Hoofdstuk I

Numeriek oplossen van differentiaalvergelijkingen

door G. Sleijpen

SLEUTELBEGRIPPEN: *benaderingsfout, foutvoortplanting, gestructureerde fout, foutschatting, betrouwbaarheid foutschatting, efficiëntie, stabiliteit*

Inleiding

Problemen die je in de praktijk tegenkomt kunnen nogal eens gemodelleerd worden als een differentiaalvergelijking of een stelsel differentiaalvergelijkingen [1]. Gewoonlijk zijn die differentiaalvergelijkingen niet analytisch oplosbaar. Middels analytische technieken [1, Hoofdstuk 2] kan je deelaspecten analyseren die enig inzicht kunnen geven in het "kwalitatieve" gedrag van de oplossing: via bijvoorbeeld linearisatie kan je iets te weten komen over de oplossing in de buurt van evenwichten, soms zijn er asymptotische uitspraken mogelijk, etc.. Helaas zijn die analytische technieken niet altijd toepasbaar of geven ze een onvolledig beeld. Bovendien is men in de praktijk geïnteresseerd in "kwantitatieve" resultaten: al in een ontwerpfase van bijvoorbeeld een gebouw zal men zeker willen weten of de constructie stormen en aardbevingen zal doorstaan; een raket wordt pas gelanceerd als de baan in grote nauwkeurigheid berekend is, etc.. In dit soort gevallen is men aangewezen op numerieke berekeningen. De belangrijke vraag is dan of de numerieke resultaten betrouwbaar zijn en hoe je kan achterhalen hoe betrouwbaar ze zijn. Verder vraag je je af of er één numerieke methode is waarmee iedere differentiaalvergelijking nauwkeurig opgelost kan worden en als dat niet zo is waar het van afhangt welke methode voor welk type differentiaalvergelijking het meest geschikt is. Zijn er standaard pakketten? Hoe roep je ze aan? Werken ze altijd? Als dat niet het geval is, hoe pas je ze aan voor je eigen probleem?

1.1 Differentiaalvergelijkingen

Gewone differentiaalvergelijkingen (GDVs), het type differentiaalvergelijkingen dat we in deze cursus bekijken, zijn van de vorm

$$u'(t) = f(t, u(t)), \quad t \in [t_0, t_0 + T]. \tag{1.1.a}$$

Hierin is f een gegeven functie en u een onbekende functie. u moet bepaald worden. De functies u en f zijn reëelwaardig of vectorwaardig: u is een C^1-afbeelding van het

Voorbeeld 1B. Het lineaire eendimensionale beginwaarde probleem (1.2) is ook van het type (1.1) als λ en g complexwaardige functies zijn. De oplossing u is dan ook complexwaardig. Door alle grootheden te splitsen in reële en imaginaire delen kan de GDV geschreven worden als een lineaire tweedimensionale reële GDV. Voor de analyse is het overigens handiger om met de complexe grootheden te werken.

In het algemeen is het zo dat iedere 'complexe' d-dimensionale GDV geschreven kan worden als een reële $2d$-dimensionale GDV.

Voorbeeld 1C. [`Flame`] Het volgende eendimensionale beginwaarde probleem is een model voor het verloop van de temperatuur in een chemische reactie. κ en μ zijn constanten, $u(t)$ is de temperatuur op tijdstip t.

$$\begin{cases} u' = \mu\,(2-u)\,e^{\kappa(1-\frac{1}{u})} & \text{op} \quad [0,1] \\ u(0) = 1. \end{cases} \tag{1.3}$$

Deze GDV is autonome. In een *autonome GDV* hangt de functie f die het rechterlid definieert niet af van de tijd: de GDV hangt via het rechterlid alleen via $u(t)$ van de tijd af. In (1.3) is $f(t,u) = \mu\,(2-u)\,\exp(\kappa(1-1/u)$. Voor een autonome GDV is het begintijdstip niet essentieel: als $u = v$ en $u = w$ de oplossing zijn van dezelfde autonome GDV met dezelfde beginwaarde u_0, maar met verschillende begintijd, dan is v gelijk aan w op een vertraging na: als $v(t_0) = u_0$ en $w(t_0 - s) = u_0$ dan is $v(t) = w(t-s)$.

Voorbeeld 1D. [`Immuun`] Antigenen ("ziekmakers") worden in ons lichaam gebonden door antilichamen en vervolgens onschadelijk gemaakt. De GDV in dit voorbeeld modelleert de verhouding in concentraties tussen deze twee. $x(t)$ is de concentratie antigenen in het lichaam op tijdstip t, $y(t)$ is de concentratie antilichamen. Het verloop van de concentraties kan, onder zekere aanname, beschreven worden door

$$\begin{cases} x' = 2(1 + \kappa x - \kappa y)x \\ y' = (-1 + 3\kappa x - \kappa y - \vartheta\kappa xy)y \end{cases} . \tag{1.4}$$

Hierin zijn κ en ϑ nog twee nader te specificeren parameters.

De GDV (1.4) is tweedimensionaal en met

$$u = \begin{bmatrix} x \\ y \end{bmatrix}, \quad f(t,u) = \begin{bmatrix} 2(1 + \kappa x - \kappa y)x \\ (-1 + 3\kappa x - \kappa y - \vartheta\kappa xy)y \end{bmatrix}$$

zien we dat deze GDV van het type is als in (1.1.a).

Voorbeeld 1E. [`vdPol`] De *van der Pol vergelijking* speelt een rol in de medische biologie bij het modelleren van de hartslag. De vergelijking bevat niet alleen eerste orde afgeleiden (u'), maar ook een hogere orde (u''). Met een eenvoudige truc kan de hogere orde afgeleide omgeschreven worden naar een afgeleide van de eerste orde. De GDV wordt hierdoor wel hoger dimensionaal.

Voor zekere reële constanten μ, β en ε en u_0, w_0 wordt de van der Pol vergelijking gegeven door

$$\begin{cases} \varepsilon u'' = \mu(u' + \beta) - u - (u' + \beta)^3 & \text{op} \quad [0,T] \\ u(0) = u_0, \quad u'(0) = w_0. \end{cases} \tag{1.5}$$

passen bij een Arenstorf baan met periode $T = 17.065\,216\,560\,157\,962\,558\,891\,720\,624\,9$. Het berekenen van een Arenstorf baan vergt hoge precisie en de startwaarden moeten in een groot aantal cijfers bekend zijn.

1.2 Kennismaking met numerieke oplosmethoden

SLEUTELBEGRIPPEN: *Betrouwbaarheid numeriek antwoord, efficiëntie*

In deze paragraaf proberen we te achterhalen of we blindelings mogen vertrouwen op de oplossingen die de computer ons levert.

Vraag 2A. Kunnen computers GDVs eigenlijk wel oplossen? Differentiëren is een limietoperatie. Computers kunnen alleen met getallen uit een eindige collectie werken. Krijg je wel de oplossing te zien?

Experiment. Los (1.2) (`Linear`) met λ als in (1.2.d) en $\varepsilon = 0.02$ op in MATLAB met de routine '`odeEF`'.[2] Plot ook de exacte oplossing.[3] Bekijk de figuren die MATLAB produceert. Is het numeriek antwoord in alle tijdstippen berekend? (Zoom in als je meer details wilt zien). Valt het numerieke antwoord samen met het exacte antwoord in relevante tijdstippen? Welke zijn de relevante tijdstippen?

Verklaring. De afgeleide $u'(t)$ is benaderd door $\frac{1}{h}[u(t+h) - u(t)]$ of een ander differentie-quotiënt. Verder is er alleen gerekend in een beperkte set van tijdstippen t_n (is $t_n = nh$ voor $n = 0, 1, 2, \ldots$?). Door te benaderen introduceer je fouten.

Vraag 2B. Wat gebeurt er als je in meer tijdstippen rekent, d.w.z. als je de afstand h tussen de rekenpunten verandert?

Experiment. Verklein h eens met, zeg, een factor 3.[4] Wat gebeurt er met de fout? Verklein h eens met een factor 10 (bereken nu niet de exacte oplossing).[5] Wat gebeurt er met de rekentijd? Wat gebeurt er als je h vergroot?

Verklaring. Met een grotere h zijn er minder rekenpunten en dus zal het rekenwerk ook wel minder zijn. Echter voor grotere h is de benadering 'grover', d.w.z. de benaderingsfouten zijn groter. h wordt de **stapgrootte** genoemd.

Vraag 2C. Zijn de antwoorden altijd betrouwbaar? Of hangt de betrouwbaarheid samen met de stapgrootte? Wordt de fout kleiner als je h verkleint? Kan je aan de resultaten zien of ze betrouwbaar zijn?

Experiment. We experimenteren met twee routines. De eerste `odeEF` is door onszelf gecodeerd en berust op een eenvoudige methode die we in de volgende paragrafen uitgebreid zullen bestuderen. De tweede routine `ode23` is zeer geavanceerd en behoort tot de ODE Toolbox van MATLAB.

[2] 'Run' hiertoe de m-file `Vraag2.m`. Kijk wel eerst of het goede beginwaarde probleem gekozen is: `ODE='Linear2'` en de goede oplosmethode: `method='odeEF'`. Check ook eerst of in de function file `Linear2.m` de GDV en de beginvoorwaarden goed gedefinieerd zijn. Een uitgebreidere handleiding bij de MATLAB files staat in de file `READ.ME`.

[3] D.w.z, neem, in de file `Vraag2.m`, `PlotExactSolution=1` .

[4] h heet `dt` in `Vraag2.m` en in de andere '`Vraag`-files'.

[5] Neem `PlotExactSolution=0`.

$u_n + hf(t_n, u_n)$: de grootheden u_n en t_n zijn bekend, en $f(t_n, u_n)$ kan uitgerekend worden. Er kan dus gemakkelijk een benadering $u_{n+1} \equiv u_n + hf(t_n, u_n)$ van $u(t_n + h)$ berekend worden. We kunnen vervolgens het spelletje herhalen: we kunnen nu op dezelfde manier in het tijdstip $t_n + 2\,h$ de oplossing schatten, etc.. Beginnen we met deze aanpak in het tijdstip $t_0 = 0$ dan weten we dat $u(0) = u_0$ (zie (1.1.b)) en komen we tot het volgend rekenproces:

$$u_{n+1} = u_n + hf(t_n, u_n), \qquad t_{n+1} = t_n + h, \quad n = 0, 1, 2, \ldots, \tag{3.2}$$

waarin u_n een benadering is voor $u(t_n)$. Dit rekenschema heet de **Euler forward** (EF) methode.

Vraag 3A. De oplossing van een beginwaarde probleem kan grafisch gerepresenteerd worden. Hoe kan je de EF methode grafisch interpreteren?

Experiment. Plot voor (1.2) (`Linear`) met $\lambda = -1$ en g als in (1.2.c),[9] de grafiek van de exacte oplossing en plot, voor niet al te kleine h, de numerieke oplossing. Verbind de punten (t_n, u_n) met lijnstukjes.[10] Plot ook de exacte oplossing van de GDV (1.2.a) door (t_1, u_1), d.w.z. plot de grafiek van die u waarvoor $u' = \lambda\, u + g$ en $u(t_1) = u_1$. Doe dat ook voor de exacte oplossingen van de GDV door achtereenvolgens (t_2, u_2),..., (t_6, u_6).[11] Bekijk de grafische resultaten in de buurt van de eerste paar t_n (zoom in).

Verklaring. EF volgt telkens over korte stukjes (stukjes met, langs de t-as, lengte h) de raaklijn en stapt daarbij telkens over op een andere 'naburige' oplossing.

Vraag 3B. Convergeert EF? Wat betekent convergentie hier?

Experiment. Onderzoek door h te verkleinen voor de testvoorbeelden (1.2) (`Linear`) met $\lambda = -1$ en (1.4) (`Immuun`) met $\kappa = 1.3$ en $\vartheta = 0.06$ of EF convergeert en als er convergentie is, in welke zin dat gebeurt.[12]

Verklaring. Er geldt

$$u(t_{n+1}) = u(t_n) + h\, f(t_n, u(t_n)) + \tau_n \quad \text{waarbij} \quad \tau_n \equiv \tfrac{1}{2}\,(h)^2\, u''(\xi_n) \tag{3.3}$$

voor geschikte ξ_n tussen t_n en t_{n+1}.
EF kan je zien als een methode waarin in ieder tijdstip t_n het **defect** τ_n weggemoffeld wordt (vergelijk (3.3) en (3.2)). De defecten τ_n gaan naar 0 als $h \to 0$ en ze gaan sneller naar 0 dan de hf-term (waarom?). Het is duidelijk dat de defecten de nauwkeurigheid van het numeriek resultaat beïnvloeden, het is niet duidelijk hoe ze dat doen. Het is evenmin duidelijk of er niet nog andere factoren een rol spelen.

Bij convergentiebeschouwingen moet je je realiseren dat voor iedere t de n in de vergelijking $t_n = t$ afhangt van h.

Opgave. *Bewijs* (3.3). *Hint: gebruik een Taylor ontwikkeling.*

We onderzoeken eerst hoe de defecten de numerieke nauwkeurigheid beïnvloeden. Als h klein is, dan is

$$\tau_n \approx \tfrac{1}{2}\, h^2\, u''(t_n). \tag{3.4}$$

[9] Dit beginwaarde probleem is gedefinieerd in `Linear.m`.

[10] Gebeurt vanzelf met `Vraag3A.m`. De `Vraag*.m`-files zijn copiën van `main.m` met waarden voor parameters passend bij de corresponderende Vraag.

[11] Neem `PlotExactNearbySol=1`.

[12] Het tweede beginwaarde probleem is gedefinieerd in `Immuun.m`.

Voor EF toegepast op (1.2) geldt

$$e_{n+1} = (1 + h\,\lambda_n)\,e_n + \tau_n \quad \text{met} \quad \lambda_n \equiv \lambda(t_n). \tag{3.7}$$

Over een tijdstraject ter lengte h vergroot of verkleint EF de 'oude' fout met een factor $1 + h\,\lambda_n$. (Hoe groot was deze factor in de beroerde gevallen?) Verder komt er in dat traject nog een 'nieuwe' fout bij, een 'lokale' onnauwkeurigheid, nl. het defect.

Opgave. *Bekijk EF toegepast op* (1.2).

a. *Bewijs* (3.7).

b. *Als* $|h\,\lambda_n| \ll 1$ *(denk aan* $|h\,\lambda_n| \leq 0.2$*) dan geldt* $1 + h\,\lambda_n \approx e^{h\,\lambda_n}$. *Bewijs dit en vergelijk dit resultaat met het resultaat in onderdeel c. van de vorige opgave.*

Conclusie.
a. *Defecten (al of niet vergroot; zie* b. *en* c.*) cumuleren.*
b. *De GDV zelf kan fouten vergroten.*
c. *De numerieke methode kan fouten vergroten.*

Als $|h\,\lambda_n| \not\ll 1$ dan springt EF anders om met 'oude' fouten dan de GDV. Dit is gewoonlijk fataal als ook nog $|1 + h\,\lambda_n| > 1$. We zullen deze situatie uitgebreid bestuderen in §1.4. Voorlopig concentreren we ons op de situatie waarin h zo klein is dat EF de fouten op min of meer dezelfde manier voortplant als de GDV zelf. In dat geval is de conclusie in 3C correct.

Vraag 3E. Kan je het inzicht uit 3C gebruiken om de fout te schatten? Kan je dit inzicht gebruiken om de stapgrootte zo aan te passen dat je een fout krijgt die kleiner is dan bv. 10^{-3}?

Experiment. Los weer (1.2) (`Linear`) met $\lambda = -1$ op met EF voor $h = 0.2$ en $h = 0.1$ en schat de fout. Bepaal ook de exacte oplossing. Hoe goed is je schatting? Met welke waarde voor h verwacht je een fout te krijgen die in alle (relevante) tijdstippen kleiner is dan 10^{-3}. Kijk of je verwachting uitkomt.

Deze manier om de fout te schatten is *a posteriori*: de fout wordt achteraf (nadat er gerekend is) uit numerieke resultaten geschat. Met behulp van de formule voor τ_n is het ook mogelijk (maar overigens niet erg practisch) om de fout te schatten voordat er gerekend wordt (zie 3K). De schatting is dan *a priori*.

Vraag 3F. Geldt die evenredigheid van de fout met h ook voor grotere h (met h toch nog zo klein dat EF fouten niet beroerder voortplant dan de GDV zelf)? Geldt die evenredigheid in alle (relevante) tijdstippen?

Experiment. Los weer (1.2) (`Linear`) met $\lambda = -1$ op met EF maar nu met $h = 1$ en $h = 0.5$.

Verklaring. Als h niet klein is (of preciezer, als u'' veel varieert op $[t, t + h]$) dan is (3.4) niet correct: hogere orde Taylor reeks zijn dan niet verwaarloosbaar. Soms is de fout overigens "per ongeluk" ≈ 0.

Als in een numerieke oplosmethode van GDVs de fouten min of meer evenredig zijn met h^p voor alle h die klein genoeg zijn dan is de methode van **orde** p: EF is een methode van orde 1.

Zijn de inzichten die we voor EF opgedaan hebben ook van toepassing voor andere oplosmethoden? We bekijken ondermeer de Trapeziumregel. Deze methode ontstaat door in iedere t_n het defect τ_n^{trap} in de relatie

$$\begin{cases} u(t_{n+1}) = u(t_n) + \frac{1}{2}h\,[f(t_n, u(t_n)) + f(t_{n+1}, u(t_{n+1}))] + \tau_n^{\text{trap}} \\ \text{met} \quad \tau_n^{\text{trap}} = -\frac{1}{12}h^3\,u'''(\xi_n) \end{cases} \tag{3.9}$$

weg te poetsen.

Vraag 3I. Hoe verandert de fout in de Trapeziumregel als je h verandert? Zijn er methoden die nog nauwkeuriger zijn?

Experiment a. Pas de Trapeziumregel toe op (1.4) (`Immuun`) met $\kappa = 1.3$ en $\vartheta = 0.06$ en kijk of je verwachting uitkomt.[14]

Opgave. *Bewijs* (3.9) *(het is ook al voldoende als je kan bewijzen dat* $\tau_n^{\text{trap}} = -\frac{1}{12}h^3\,u'''(t_n) + \mathcal{O}(Dt^4)$*).*
Hint: bedenk dat $f(t, u(t)) = u'(t)$ *en ontwikkel de voorkomende functies rond* t_n*.*

Experiment b. Welke orde heeft RK4?[15][16] De orde is lastig af te lezen uit de plaatjes. Hoe komt dat?

Conclusie. *De Trapeziumregel is van orde 2,* RK4 *is van orde 4.*

Opgave 3J. *Bewijs dat* (3.9) *ontstaat door de Trapeziumregel voor numeriek integreren toe te passen op de integraal in de relatie* $u(t+h) = u(t) + \int_t^{t+h} u'(s)\,\mathrm{d}s$.

Opgave 3K. *Hieronder een schets van het bewijs dat de EF "convergeert". Details worden als opgave aan de lezer over gelaten.*

Neem $d = 1$*. Bewijs de volgende uitspraken.*

a. $f(t_n, u(t_n)) - f(t_n, u_n) = \frac{\partial f}{\partial u}(t_n, \widetilde{u}_n)\,e_n$ *voor zekere* $\widetilde{u}_n$ *tussen* $u(t_n)$ *en* u_n*.*

b. $e_0 = 0, \quad e_{n+1} = e_n + h\,\frac{\partial f}{\partial u}(t_n, \widetilde{u}_n)\,e_n + \tau_n = \left(1 + h\,\frac{\partial f}{\partial u}(t_n, \widetilde{u}_n)\right) e_n + \tau_n.$

c. *Als* $\left|\frac{\partial f}{\partial u}(t_n, \widetilde{u}_n)\right| \le M$ *voor alle* n *dan geldt*

$$|e_{n+1}| \le (1 + h\,M)|e_n| + |\tau_n| \le \textstyle\sum_{j=0}^{n} (1 + h\,M)^{n-j}|\tau_j|$$

d. *Omdat* $1 + h\,M \le e^{h\,M}$ *is* $(1 + h\,M)^i \le e^{ih\,M}$ *en geldt*

$$|e_n| \le e^{n\,h\,M} \textstyle\sum_{j<n} |\tau_j| \le e^{TM} \sum_{j<n} |\tau_j| \quad \text{voor alle} \quad t_n,\ t_n - t_0 = n\,h \le T.$$

e. $\sum_{j<n} |\tau_j| \le n \max_{j<n} |\tau_j| \le n\,h^2\,\frac{1}{2} \max_{\xi} |u''(\xi)| \le \left(T\,\frac{1}{2} \max |u''(\xi)|\right) h.$

f. *EF convergeert als de functies* $|u''|$ *en* $|\frac{\partial f}{\partial u}|$ *begrensd zijn.*

g. *Stel nu dat* $\frac{\partial f}{\partial u}$ *ook nog negatief is. Laat zien dat dan* $|1 + h\frac{\partial f}{\partial u}(t_n, \widetilde{u}_n)| \le 1$ *voor iedere* h, $0 < h \le 2/M$*. Gebruik dit om je schattingen in c. en d. aan te scherpen.*

We onderzoeken of we bovenstaande schattingen praktisch kunnen gebruiken.

h. *Bepaal een* h *zodat de fout in EF toegepast op* (1.2) (`Linear`) *met* $\lambda = -1$ *(en* g *als in* (1.2.c) *zo dat* $u(t) = \cos(t)$*) over het hele integratietraject* $[0, 20]$ *ten hoogste* 10^{-2} *is. Gebruik hierbij eerst de oorspronkelijke schattingen en daarna de aangescherpte.*

[14] De Trapeziumregel roep je aan met `odeIT`.

[15] `odeRK` maakt gebruik van RK4. Het heeft hier geen zin om de exacte oplossing te laten plotten omdat de "exacte" oplossing berekend wordt met `odeRK`: neem `PlotExactSolution=0`.

[16] RK4 is 'n Runge–Kutta methode. Maar er zijn er vele meer. De RK4 die wij hier bekijken wordt wel de 'klassieke' Runge–Kutta methode genoemd en is degene die voorgesteld is door Runge en Kutta.

Experiment c. Met $\lambda = 0.1$ en $h = 0.04$ is $|1 + \lambda h| = 1.004$. Dit is precies de waarde in ons vorige experiment waarvoor we slechte resultaten kregen. Hoe pakt dat hier uit?

Verklaring. Hoeveel stappen heb je nu nodig?

> **Conclusie. a.** *Als* $|h\,\lambda| \ll 1$ *(denk aan* $|h\,\lambda| \leq 0.2$*) dan plant EF de fouten voort op ongeveer dezelfde manier als de GDV dat doet.*
> **b.** *Als* $|h\,\lambda| \not\ll 1$ *dan kan je alleen een nauwkeurige numerieke oplossing krijgen als (vrijwel) exact voldaan is aan de* stabiliteitseis $|1 + h\,\lambda| \leq 1$.

Vraag 4B. Bovenstaande conclusie geldt voor de eendimensionale lineaire GDV in (1.2) voor $\lambda(t) = \lambda_0$ is constant. Wat hebben we eraan voor gecompliceerdere GDVs? Blijft dan nog iets van de conclusie overeind? Kunnen we gemakkelijk zien of we met instabiliteiten te maken hebben?

Experiment a. Pas EF weer toe op (1.2) maar neem nu een variabele λ: neem λ als in (1.2.d) met $\varepsilon = 0.02$ (`Linear2`).[17] Voor welke h heb je geen stabiliteitsproblemen? Bekijk voor $h = 0.04$ ook eens de grafiek van $h\lambda(t)$.[18] Kan je de instabiliteit verklaren? Kan je, aan de hand van de grafiek van $h\lambda$ schatten voor welke h je van de instabiliteit af bent?

Verklaring. Op trajecten waar $|1 + h\,\lambda(t)| > 1$ blaast de fout op.

Experiment b. We kijken eens wat er van onze inzichten overblijft voor niet-lineare GDVs. Pas EF toe op (1.3) (`Flame`) met $\mu = \frac{1}{4}$ en $\kappa = 20$. Probeer de "kritische" h te vinden. Plot ook eens de grafiek van $h\,\lambda(t)$, nu met $\lambda(t) \equiv \frac{\partial f}{\partial u}(t, u(t))$.[19]

Verklaring. De fouten gedragen zich min of meer als de oplossing van een lineaire GDV (en wel de GDV (3.8)).

Experiment c. Los (1.6) (`vdPol`) op met EF en stapgrootte $h = 0.04$. Plot een faseplaatje. Voor welke waarde van h raak je de 'wiggels' kwijt? Bekijk voor deze h ook $h\,\lambda_i(t_n)$ met $\lambda_1(t_n)$ en $\lambda_2(t_n)$ de eigenwaarden van de Jacobi matrix $\frac{\partial f}{\partial u}(t_n, u_n)$.[20]

Verklaring. In de meerdimensionale situatie hangt de stabiliteit samen met $h\,\lambda_i(t)$ waarbij $\lambda_i(t)$ de i-de eigenwaarde is van de Jacobi matrix $h\,\frac{\partial f}{\partial u}(t, u(t))$.

> **Conclusie.** *Ook voor niet-lineaire (meerdimensionale) GDVs manifesteert instabiliteit zich in de vorm van 'wiggels' in de numerieke oplossing. In zo'n geval moet de stapgrootte h kleiner genomen worden. Wiggels wijzen op instabiliteit. In geval van instabiliteit krijgen we gewoonlijk niet een onnauwkeurige oplossing die er toch "redelijk" uitziet.*

We kunnen nu ook beter omschrijven wanneer, in de conclusie na Vraag 3H, 'h voldoende klein' is.

> **Conclusie.** *De EF fout is min of meer evenredig met h als h voldoende klein is, en dat is het geval als* (i) *én* (ii) *geldt. Hierbij is:*
> (i) *voor alle i en vrijwel alle t geldt dat* $|h\lambda_i(t)| \ll 1$ *of* $|1 + h\lambda_i(t)| \leq 1$
> (ii) *hogere orde Taylor reeks termen in het defect verwaarloosbaar zijn.*

[17] Deze GDV staat in `Linear2.m`.

[18] Neem in `Vraag4.m`, `ShowEigenvaluesJacobian = 1;`. Na de volgende twee experimenten zul je begrijpen waarom we de naam 'ShowEigenvaluesJacobian' gekozen hebben voor het plotten van de grafiek van $h\lambda(t)$.

[19] Neem `ShowEigenvaluesJacobian = 1;`.

[20] Neem `ShowEigenvaluesJacobian = 1;`.

Experiment. Vergelijk de grootte van de stapgrootte die EF vereist met die voor RK4 voor de GDV (1.2) met $\lambda = -500$ en weer $T = 20$ (zie experiment c in 4A). Hoeveel groter kan de stapgrootte met RK4 zijn? Zie je deze factor ook terug bij ingewikkeldere GDVs? Bekijk ook eens (1.3) (zie experiment b in 4B).

Vraag 4E. We hebben hierboven gezien dat een nauwkeurigere methode voor stijve GDVs ook wel eens een heel kleine stapgrootte kan vergen. Je kan je afvragen of er überhaupt wel methoden bestaan waarmee je ook met een grotere stapgrootte nauwkeurige resultaten kan behalen?

We proberen de Euler backward methode (EB). EB is gebaseerd op de 'terugwaartse' differentie (vergelijk dit met (3.1))

$$u'(t) \approx \frac{u(t) - u(t-h)}{h}, \tag{4.1}$$

Dit leidt tot de relatie

$$u(t_n) = u(t_{n-1}) + h\, f(t_n, u(t_n)) + \tau_n^{\mathrm{EB}} \quad \text{waarbij} \quad \tau_n^{\mathrm{EB}} = -\tfrac{1}{2}\, h^2\, u''(\xi_n). \tag{4.2}$$

Wegpoetsen van de defecten τ_n^{EB} geeft EB.

Experiment a. Gebruik de routine '`odeEB`' om de GDV uit experiment a in 4B (`Linear2`) op te lossen. Met welke h (zo groot mogelijk) krijg je nog redelijke oplossingen? Vergelijk dit met de situatie voor EF. Hoe pakt het uit voor de GDV (`vdPol`) in experiment c in 4B? Neem ook eens in `Linear2` $\varepsilon = 10^{-6}$ en $h = 0.1$.

Verklaring. Probeer EB eens grafisch te interpreteren.
Verder geldt voor de fouten

$$e_n = \tfrac{1}{1-h\,\lambda_n}(e_{n-1} + \tau_n^{\mathrm{EB}}). \tag{4.3}$$

Opgave. *Bewijs* (4.3).

Experiment b. Wat verwacht je van EB voor stijve niet-lineaire GDVs? Pas EB ook eens toe op (1.7) (`Emigration`) met $\alpha = 2$, $\gamma = 1$, $\varepsilon = -0.01$. Wat is de "optimale" stapgrootte? Hoe ligt dat voor EF? Moet je de "optimale" stapgrootte voor EB en EF nog herzien als je $\varepsilon = -0.001$ neemt?

Verklaring. De fouten gedragen zich min of meer als de oplossing van een lineaire GDV (zie (3.8)).

Conclusie. *EB is aantrekkelijk met name voor stijve GDVs.*

Opgave 4F. *Bewijs, voor $d = 1$, dat EB "convergeert". Ga daarbij te werk als in opgave 3K.*

Vraag 4G. Waarom werken we niet altijd met EB?

Experiment a. Probeer eens een stap EB uit te voeren "met de hand" voor (1.3).

Experiment b. Los (1.4) (`Immuun`), nu met $\kappa = 1$ and $\vartheta = 0.25$, eens op met EB met $h = 0.1$ en ook met EF met $h = 0.1$. Hoe kan je de verschillen verklaren? Plot ook $h\,\lambda_i(t_n)$ als functie van t_n. Neem nu eens voor h in EB de waarde 0.11.

Verklaring. Kijk eens naar de foutvoortplantingsfactor $\frac{1}{1-h\,\lambda_n}$ van EB voor de λ's die we hier tegen kwamen.

B 12 cijfers nauwkeurigheid produceert: je zit dan 'appels met peren' te vergelijken![22] Of een rekenproces efficiënter is kan overigens ook afhangen van de machine waarop je rekent en van de software (compiler) die je gebruikt: sommige rekenprocessen kunnen deels parallel worden uitgevoerd en daar kan het proces van profiteren als je machine meerdere processoren heeft en je compiler dat weet uit te buiten. Om de machine afhankelijkheid zullen we ons in deze cursus verder niet bekommeren (we doen alle berekening op een standard PC of SUN systeem en we gebruiken MATLAB).

Is EF efficiënter dan EB? Dit is een zinloze vraag. Je zult eerst moeten vaststellen welk (beginwaarde) probleem je wilt oplossen (wat is X?) en hoe nauwkeurig je dat wilt doen (en eigenlijk ook op welke machine). Dus: voor 'n specifieke GDV en 'n specifiek nauwkeurigheid zou EF efficiënter kunnen zijn dan EB en in 'n ander geval wellicht niet. In het algemeen is het overigens onmogelijk om *exact* vast te stellen met welke nauwkeurigheid het rekenproces daadwerkelijk de uitkomst berekend heeft. Gelukkig kunnen we wel de nauwkeurigheid schatten (zie Vraag 3C) en kunnen we vaststellen of die schatting betrouwbaar is (zie Vraag 3G). Onze efficiëntie uitspraken baseren we daarom op dit soort schattingen. Het schatten van de fout en het inzien of de schatting betrouwbaar is kost overigens ook rekentijd. Je kan je afvragen of je die tijd in je efficiëntie analyse mee moet tellen. In onze experimenten en opdrachten kiezen we ervoor om dat niet te doen, en wel om de volgende reden. In de praktijk voert men vaak een efficiëntie analyse uit voor een *model probleem*, dat wil zeggen voor een lager dimensionale vereenvoudigde versie. Of, als je een familie van problemen hebt die afhangen van 'n parameter, doe je de analyse voor één specifieke goedgekozen waarde voor de parameter. Je hoopt dan dat de conclusies ook correct zijn voor het echte probleem of voor het probleem met de andere waardes voor de parameter. De rekentijd die in het betrouwbaar schatten van de fout gaat zitten (en jouw tijd om alles te analyseren? :-)) is dan verwaarloosbaar ten opzichte rekentijd die nodig is om het echte probleem, of de familie van problemen, op te lossen. De echte 'sterke' codes schatten overigens ook de fout en zijn daardoor trager dan hun 'zwakkere broeders'. Als je professionele codes wilt gebruiken moet je besluiten of je voor betrouwbaarheid kiest (de sterke maar langzamere code met betrouwbare foutschatting) of voor efficiëntie (de snellere code waarbij je zelf maar moet zien te achterhalen hoe nauwkeurig de antwoorden zijn).

EF kan je met diverse stapgroottes toepassen. De efficiëntste variant van EF voor een specifiek probleem met een specifieke nauwkeurigheid krijg je uiteraard als je met de grootste stapgrootte werkt die de oplossing produceert met de gewenste nauwkeurigheid.[23] In iedere efficiëntie analyse voor het oplossen van een specifiek probleem, moet je dus eerst, nadat je de gewenste nauwkeurigheid hebt vastgeteld, achterhalen wat de grootste stapgrootte is die je je, gezien de gewenste nauwkeurigheid, kunt permiteren. Eenzelfde opmerking geldt uiteraard ook voor EB en voor andere numerieke oplosmethoden.

Je kunt natuurlijk toch de foute vraag stellen: is EF efficiënter dan EB? Alleen kan je dan geen eenduidig antwoord verwachten.

Vraag 5A. Is, voor een nauwkeurigheid van 10^{-3}, EF efficiënter dan EB voor het oplossen van (1.2) met λ als in (1.2.d)?

[22] Als je je niet om nauwkeurigheid bekommert zou je, ongeacht het probleem, altijd kunnen zeggen dat de uitkomst π is en voor die uitkomst hoef je zelfs niet te rekenen :-)

[23] de rekentijd is ruwweg evenredig met het aantal stappen dat je nodig hebt om van t_0 in $t_0 + T$ te komen, en is dus ongeveer omgekeerd evenredig met de stapgrootte h

in $t = 10$ met een nauwkeurigheid van 10^{-2}?[27] Kies de toleratie *tol* in `ode23` zodat `ode23` dezelfde nauwkeurigheid haalt in $t = 10$. Welke methode is het efficiëntst?

Verklaring. Kijk eens naar de stapgroottes die `ode23` genomen heeft.[28]

We hebben de mogelijkheid van variabele stapgrootte aangekaart omdat EF in (1.2) met λ als in (1.2.d) maar in een beperkt deel van het tijdsinterval 'behoefte' had aan een kleine stapgrootte. Die behoefte was er om stabiliteitsredenen. In de strategie waarmee `ode23` zijn stapgrootte bestuurt wordt echter geen rekening gehouden met de stabiliteit.

Vraag 5C. Werkt de stapgroottebesturings strategie van `ode23` ook goed in de gevallen waarin EF stabiliteitsproblemen heeft? Welke methode is dan het efficiëntst?

Experiment. Hoe efficiënt is `ode23` voor de GDV in het experiment in Vraag 5A (`Linear2`) voor de waarden 10^{-3} en 10^{-5} voor ε en een nauwkeurigheid van 10^{-3}? Stel eerst vast hoe groot de toleratie *tol* moet zijn om met `ode23` de gewenste nauwkeurigheid te kunnen halen. (Haalt `ode23` de 'beloofde' nauwkeurigheid?) Welke methode (EF, EB of `ode23`) is het efficiëntst? Hangt dat nog van ε af?

Voor $\varepsilon = 10^{-3}$ lijkt `ode23` te winnen. Kijk ook eens naar de nauwkeurigheid in de afgeleide bij `ode23` en EB.[29]

Hoewel `ode23` mischien minder nauwkeurig is dan het op het eerste gezicht lijkt (we hadden overigens alleen een zekere nauwkeurigheid in de oplossing verlangd en niet in diens afgeleide!), lijkt de methode eventuele stabiliteitsproblemen automatisch op te lossen. Hoe komt dat?

Verklaring. Laat je inspireren door de exacte oplossingen door de numerieke verkregen waarden.[30]

Vraag 5D. De strategie waarmee `ode23` de stapgrootte bestuurt is gericht op (lokale) nauwkeurigheid, maar lijkt impliciet stabiliteitstproblemen op te lossen. Is het verstandig om `ode23` voor stijve problemen te gebruiken?

Experiment. Los (1.2) met $\lambda = -400$ (`Linear1`) zo efficiënt mogelijk op met EF en ode23 met een nauwkeurigheid van 10^{-3}.[31] Welke methode is het efficiëntst? Aan welke methode geef je voor deze GDV de voorkeur, EF, EB of `ode23`?

> **Conclusie.** *Stapgroottebesturing gebaseerd op controle van de lokale nauwkeurigheid leidt tot een efficiënte methode in geval de oplossing in delen van het tijdsinterval sterk variëert. De strategie kan ook effectief gebruikt worden voor problemen die maar in een beperkt tijdsinterval stijf zijn en waarbij de stijfheid niet al te groot is.*

---

[27] De exacte oplossing is niet bekend, maar met `PlotError = 1;` krijg je een schatting voor de fout in $t = T$.

[28] Met `PlotDensityStepsizes = 1;` krijg je in figuur 7 de $\log_{10}(1/h_n)$ te zien in iedere t_n. Dit geeft een beeld van de 'dichtheid' van de stappen: in een gebiedje waar de stapgrootten klein zijn (d.w.z. waar de $1/h_n$ groot zijn), worden veel stappen gezet. In hetzelfde figuur staat ook de oplossing. Let op het verband tussen de stapgrootten en veranderingen in de functiewaarden (d.w.z. de grootte van de afgeleiden).

[29] Neem `J = [0,1,2];` en laat de fout plotten: `PlotError = 1;`.

[30] Neem `PlotExactNearbySol = 1;` en kijk naar de gevonden oplossingen voor $t \approx 0$. Tip: inzoomen kan handig met het commando `axis([0,0.08,0.997,1.003])`.

[31] Neem weer `PlorError =1;`.

1.7 Samenvatting

Bij GDVs wordt de fout in het numeriek resultaat bepaald door

1. *de wijze waarop de GDV fouten voortplant,*
2. *de fouten die in iedere stap van het rekenproces geïntroduceerd worden,*
3. *de wijze waarop de numerieke oplosmethode fouten voortplant.*

In §§1.7.1-1.7.3 analyseren we achtereenvolgens deze drie aspecten voor het eendimensionale beginwaarde probleem

$$\begin{cases} u'(t) = \lambda(t)u(t) + g(t), & t \in [0,T] \\ u(0) = u_0. \end{cases} \tag{7.1}$$

Hierin zijn λ en g gegeven reëel (eventueel complex) waardige functies en u_0 is een gegeven scalair. u is de exacte oplossing van (7.1). De functie f is dus in deze §§ gegeven door $f(t,u) = \lambda(t)u + g(t)$. Daarna geven we in §1.7.4 aan hoe de resultaten er voor niet-lineaire – en meerdimensionale GDVs er uit zien.

1.7.1 Foutvoortplanting door de GDV

Een verstoring van bijvoorbeeld de beginvoorwaarde leidt tot een andere oplossing. In sommige gevallen kan die oplossing flink afwijken van de beoogde oplossing. Dat kan zelfs gebeuren voor heel kleine verstoringen. In zo'n geval mogen we niet verwachten dat we de GDV numeriek nauwkeurig kunnen oplossen: immers in een numeriek proces zijn fouten onvermijdelijk (en fouten worden niet alleen in het begin gemaakt!).

Om te achterhalen wat we op zijn minst aan fouten in de numerieke resultaten mogen verwachten analyseren we het effect van een verstoring ε op de beginvoorwaarde in (7.1).

Als $\widetilde{u}$ de 'verstoorde' oplossing is van (7.1), d.w.z. de oplossing van

$$\begin{cases} u'(t) = \lambda(t)u(t) + g(t), & t \in [0,T] \\ u(0) = u_0 + \varepsilon, \end{cases} \tag{7.2}$$

dan geeft

$$e(t) \equiv u(t) - \widetilde{u}(t) \tag{7.3}$$

aan hoe de verstoring van de beginvoorwaarde doorwerkt op de oplossing u in het tijdstip t. Trekken we (7.2) af van (7.1) dan zien we dat

$$\begin{cases} e'(t) = u'(t) - \widetilde{u}'(t) = \lambda(t)\, e(t), \\ e(0) = -\varepsilon. \end{cases} \tag{7.4}$$

Dus

$$e(t) = -c(t)\varepsilon, \quad \text{met} \quad c(t) \equiv \exp\left(\int_0^t \lambda(s)\, ds\right) : \tag{7.5}$$

$c(t)$ geeft aan hoe de verstoring door de GDV voortgeplant wordt.
Met $\lambda_{\min} \equiv \min_t \mathrm{Re}(\lambda(t))$ en $\lambda_{\max} \equiv \max_t \mathrm{Re}(\lambda(t))$ geldt $e^{\lambda_{\min} T} \leq |c(t)| \leq e^{\lambda_{\max} T}$.
Als $\lambda_{\min} > 0$ dan wordt de verstoring vergrotend voortgeplant, als $\lambda_{\max} < 0$ is dat verkleinend. Met bv $\lambda_{\min} = 2$ en $T = 20$ is $e^{\lambda_{\min} T} = 2.4\,10^{17}$ en de GDV vergroot een fout ε in het begin minstens tot een fout $2.4\,10^{17}\varepsilon$ in de eindtijd $T = 20$.

op dat een cumulatie van defecten een orde h groter kan zijn dan de defecten afzonderlijk:

$$|\sum_{j<n} \tau_j| \leq \sum_{j<n} |\tau_j| \leq n \max_j |\tau_j| \leq \frac{T}{h} \max_j |\tau_j|.$$

Onder omstandigheden kunnen de ongelijkheden gelijkheden zijn.
In het algemeen is $\lambda \neq 0$ en hebben we niet simpel weg te maken met een cumulatie van defecten maar ook nog met de *manier* waarop oude fouten worden voortgeplant. Afhankelijk van de grootte in absolute waarde van $1+h\,\lambda_n$ worden oude fouten groeiend of krimpend voortgeplant. Deze factor wordt daarom de **foutvoortplantings factor** voor EF genoemd. Merk op dat

$$1 + h\,\lambda_n \approx e^{h\,\lambda_n} \quad \text{is als} \quad |h\,\lambda_n| \ll 1 \qquad (\text{denk aan } |h\,\lambda_n| \leq 0.2);$$

in dit geval planten EF en de GDV zelf fouten op min of meer dezelfde manier voort (zie (7.6)).

Als λ begrensd is onafhankelijk van h dan kunnen we de fout majoreren door een constante maal h.

Stelling 1.7.1 *Als $|\lambda|$ begrensd is op $[0,T]$ dan is er een $C > 0$ zodat voor iedere h, die voldoende klein is, en iedere n waarvoor $t_n \leq T$, geldt dat*

$$|u(t_n) - u_n| \leq C \sum_{j<n} |\tau_j| \leq C \left(\tfrac{1}{2} T \max_{\xi \leq T} |u''(\xi)| \right) h. \tag{7.10}$$

Door h naar 0 te laten gaan volgt **convergentie**.

Stelling 1.7.2 *Als λ begrensd is dan convergeert EF* **uniform** *op $[0,T]$, d.w.z.,*

$$\sup_n |u(t_n) - u_n| \to 0 \quad \textit{voor} \quad h \to 0$$

Hierbij is het supremum genomen over alle n waarvoor $t_n \leq T$.

Schatting (7.10) is niet erg praktisch: in de praktijk kennen we u'' niet. Schatting (7.10) geldt ook voor niet lineaire GDVs en dan kennen we zelfs C niet (zie §1.7.3). Gelukkig vertoont de fout, voor h voldoende klein, een "h-gedrag": er is een functie w, die niet afhangt van h en n, zodanig dat

$$u(t_n) - u_n \approx h\,w(t_n), \tag{7.11}$$

voor h voldoende klein (zie stelling 1.7.4). Omdat de fout zo fraai gestructureerd is kunnen we aan de hand van de numerieke oplossingen zelf de fout schatten:

Stelling 1.7.3 *Als $\tau \in [0,T]$ en h is zo dat $\ell h = \tau$ en voldoende klein is, dan geldt*

$$u(\tau) - u_{2\ell}^{(2)} \approx \tfrac{1}{2} \left[u(\tau) - u_{\ell}^{(1)} \right], \tag{7.12}$$

hierin is $u_n^{(1)}$ de EF benadering met stapgrootte h en $u_n^{(2)}$ die met stapgrootte $\frac{1}{2}h$.

De fout halveert dus min of meer als de stapgrootte halveert. De fout moet uiteraard wel gemeten worden in geschikte punten.

(ii) hogere orde Taylor resttermen verwaarloosbaar zijn ten opzichte van de laagste.

Als $-M \leq \lambda(t) \leq 0$ voor alle t dan geldt (i.2) als $h \leq 2/M$ en voor dit soort h hangt verder het foutgedrag nog maar alleen af van Taylor resttermen. Voor bv. $M = 5$ is $2/M = 0.4$ en vormt de stabiliteitseis (i.2) niet echt een extra beperking. Voor grote M (denk aan $M \geq 10^4$) kan dat wel het geval zijn. Andere numerieke oplosmethoden met betere stabiliteitseigenschappen kunnen dan uitkomst bieden. Een variant op de afleiding van EF leidt tot zo'n methode.

Voor kleine h geldt ook $u(t-h) \approx u(t) - h\,u'(t)$ of, equivalent hiermee, $u(t) \approx u(t-h) + h\,f(t,u(t))$. Dit leidt tot de **Euler Backward** (EB) methode:

$$\begin{cases} u_n = u_{n-1} + h\,f(t_n, u_n), \\ u(t_n) = u(t_{n-1}) + h\,f(t_n, u(t_n)) + \tau_n \quad \text{met} \quad \tau_n \equiv -\frac{1}{2}\,h^2\,u''(\xi_n). \end{cases} \tag{7.15}$$

De eerste relatie is het rekenschema. Voor de fout $e_n \equiv u(t_n) - u_n$ geldt

$$e_n = (1 - h\,\lambda_n)^{-1}\,(e_{n-1} + \tau_n)\,. \tag{7.16}$$

Fouten in EB worden niet vergroot als

$$\left|(1 - h\,\lambda_n)^{-1}\right| \leq 1 \quad \text{voor alle} \quad n. \tag{7.17}$$

In dat geval kan de fout gemajoreerd worden door $\sum_{j<n} |\tau_j|$.
Als bv. $\mathrm{Re}(\lambda(t)) \leq 0$ voor alle t, dan geldt (7.17) voor iedere h en de nauwkeurigheid in de numerieke oplossing wordt volledig bepaald door de defecten, ongeacht hoe groot $|\lambda(t)|$ is. Als $\mathrm{Re}(\lambda(t)) \leq 0$ dan hoeven we ons met EB alleen om de grootte van de defecten te bekommeren, terwijl we met EF ook nog rekening moeten houden met de stabiliteit. Voor zogenaamde **stijve** GDVs, GDVs waarbij $\mathrm{Re}(\lambda(t))$ negatief is maar absoluut groot en $|u''|$ niet al te groot, kunnen we met EB met veel grotere stappen werken dan met EF. Helaas kleeft er ook een nadeel aan EB. In iedere stap moet u_n uit de vergelijking $u_n - h\,f(t_n, u_n) = u_{n-1}$ *opgelost* worden, terwijl in EF u_n simpel volgt door *evaluatie* van $f(t_{n-1}, u_{n-1})$. EF wordt daarom een **expliciete** methode genoemd, en EB noemt men een **impliciete** methode.

1.7.4 Niet-lineaire en meerdimensionale GDVs

Vervangen we in geval de eendimensionale GDV niet-lineair is de functie $\lambda(t)$ in de uitspraken over fouten door

$$\lambda(t) = \frac{\partial f}{\partial u}(t, u(t))$$

dan vinden we in essentie de correcte uitspraken voor niet-lineaire GDVs. Dit laat zich als volgt verklaren. Stellen we voor het niet-lineaire geval relaties op voor de fout dan lopen we aan tegen $f(t,u(t)) - f(t,u(t)+e)$ hetgeen min of meer gelijk is aan $\frac{\partial f}{\partial u}(t,u(t))e$.

Het rekenschema voor meerdimensionale problemen is exact gelijk aan dat voor een-dimensionale. De discussie betreffende nauwkeurigheid en stabiliteit loopt ook niet wezenlijk anders. In plaats van scalairen komen in de discussie vectoren en matrices voor. De grootheden u_n^* en u_n zijn dan d-dimensionale vectoren terwijl voor $\frac{\partial f}{\partial u}$ de Jacobi matrix $(\frac{\partial f_j}{\partial u_i})$ gelezen moet worden: $\frac{\partial f_j}{\partial u_i}$ is het (i,j)-de element van die matrix. De absolute waarde $|v|$ van de vector $v = (v_1, \ldots, v_d)^T$ moet men, voor 'n geschikte "norm",