 + x^2)/5 \end{pmatrix},$$

waarbij ϵ, L en S de parameters van het model zijn (bron: ongepubliseerde PhD thesis van A.Yu. Kuznetsova, Saratov University).

De gïtereerde afbeelding M_k^3 heeft een vast punt $(x^\star, y^\star)$
$= (0.37588802742303, -0.62783638474655)$ als de parameterwaarden gegeven worden door $(\epsilon, L, S) = (1, 1.3353, -0.799600)$. Na continueren van de vaste punten van deze derde iteratie, met L vrij en ϵ, S vast gehouden, krijgen we een superkritisch flip bifurcatiepunt bij de parameterwaarde $L = 1.335343$.
De afbeelding M_K^3 heeft een cascade van flippunten die we achtereenvolgens kunnen berekenen door over te gaan op de nieuwe takken van verdubbelde periode in de PD punten. We berekenen op die manier de PD punten van orde 3, 6, 12 24, 48, 96 en 192 samen met hun respectievelijke normaalvormcoëfficiënten .

Om het gebruik van CL_MatContM te illustreren geven hier ook de code om de opeenvolgende PD punten en de bijhorende normaalvormcoëfficiënten te berekenen.
Eerst en vooral moeten we een afbeeldingsfile aanmaken voor de afbeelding M_K onder beschouwing. Zo een afbeeldingsfile bevat onder andere de secties `init, fun_eval, jacobian, jacobianp, hessians, hessiansp, der3, der4, der5`. Met andere woorden bevat deze file onder andere de opeenvolgende symbolische afgeleiden van oplopende orde en werd aangemaakt met behulp van MatCont door de optie 'Symbolic' aan te duiden in het 'System' window. Voor de afbeelding M_K ziet de afbeeldingsfile er als volgt uit

```
function out = TestCase1
out{1} = @init;
out{2} = @fun_eval;
out{3} = @jacobian;
out{4} = @jacobianp;
out{5} = @hessians;
out{6} = @hessiansp;
out{7} = @der3;
out{8} = @der4;
out{9} = @der5;

% ------------------------------------------------------------------
```

```
function dydt = fun_eval(t,kmrgd,S,e,L)
dydt=[S*kmrgd(1)-kmrgd(2)-(e*kmrgd(2)^2+kmrgd(1)^2);
L*kmrgd(1)-(kmrgd(2)^2+kmrgd(1)^2)/5;];

%-------------------------------------------------------------------
function [tspan,y0,options] = init
handles = feval(TestCase1);
y0=[0,0];
options = odeset('Jacobian',handles(3),'JacobianP',handles(4),
'Hessians',handles(5),'HessiansP',handles(6));
tspan = [0  10];

%-------------------------------------------------------------------
function jac = jacobian(t,kmrgd,S,e,L)
jac=[[S - 2*kmrgd(1),- 2*e*kmrgd(2) - 1];
[L - (2*kmrgd(1))/5,-(2*kmrgd(2))/5]];
%-------------------------------------------------------------------
function jacp = jacobianp(t,kmrgd,S,e,L)
jacp=[[kmrgd(1),-kmrgd(2)^2,0];[0,0,kmrgd(1)]];
%-------------------------------------------------------------------
function hess = hessians(t,kmrgd,S,e,L)
hess1=[[-2,0];[-2/5,0]];
hess2=[[0,(-2)*e];[0,-2/5]];
hess(:,:,1) =hess1;
hess(:,:,2) =hess2;
%-------------------------------------------------------------------
function hessp = hessiansp(t,kmrgd,S,e,L)
hessp1=[[1,0];[0,0]];
hessp2=[[0,(-2)*kmrgd(2)];[0,0]];
hessp3=[[0,0];[1,0]];
hessp(:,:,1) =hessp1;
hessp(:,:,2) =hessp2;
hessp(:,:,3) =hessp3;
%-------------------------------------------------------------------
function tens3  = der3(t,kmrgd,S,e,L)
tens31=[[0,0];[0,0]];
tens32=[[0,0];[0,0]];
tens33=[[0,0];[0,0]];
tens34=[[0,0];[0,0]];
tens3(:,:,1,1) =tens31;
tens3(:,:,1,2) =tens32;
tens3(:,:,2,1) =tens33;
tens3(:,:,2,2) =tens34;
%-------------------------------------------------------------------
function tens4  = der4(t,kmrgd,S,e,L)
```

```
tens41=[[0,0];[0,0]];
tens42=[[0,0];[0,0]];
tens43=[[0,0];[0,0]];
tens44=[[0,0];[0,0]];
tens45=[[0,0];[0,0]];
tens46=[[0,0];[0,0]];
tens47=[[0,0];[0,0]];
tens48=[[0,0];[0,0]];
tens4(:,:,1,1,1) =tens41;
tens4(:,:,1,1,2) =tens42;
tens4(:,:,1,2,1) =tens43;
tens4(:,:,1,2,2) =tens44;
tens4(:,:,2,1,1) =tens45;
tens4(:,:,2,1,2) =tens46;
tens4(:,:,2,2,1) =tens47;
tens4(:,:,2,2,2) =tens48;
%-------------------------------------------------------------------------
function tens5  = der5(t,kmrgd,S,e,L)
tens51=[[0,0];[0,0]];
tens52=[[0,0];[0,0]];
tens53=[[0,0];[0,0]];
tens54=[[0,0];[0,0]];
tens55=[[0,0];[0,0]];
tens56=[[0,0];[0,0]];
tens57=[[0,0];[0,0]];
tens58=[[0,0];[0,0]];
tens59=[[0,0];[0,0]];
tens510=[[0,0];[0,0]];
tens511=[[0,0];[0,0]];
tens512=[[0,0];[0,0]];
tens513=[[0,0];[0,0]];
tens514=[[0,0];[0,0]];
tens515=[[0,0];[0,0]];
tens516=[[0,0];[0,0]];
tens5(:,:,1,1,1,1) =tens51;
tens5(:,:,1,1,1,2) =tens52;
tens5(:,:,1,1,2,1) =tens53;
tens5(:,:,1,1,2,2) =tens54;
tens5(:,:,1,2,1,1) =tens55;
tens5(:,:,1,2,1,2) =tens56;
tens5(:,:,1,2,2,1) =tens57;
tens5(:,:,1,2,2,2) =tens58;
tens5(:,:,2,1,1,1) =tens59;
tens5(:,:,2,1,1,2) =tens510;
tens5(:,:,2,1,2,1) =tens511;
```

```
tens5(:,:,2,1,2,2) =tens512;
tens5(:,:,2,2,1,1) =tens513;
tens5(:,:,2,2,1,2) =tens514;
tens5(:,:,2,2,2,1) =tens515;
tens5(:,:,2,2,2,2) =tens516;
```

Om het eerste PD punt te vinden van de derde iteratie van M_K gaan we als volgt te werk

```
disp('>> global cds fpmds; ');
global opt cds fpmds
disp('>> p=[-0.799600;1;1.3353];ap=3;; ');
p=[-0.799600;1;1.3353];ap=3;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>> opt = contset(opt,''Backward'',0); ');
opt = contset(opt,'Backward',0);
opt = contset(opt,'MaxNumPoints',200);
disp('>>opt = contset(opt,''Singularities'',1); ');
opt = contset(opt,'Singularities',1);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',0);

disp('opt = contset(opt,''Singularities'',1);');
disp('>>> curve of fixed points <<< ')
disp('>> [x0,v0]=init_FPm_FPm(@TestCase1,
          [0.37588802742303;-0.62783638474655] , p, ap,3); ');
[x0,v0]=init_FPm_FPm(@TestCase1,[0.37588802742303;
                -0.62783638474655], p, ap,3);
disp('>> [x1,v1,s1,h1,f1]=cont(@fixedpointmap,x0,[],opt); ');
[x1,v1,s1,h1,f1]=cont(@fixedpointmap,x0,[],opt);
disp('>> cpl(x1,v1,s1,[3 1]); ');
cpl(x1,v1,s1,[3 1]);
```

We vinden dan

```
>> global cds fpmds;
>> p=[-0.799600;1;1.3353];ap=3;;
>> opt = contset;
>> opt = contset(opt,'Multipliers',1);
>> opt = contset(opt,'Backward',0);
>>opt = contset(opt,'Singularities',1);
>>opt = contset(opt,'AutDerivative',1);
```

```
opt = contset(opt,'Singularities',1);
>>> curve of fixed points <<<
>> [x0,v0]=init_FPm_FPm(@TestCase1,[0.37588802742303;
                      -0.62783638474655], p, ap,3);
>> [x1,v1,s1,h1,f1]=cont(@fixedpointmap,x0,[],opt);
first point found
tangent vector to first point found
label = PD   , x = ( 0.375974 -0.627941 1.335343 )
Elapsed time is 0.002991 seconds.
 normal form coefficient of PD = 5.657152e+003
label = LP   , x = ( 0.375976 -0.627944 1.335343 )
normal form coefficient of LP =-3.260989e+000
label = NS   , x = ( 0.398882 -0.830845 1.522617 )
 normal form coefficient of NS = -1.128603e+002
label = PD   , x = ( 0.402012 -0.809927 1.492462 )
Elapsed time is 0.002800 seconds.
 normal form coefficient of PD = -4.717937e+001

elapsed time  = 0.4 secs
npoints curve = 200
>> cpl(x1,v1,s1,[3  1]);
```

Het eerst vermelde PD punt is datgene waarbij $x = 0.375974$, $y = -0.627941$ en $L = 1.335343$. De gevonden normaalvormcoëfficiënt is $b = 5.657152e{+}003$.

Voor het tweede PD punt, dit op de tak van dubbele periode 6 vinden we door

```
disp('>> global x1 v1 s1  opt cds fpmds ');
global x1 v1 s1 opt cds fpmds
TestCase11;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>>x11=x1(1:2,s1(2).index);p1=p;
p1(fpmds.ActiveParams)=x1(3,s1(2).index); ');
x11=x1(1:2,s1(2).index);
p1=p;
p1(fpmds.ActiveParams)=x1(3,s1(2).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

```

```
disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x2,v2]=init_PDm_FP2m(@TestCase1,x11,p1,s1(2),0.01,3);');
[x2,v2]=init_PDm_FP2m(@TestCase1,x11,p1,s1(2),0.01,3);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',1);
disp('>> [x3,v3,s3,h3,f3]=cont(@fixedPointmap,x2,[],opt,2);');
[x3,v3,s3,h3,f3]=cont(@fixedPointmap,x2,[],opt);
cpl(x3,v3,s3,[3 1])
```

De resultaten zijn dan

```
>> opt = contset;
>> opt = contset(opt,'Multipliers',1);
>>x11=x1(1:2,s1(2).index);p1=p;
     p1(fpmds.ActiveParams)=x1(3,s1(2).index);
>> opt = contset(opt,'MaxNumPoints',100);
>>opt = contset(opt,'AutDerivative',1);
>> opt=contset(opt,'Singularities',1);
>> [x2,v2]=init_PDm_FP2m(@TestCase1,x11,p1,s1(2),0.01,3);
>> opt=contset(opt,'Backward',1);
>> [x3,v3,s3,h3,f3]=cont(@fixedPointmap,x2,[],opt,2);
first point found
tangent vector to first point found
label = BP   , x = ( 0.375974 -0.627941 1.335343 )
label = LP   , x = ( 0.373720 -0.668294 1.335226 )
normal form coefficient of LP =-4.912155e-001
label = PD   , x = ( 0.349755 -0.948447 1.436871 )
Elapsed time is 0.005262 seconds.
 normal form coefficient of PD = 8.753605e+001
label = NS   , x = ( 0.325091 -1.022694 1.543400 )
Neutral saddle
result =

Neutral saddle

label = PD   , x = ( 0.300421 -1.063404 1.638879 )
Elapsed time is 0.005115 seconds.
 normal form coefficient of PD = 9.751120e+002

elapsed time  = 0.4 secs
npoints curve = 100
```

Het eerst gevonden PD punt is nu datgene waarbij $x = 0.349755$, $y =$

-0.948447 en $L = 1.436871$. De gevonden normaalvormcoëfficiënt is $b = 8.753605e + 001$.

J	bifurcatiepunt
3	(0.375974,-0.627941,1.335343)
6	(0.349755,-0.948447,1.436871)
12	(0.336832,-0.881886,1.463676)
24	(0.330787,-0.857379,1.469354)
48	(0.329173,-0.852556,1.470561)
96	(0.3391975,-0.925268,1.470819)
192	(0.339223,-0.925734,1.470874)

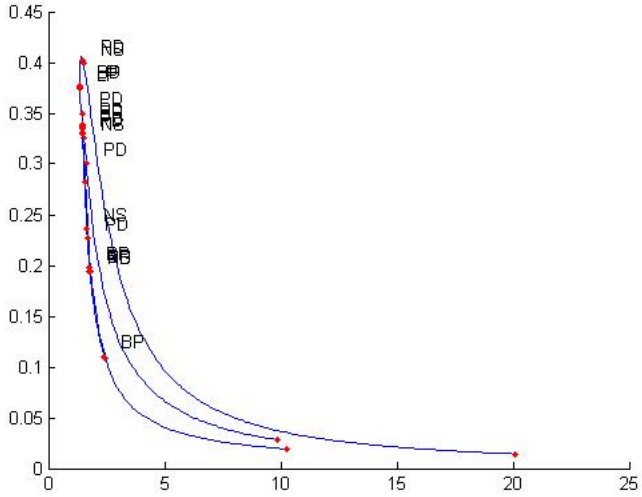
Tabel 3.2: PD bifucatiepunten van de iteraties J van M_K.

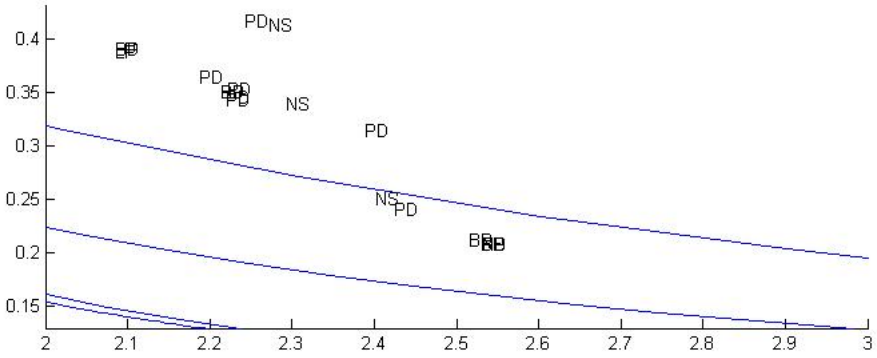
J	SD	t	AD	t
3	5.657152e+3	0.002117	5.657152e+3	0.002991
6	8.753605e+1	0.005247	8.753605e+1	0.005262
12	5.807278e+2	0.019042	5.807278e+2	0.019314
24	2.080773e+4	0.082113	2.080773e+4	0.182804
48	6.022927e+5	0.487828	6.022927e+5	0.363931
96	4.881097e+6	3.317996	4.881097e+6	0.719735
192	1.969787e+8	25.489389	1.969787e+8	1.442571

Tabel 3.3: Normaalvormcoëfficiënten voor de PD bifucatiepunten van de iteraties J van M_K.

Analoog kunnen we voor de periodes 12, 24, 48, 96 en 192 PD punten vinden (Zie code in bijlage). Door achtereenvolgens de optie AutDerivative aan of uit te schakelen, kunnen we de normaalvormcoëfficiënten respectievelijk met AD en SD berekenen. De punten (x, y, L) worden weergegeven in tabel 3.2 en de bijhorende normaalvormcoëfficiënten vindt u in tabel 3.3, samen met de verlopen tijd om de berekeningen te maken. We zien dat in het geval van AD de verlopen tijd lineair groeit met het aantal iteraties J. Bij SD zal daarentegen de verlopen tijd veel sneller groeien. Voor de iteraties van lage orde 3, 6, 12 en 24 is SD echter nog steeds sneller dan AD, maar vanaf een orde van 48 wordt AD veel sneller dan SD.

We krijgen dan volgende plots





3.2.4 Symbolisch afleiden in Maple

We hernemen hier de uitdrukkingen (2.7) en (2.8) aangepast voor een geïtereerde afbeelding f^J

$$B(v,v) = \frac{d^2}{d\tau^2} f^J(x_0 + \tau q_1, \alpha_0)\Big|_{\tau=0}, \tag{3.11}$$

$$C(v,v,v) = \frac{d^3}{d\tau^3} f^J(x_0 + \tau q_1, \alpha_0)\Big|_{\tau=0}, \tag{3.12}$$

met q_1 een eigenvector van de Jacobiaan $(f^j)_x$, om de normaalvormcoëfficiënten voor bifurcaties in het punt x_0 te berekenen.

Om nog een extra vergelijking te maken tussen de efficiëntie van symbolisch afleiden en automatisch afleiden en het nut van Cl_MatcontM om o.a. normaalvormcoëfficiënt te berekenen, te illustreren kunnen we proberen een geïtereerde functie $f^J(x_0 + tq_1)$ in Maple af te leiden naar variabele t. Maple gebruikt net zoals de meeste wiskundesoftware symbolische differentiatie.
Voor een eenvoudige functie ondervinden we niet zo snel problemen. Nemen we echter een iets complexere functie kan Maple de afgeleide van de geïterteerde functie niet meer berekenen.

Als eenvoudig voorbeeld om ons op te baseren voor de complexere afbeelding nemen we de afbeelding

$$d(t) := cos(4 + sin(7 + t)).$$

Gebruik makend van volgende code in Maple kunnen we de afgeleide van d^{64} nog net berekenen, hoewel dit al heel lang duurt.

```
> restart; with(Student[VectorCalculus]);
> d := t -> cos(4+sin(7+t));
> diff(d(t), t);
> d2 := t -> d(d(t));
> diff(d2(t), t);
> d4 := t -> d2(d2(t));
> d4(t);
> diff(d4(t),t);
> d8 := t -> d4(d4(t));
> d8(t);
> diff(d8(t), t);
> d16 := t -> d8(d8(t));
> d16(t);
> diff(d16(t), t);
```

```
> d32 := t -> d16(d16(t));
> d32(t);
> diff(d32(t), t);
> d64 := t -> d32(d32(t));
> d64(t);
> diff(d64(t), t);
> d128 := t -> d64(d64(t));
> d128(t);
> diff(d128(t), t);
```

Bij iteratie d^{128} wordt de melding *Too many levels of recursion for display"* weergegeven. Het is dus onmogelijk om bijvoorbeeld de symbolische afgeleide van de duizendste iteratie van deze eenvoudige afbeelding te berekenen met Maple, iets wat met Cl_MatContM geen probleem is.

Nemen we nu echter de complexere afbeelding uit TestCase 1 dan zal Maple het al opgeven bij iteratie 8.

```
> restart; with(Student[VectorCalculus]);
> F := 1;
  P := 2;
  mu1 := 3;
  mu2 := 4;
  beta1 := 1;
  beta2 := 2;
  beta3 := 3;
> MCS :=(x1, x2)-> <(F*exp(-beta1*x2)*x2+(1-mu1)*exp(-beta2*x2)*x1,
                     P*exp(-beta3*x2)*x1+(1-mu2)*x2)>;
> dMCS := MCS(MCS(x, y)[1], MCS(x, y)[2]);
> diff(dMCS[1], x);
> diff(dMCS[1], y);
> x0 := Vector([26, 3]); q1 := Vector([1, 2]);
> h := t -> MCS(x0[1]+t*q1[1], x0[2]+t*q1[2]);
> dh := diff(h(t), t);
> t := 0;
> dh;
> h2 := t -> h(h(t)[1], h(t)[2]);
```

```
> dh2 := diff(h2(t), t);
> t := 0;
> dh2;
> h4 := t -> h2(h2(t)[1], h2(t)[2]);
> dh4 := diff(h4(t), t);
> t := 0;
> dh4;
> h8 := t -> h4(h4(t)[1], h4(t)[2]);
> dh8 := diff(h8(t), t);
> t := 0;
> dh8;
```

Bij de laatste afgeleide van de MCS^8 geeft Maple de melding *"Length of output exceeds limit of 1000000"*. Voor een iets moeilijker afbeelding is het dus vanaf de achtste iteratie al onmogelijk om de afgeleide te berekenen.

3.3 Het aanpassen van de bestaande Matlab-code

Om automatische differentiatie mogelijk te maken voor Taylorpolynomen werd gebruik gemaakt van Matlab. Dit gebeurde in de vorm van de nieuwe `adtayl` klasse ter uitbreiding van het Matlabpakket Cl_MatcontM. In deze klasse werden bijna alle standaard Matlabfuncties overschreven voor `adtayl` objecten, nl.

```
sqrt,exp,log,log10,sin,cos,tan,cotan,sec,csec,asin,acos,
   atan,acotan,asec,acsec,sinh,cosh,tanh,cotanh,sech,
      csech,asinh,acosh,atanh,acotanh,asech,acsech
```

Voor deze thesis werd vertrokken van de code van Prof J. Pryce van Cranfield University en het resultaat van het doctoraatsproefschrift van R. Khoshsiar Ghaziani. Niet alle standaardfuncties werken even goed of werken soms zelfs niet. Daarnaast werkten de meeste functies in de `adtayl` klasse alleen voor scalairen en nog niet voor vector/matrix argumenten. De belangrijkste taak voor deze thesis was dus het aanpassen en optimaliseren van de code.

In de map AD van de `adtayl` klasse vindt men de hierboven vermelde overschreven Matlabfuncties. Een eerste opdracht bestond erin te kijken welke functies al dan niet werkten voor scalairen. Hier werd duidelijk dat er problemen waren bij de machtsverheffing en alle functies die op deze machtsverheffing steunden zoals `acsc, asec` en `asech`. Ook bij verder tests om na te gaan welke functies werkten voor vectoren en matrices kwamen deze problemen voor.
In Matlab vinden we twee soorten machtsverheffig: de elementsgewijze `power` functie .^ en de `mpower` functie ^ voor matrices. Aangezien we werken met Taylorpolynomen is de betekenis van de `mpower` functie ^ niet echt meer duidelijk en zelfs onlogisch. Er werd dus geopteerd om deze functie te overschrijven door de elementsgewijze `power` functie .^ voor `adtayl` objecten. Deze functie ziet er nu als volgt uit:

```
function x = mpower(x,y)
% X^Y for ADTAYL objects.
% X is an ADTAYL or numeric array. If it is numeric, Y must be
% ADTAYL (for this function to be called at all), and then X
% is converted to a constant ADTAYL of the same size.
% Y is an numeric scalar, or an ADTAYL array. In the latter case,
% X and Y must have the same Taylor order, and either they have
% the same size, or one is scalar.
% When Y is an integer numeric scalar, the power is found by
% repeated multiplication, if Y>0; and by reciprocation and r
% epeated multiplication, if Y<0; and is the constant 1 ADTAYL
% array, if Y=0.
% If it is a non-integer numeric scalar, it is converted to an
% ADTAYL. In this case, and when X, Y are both ADTAYLs
% (of same size), the formula X^Y = exp(Y*log(X)) is used,
%elementwise.

% We don't check at present that X, Y are either ADTAYL or
%numeric.
if ~isa(x,'adtayl'), x = adtayl(x); end
if ~isa(y,'adtayl')
  if ~(isnumeric(y) && numel(y)==1)
    error('Y must be ADTAYL or numeric scalar');
  end
  if fix(y)==y % Y is an integer scalar:
    x=powerin(x,y);
    return
  else % Y is a non-integer scalar, convert to constant ADTAYL:
    y = adtayl(y);
```

```
30    end
31  end
32  % Both are now ADTAYLs:
33  if order(x)~=order(y)
34    error('Arguments X and Y have different Taylor orders')
35  end
36  [m,n] = size(x);
37  [my,ny] = size(y);
38  if ~( (m==my && n==ny) || m*n==1 || my*ny==1)
39    error('X and Y must be of same size or one must be a scalar');
40  end
41  ind = find(x.tc(:,:,1)==0);
42  if any(ind)
43    error('Some entry in X has zero constant term: cannot raise to
44    noninteger power');
45  end
46  x=exp(y.*log(x));
```

Zoals reeds vermeld in de commentaar van deze functie wordt er een onderscheid gemaakt tussen de drie verschillende gevallen:

- Y is een geheel getal (dus `fix(y)==y`),
- Y is een niet-geheel getal,
- Zowel X als Y is een `adtayl` object.

In het eerste geval wordt gebruik gemaakt van de `powerin` functie, dit is de functie X^Y voor `adtayl` objecten voor het geval waarbij X een algemeen `adtayl` object is en Y een geheel getal In het tweede geval wordt Y omgezet naar een `adtayl` object om dan over te gaan naar geval 3 waarbij gebruik gemaakt wordt van de formule

$$X^Y = \exp(Y \log(X)).$$

Naast de machtsverheffing ontstonden heel wat problemen met Matlab's ingebouwde `filter` functie. Eerst en vooral was het niet helemaal duidelijk wat deze functie nu juist deed. De helpfiles van deze functie waren dan ook ontoereikend voor onze doeleinden. Na overleg met professor Pryce werd dan toch duidelijk hoe deze functie te werk ging (zie ook in de inleiding van dit

hoofdstuk). Een opmerkelijk feit was ook dat eens de code was aangepast en alles leek te werken in Matlab 7.5, overschakelen op Matlab 7.7 opnieuw foutmeldingen gaf. Zo werd duidelijk dat de `filter` functie in Matlab 7.7 strenger is dan deze in Matlab 7.5 waardoor de code opnieuw aanpassingen moest ondergaan.

De code om de vierkantswortel uit een `adtayl` object te nemen, werd geoptimaliseerd aan de hand van de Newtoniteratie om een forlus te vermijden:

```
function u = sqrt(x)

% sqrt for adtayl objects.
[m,n,p1] = size(x.tc);
xtc = reshape(x.tc,[m*n,p1]);
x0 = xtc(:,1); %array of constant coefficients
ind = find(x0==0); % positions where const coeff is zero
indall0 = find(sum(abs(xtc),2)==0); %always a subset of ind
if numel(indall0) < numel(ind) % some TS has const coeff 0 but is
                               % not all 0
    error('Cannot take sqrt of nonzero TS with zero constant
                term');
end
x0(ind) = 1; % Dummy 1's in places of identically zero TS's
x.tc(:,:,1) = reshape(x0,[m,n]);
u=adtayl(sqrt(x.tc(:,:,1)));
ncoeff=1;
while ncoeff<p1
    u=(u+x./u)/2;
    ncoeff=2*ncoeff;
end
u.tc(ind) = 0;
```

De redenering hierachter gaat als volgt.

Zij v_0 een initiële schatting, dan is

$$v_1 := \frac{v_0 + \frac{u}{v_0}}{2} \tag{3.13}$$

een betere schatting voor de vierkantswortel uit u.
Dit samen met de volgende stelling zorgt ervoor dat we een efficiënte code krijgen.

Stelling 3.1. *Als u een polynoom is en de initiële v heeft een correcte constante term dan zal 3.13 het aantal correcte coëfficiënten verdubbelen bij elke*

iteratie.

Om de code te laten werekn voor matrices en vectoren van `adtayl` objecten moesten slechts enkele kleine aanpassingen gebeuren die te maken hadden met de `reshape` functie in Matlab. Deze werk als volgt

`B = reshape(A,m,n,p,...) of B = reshape(A,[m n p ...])`

geeft een n-dimensionale array met dezelfde elementen als A maar herschikt tot de dimensies m-bij-n-bij-p-bij-...Het product van de dimensies, $m*n*p*...$, moet dezelfde waarde hebben als $prod(size(A))$. Deze `reshape` wordt andere gebruikt voor de functie `acos`:

```
function x = acos(x)
%acos for adtayl objects.

[m,n,p] = size(x.tc);
for i=1:m
    for j=1:n

    x1.tc= reshape(x.tc(i,j,:),[1, 1,p]);
    x1=class(struct('tc',x1.tc),'adtayl');
    acos0 = acos(x1.tc(1,1));
    xx=sqrt((1-sqr(x1)));
    xy=reshape(xx.tc,1,p);
    dx1 = -x1.tc(:,2:end).*(1:p-1);
    y = filter(1,xy,dx1)./(1:p-1);
    xx1.tc(i,j,:)=[acos0 y];
end
 end

x.tc = reshape(xx1.tc, m,n,p);
```

Hierbij wordt van het `adtayl` object een vector gemaakt met 1 kolom waarvan elk element een p-vector is. Zodoende kan voor een m-bij-n-bij-p matrix de code voor een p-vector worden toegepast. Daarna wordt de kolomvector opnieuw herschikt tot een matrix van orde m-bij-n-bij-p.

Verder klopten de tekens hier een daar niet. Toevoegen van een -teken bij o.a. de functies `acotan, acsc` en `acsch` loste deze problemen op.

Voor de uiteindelijke code die behoort te werken voor zowel `adtayl` scalairen, vectoren en matrices kan u terecht op [MatCont]. Eventuele bugs kunnen worden gerapporteerd naar ...

Hoofdstuk 4

Toekomstig werk

De overload-algoritmen voor automatische differentiatie in Matlab zijn nu wel geschreven en getest, maar er blijven twee niet-triviale taken. Als een alternatief voor symbolisch afleiden werd automatische differentiatie geïmplementeerd voor de berekening van de normaalvormcoëfficiënten , zodat deze nauwkeurig kunnen worden berekend wanneer symbolische afgeleiden niet beschikbaar zijn en zodat deze berekeningen sneller verlopen. Daarom is de eerste toekomstige taak deze algoritmen zodanig in te bouwen in het algoritmisch pakket van MatCont dat ze waar nodig gebruikt kunnen worden.
De tweede taak bestaat erin de gebruikersinterface van MatCont zo aan te passen dat de gebruiker ze kan oproepen als hij het wenst (het is ook niet zeker dat ze altijd beter zijn dan symbolische afgeleiden). Er zullen dus vele tests nodig zijn om uit te maken wat de ideale situatie is. Daarnaast zou ook een afzonderlijk GUI (Graphic User Interface) gelijkaardig aan deze voor Matcont voor Cl_MatcontM nuttig zijn.

Een ander voorstel voor de toekomst is om automatische differentiatie te gebruiken om de symbolische code te genereren voor de afgeleiden van $f^J(x_0 + tq_1)$. Door bijvoorbeeld in tekstvorm de symbolische code te laten genereren wordt de dure symbolische toolbox van Matlab overbodig.

Bibliografie

[GRI80] Griewank A.: Starlike domains of convergence for Newton's method at singularities. Numer. Math. 35 95-111 (1980)

[GRIE00] Griewank A.: *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation.* SIAM, Philadelphia (2000)

[GRIE08] Griewank A. en Walther A.: *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation, Second Edition.* SIAM, Philadelphia (2008)

[GOV07] Govaerts W., Khoshsiar Ghaziani R., Kuznetsov Yu. A. en Meijer H.G.E: *Numerical methods for two-parameter local bifurcation analysis of maps.* SIAM J Sci. Comput. 29(6) 2644-2667 (2007)

[KUZ04] Kuznetsov Yu. A.: *Elemenents of Applied Bifurcation Theory, Third Edition.* Springer-Verlag, New York (2004)

[KUZ05] Kuznetsov Yu. A. en Meijer H.G.E: *Numerical normal forms for codim 2 bifurcations of maps with at most two critical eigenvalues.* SIAM J Sci. Comput. 26(6) 1932-1954 (2005)

[KHOS08] Khoshsiar Ghaziani R.: *Bifurcations of maps: numerical algorithms and applications.* Doctoraatsproefschrift Ugent, Faculteit Wetenschappen, Vakgroep Toegepaste Wiskunde en Informatica (2008)

[MEIJ06] Meijer, H.G.E.: *Codimension 2 Bifurcations of Iterated Maps.* http://igitur-archive.library.uu.nl/dissertations/2006-1204-200716/index.htm, Doctoraatsproefschrift Universiteit Utrecht, Wiskunde en Informatica Proefschriften (2006)

[MatCont] `http`: //`sourceforge.net/projects/matcont`

[BUCK05] Bücker H.M.: *Looking for narrow interfaces in automatic differentiation using graph drawing* FGCS., 21 1418-1425 (2005)

[CASW01] Caswell H.: *Matrix Population Models* Sunderland, Massachusetts: Sinaur Ass. Inc. Publishers (2001)

[CUSH98] Cushing J.M., Constantino R.F., Dennis B., Deshamias R.A. en A.M. Henson: *Nonlinear population dynamics: models, experiments and data* J. Theor. Biol., 194 1-9 (1998)

[DENN97] Dennis B., Deshamais R.A., Cushing J.M. en Constantino R.F.: *Transition in popular dynamics: equilibria to periodic cycles to aperiodic cycles* J. Anim. Ecol., 6b 704-729 (1997)

[JUED92] Juedes D.W.: *A Taxonomy of Automatic Differentiation Tools* TR91-20 (1992)

[BUCK06] Bcker H.M., Corliss G., Hovland P., Naumann U., Norris B.: *Automatic Differentiation: Applications, Theory, and Implementations* Series: Lecture Notes in Computational Science and Engineering , Vol. 50 (2006)

Bijlage A

Bijlage

De code voor TestCase1 bestaat uit 7 deeltjes:

1. TestCase11.m

```
disp('>> global cds fpmds; ');
global opt cds fpmds
disp('>> p=[-0.799600;1;1.3353];ap=3;; ');
p=[-0.799600;1;1.3353];ap=3;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>> opt = contset(opt,''Backward'',0); ');
opt = contset(opt,'Backward',0);
opt = contset(opt,'MaxNumPoints',200);
disp('>>opt = contset(opt,''Singularities'',1); ');
opt = contset(opt,'Singularities',1);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

disp('opt = contset(opt,''Singularities'',1);');
disp('>>> curve of fixed points <<< ')
disp('>> [x0,v0]=init_FPm_FPm(@TestCase1,
[0.37588802742303;-0.62783638474655], p, ap,3); ');
[x0,v0]=init_FPm_FPm(@TestCase1,
[0.37588802742303;-0.62783638474655],p, ap,3);
disp('>> [x1,v1,s1,h1,f1]=cont(@fixedpointmap,x0,[],opt); ');
```

```
[x1,v1,s1,h1,f1]=cont(@fixedpointmap,x0,[],opt);
disp('>> cpl(x1,v1,s1,[3 1]); ');
cpl(x1,v1,s1,[3 1]);
```

2. TestCase12.m

```
disp('>> global x1 v1 s1 opt cds fpmds ');
global x1 v1 s1 opt cds fpmds
TestCase11;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>>x11=x1(1:2,s1(2).index);p1=p;
p1(fpmds.ActiveParams)=x1(3,s1(2).index); ');
x11=x1(1:2,s1(2).index);
p1=p;
p1(fpmds.ActiveParams)=x1(3,s1(2).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x2,v2]=init_PDm_FP2m(@TestCase1,x11,p1,s1(2)
,0.01,3);');
[x2,v2]=init_PDm_FP2m(@TestCase1,x11,p1,s1(2),0.01,3);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',1);
disp('>> [x3,v3,s3,h3,f3]=cont(@fixedPointmap,x2,[],opt);');
[x3,v3,s3,h3,f3]=cont(@fixedPointmap,x2,[],opt);
cpl(x3,v3,s3,[3 1])
```

3. TestCase13.m

```
disp('>> global x3 v3 s3 opt cds fpmds ');
global x3 v3 s3 opt cds fpmds
TestCase12;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>>x12=x3(1:2,s3(4).index);p1=p;
p1(fpmds.ActiveParams)=x3(3,s3(4).index); ');
```

```
x12=x3(1:2,s3(4).index);
p1=p;
p1(fpmds.ActiveParams)=x3(3,s3(4).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x4,v4]=init_PDm_FP2m(@TestCase1,x12,p1,s3(4)
,0.01,6);');
[x4,v4]=init_PDm_FP2m(@TestCase1,x12,p1,s3(4),0.01,6);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',1);
disp('>> [x5,v5,s5,h5,f5]=cont(@fixedPointmap,x4,[],opt);');
[x5,v5,s5,h5,f5]=cont(@fixedPointmap,x4,[],opt);
cpl(x5,v5,s5,[3  1])
```

4. TestCase14.m

```
disp('>> global x5 v5 s5  opt cds fpmds ');
global x5 v5 s5 opt cds fpmds
TestCase13;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>>x13=x5(1:2,s5(2).index);p1=p;
p1(fpmds.ActiveParams)=x5(3,s5(2).index); ');
x13=x5(1:2,s5(2).index);
p1=p;
p1(fpmds.ActiveParams)=x5(3,s5(2).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x6,v6]=init_PDm_FP2m(@TestCase1,x13,p1,s5(2)
,0.01,12);');
[x6,v6]=init_PDm_FP2m(@TestCase1,x13,p1,s5(2),0.01,12);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',1);
```

```
disp('>> [x7,v7,s7,h7,f7]=cont(@fixedPointmap,x6,[],opt);');
[x7,v7,s7,h7,f7]=cont(@fixedPointmap,x6,[],opt);
cpl(x7,v7,s7,[3  1])
```

5. TestCase15.m

```
disp('>> global x7 v7 s7  opt cds fpmds ');
global x7 v7 s7 opt cds fpmds
TestCase14;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>>x14=x7(1:2,s7(2).index);p1=p;
p1(fpmds.ActiveParams)=x7(3,s7(2).index); ');
x14=x7(1:2,s7(2).index);
p1=p;
p1(fpmds.ActiveParams)=x7(3,s7(2).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x8,v8]=init_PDm_FP2m(@TestCase1,x14,p1,s7(2)
,0.01,24);');
[x8,v8]=init_PDm_FP2m(@TestCase1,x14,p1,s7(2),0.001,24);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',1);
opt=contset(opt,'FunTolerance',1e-4);
opt=contset(opt,'VarTolerance',1e-4);
opt=contset(opt,'TestTolerance',1e-3);
disp('>> [x9,v9,s9,h9,f9]=cont(@fixedPointmap,x8,[],opt);');
[x9,v9,s9,h9,f9]=cont(@fixedPointmap,x8,[],opt);
cpl(x9,v9,s9,[3  1])
```

6. TestCase16.m

```
disp('>> global x9 v9 s9  opt cds fpmds ');
global x9 v9 s9 h9 opt cds fpmds
TestCase15;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
```

```
opt = contset(opt,'Multipliers',1);
disp('>>x15=x9(1:2,s9(2).index);p1=p;
p1(fpmds.ActiveParams)=x9(3,s9(2).index); ');
x15=x9(1:2,s9(2).index);
p1=p;
p1(fpmds.ActiveParams)=x9(3,s9(2).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x10,v10]=init_PDm_FP2m(@TestCase1,x15,p1,s9(2)
,0.001,48);');
[x10,v10]=init_PDm_FP2m(@TestCase1,x15,p1,s9(2),0.001,48);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',0);
opt=contset(opt,'FunTolerance',1e-4);
opt=contset(opt,'VarTolerance',1e-4);
opt=contset(opt,'TestTolerance',1e-3);
disp('>> [x11,v11,s11,h11,f11]=cont(@fixedPointmap,x10,[]
,opt);');
[x11,v11,s11,h11,f11]=cont(@fixedPointmap,x10,[],opt);
cpl(x11,v11,s11,[3 1])
```

7. TestCase17.m

```
disp('>> global x9 v9 s9  opt cds fpmds ');
global x11 v11 s11 opt cds fpmds
TestCase16;
disp('>> opt = contset; ');
opt = contset;
disp('>> opt = contset(opt,''Multipliers'',1); ');
opt = contset(opt,'Multipliers',1);
disp('>>x15=x9(1:2,s9(2).index);p1=p;
p1(fpmds.ActiveParams)=x9(3,s9(2).index); ');
x16=x11(1:2,s11(3).index);
p1=p;
p1(fpmds.ActiveParams)=x11(3,s11(3).index);
disp('>> opt = contset(opt,''MaxNumPoints'',100);');
opt=contset(opt,'MaxNumPoints',100);
disp('>>opt = contset(opt,''AutDerivative'',1); ');
opt = contset(opt,'AutDerivative',1);

```

```
disp('>> opt=contset(opt,''Singularities'',1);');
opt=contset(opt,'Singularities',1);
disp('>> [x10,v10]=init_PDm_FP2m(@TestCase1,x15,p1,s9(2)
,0.001,48);');
[x12,v12]=init_PDm_FP2m(@TestCase1,x16,p1,s11(3),0.001,96);
disp('>> opt=contset(opt,''Backward'',1);');
opt=contset(opt,'Backward',1);
opt=contset(opt,'FunTolerance',1e-5);
opt=contset(opt,'VarTolerance',1e-5);
opt=contset(opt,'TestTolerance',1e-4);
disp('>> [x11,v11,s11,h11,f11]=cont(@fixedPointmap,x10,[]
,opt);');
[x13,v13,s13,h13,f13]=cont(@fixedPointmap,x12,[],opt);
cpl(x13,v13,s13,[3 1])
```