stichting
mathematisch
centrum

MC

REKENAFDELING NR 12/70 JULI

W.E. BAANSTRA

METHODEN VOOR HET OPLOSSEN VAN BEGINWAARDE PROBLEMEN

RA

2e boerhaavestraat 49 amsterdam

INHOUD

Voorwoord

Deze scriptie bevat een aantal ALGOL 60 programma's, die beginwaarde problemen oplossen.
De programma's zijn getest op een Electrologica X8 computer met gebruikmaking van het "MC ALGOL-60-systeem voor de EL X8" [9].

De schrijver wil zijn dank betuigen aan Prof.Dr.Ir. A. van Wijngaarden, Dr. P.J. v.d. Houwen en Dr. T.J. Dekker voor hun waardevolle suggesties, aan H.L. Oudshoorn voor het corrigeren van de tekst, aan A.C. IJsselstein voor het verzorgen van de grafieken, aan mej. de Jong voor het typen van deze scriptie, aan mevr. Homburg voor het ponsen en aan D. Zwarst en J. Suiker voor het drukken en binden.

Hoofdstuk I

§1. Inleiding

Er zijn verschillende methodes ontwikkeld om gewone differentiaalvergelijkingen met beginvoorwaarden op te lossen.
Eén ervan, de Runge-Kutta methode, zullen we in dit hoofdstuk kort bespreken.
Voor details zij verwezen naar Zonneveld [1] waar de afleiding van de formules gevonden kan worden en voor een beschrijving van de achtergronden, die ten grondslag liggen aan genoemde methode naar Dekker [2].

§2. De Runge-Kutta methode

Zij gegeven het stelsel differentiaalvergelijkingen

$$(2.1)\quad \begin{cases} \dfrac{dy_j}{dt} = f_j\,(t, y_1, \ldots, y_n) \\ \text{met beginvoorwaarden} \\ y_j(t_0) = Y_j,\ j = 1(1)n. \end{cases}$$

Laten $\{y_j(t)\}_{j=1}^{n}$ de exacte oplossingen zijn van (2.1) en $\{y_j^*(t)\}_{j=1}^{n}$ de numerieke berekende.

Om $y_j(t)$ in het punt $t = t_0 + h$ (hierin is $h (h > 0)$ de <u>stap grootte</u>) te berekenen is door Runge en Kutta het volgende idee ontwikkeld (zie ook van Wijngaarden [3] blz. 97 e.v.).
Zij $k_{ij} = hf_j(t_i, Y_1 + \sum_{l=0}^{i-1} L_{il}\, k_{l1}, \ldots, Y_n + \sum_{l=0}^{i-1} L_{il}\, k_{ln})$, met $t_i = t_0 + M_i h$, $M_0 = 0$.

Bereken achtereenvolgens de m grootheden $k_0, k_1, \ldots, k_{m-1}$ en vorm het gewogen gemiddelde

$$(2.2)\qquad \Delta y_j = \sum_{i=0}^{m-1} a_i\, k_{ij},\ m = 1, 2, \ldots .$$

De parameters M_i, L_{il} en a_i worden zo gekozen dat

$$y_j^*(t_0+h) = Y_j + \Delta y_j$$

een goede benadering voor $y_j(t_0+h)$ is.
Daartoe worden $y_j^*(t_0+h)$ en $y_j(t_0+h)$ ontwikkeld in machtreeksen.
Bij vast gekozen m kiezen we de genoemde parameters nu zo dat zoveel mogelijk termen van de beide machtreeksen overeenstemmen. Dit leidt tot een stelsel niet lineaire vergelijkingen in de parameters.
De exponent van h in de laatste term die nog overeenstemt noemt men de orde van het Runge-Kutta proces.

§3. De correctie term en de stapgrootte contrôle

Zonneveld [1] heeft zowel de berekening van de correctie term als het aanpassen van de stapgrootte vereenvoudigd.
Zij het Runge-Kutta proces van de orde m, dan is de m-de orde term de laatste die in de machtreeks ontwikkelingen van $y_j(t_0+h)$ en $y_j^*(t_0+h)$ overeenstemt. Noem deze term $th^m\ \Delta y_j$.
Het increment Δy_j (zie (2.2)) schreven we als een lineaire combinatie van k_{ij}'s. Kan men $th^m\ \Delta y_j$ ook door een lineaire combinatie van k_{ij}'s berekenen? Bij een tweede-orde formule lukt dit, bij hogere orde formules hebben we hiervoor minstens één k_{ij} meer nodig. Hierdoor hebben we de laatste term van de machtreeks ontwikkelingen die door het Runge-Kutta proces in rekening wordt gebracht expliciet in handen.
De term $th^m\ \Delta y_j$ is nu te beschouwen als de correctieterm, die gebruikt kan worden om de stapgrootte h aan te passen (zie [3] blz. 90 e.v. en [2] blz. 228 e.v.).

§4. De stabiliteit van de Runge-Kutta formules

(4.1) Definitie

De m-de orde Runge-Kutta formule is in het geval waarin de Jacobiaan van het stelsel reële eigenwaarden λ_j heeft stabiel als deze eigenwaarden voldoen aan:

$$\left| \sum_{\nu=0}^{m} \frac{h^{\nu}}{\nu!} \lambda_j \right| \leq 1, \quad j = 1(1)n,$$

(zie Wilf [4]).

We zullen slechts de stabiliteits voorwaarden geven voor de tweede-, derde- en vierde-orde formules.

De theoretische afleiding van deze stabiliteitsvoorwaarden kan men wat de vierde-orde formule betreft vinden in Abbas I. Abdel Karim [5].

De stabiliteitsvoorwaarden van de tweede- en derde-orde formules zijn op analoge wijze af te leiden.

Voor de tweede-orde formules gelden de volgende stabiliteitsvoorwaarden:

$$(4.2) \qquad \begin{cases} \lambda_j \leq 0, \\ -2 \leq h\,\lambda_j \leq 0. \end{cases}$$

Voor de derde-orde formules geldt:

$$(4.3) \qquad \begin{cases} \lambda_j \leq 0, \\ -2.52 \leq h\,\lambda_j \leq 0, \end{cases}$$

en voor de vierde-orde formules krijgen we:

$$(4.4) \qquad \begin{cases} \lambda_j \leq 0, \\ -2.78 \leq h\,\lambda_j \leq 0, \quad j = 1(1)n. \end{cases}$$

§5. De tweede-, derde- en vierde-orde Runge-Kutta formules

Tweede orde:

$$k_{0j} = hf_j\,(t_0, Y_1, \ldots, Y_n),$$

$$k_{1j} = hf_j\,(t_0+h, Y_1+k_{01}, \ldots, Y_n+k_{0n}),$$

$$y_j^*(t_0+h) = Y_j + \frac{k_{0j} + k_{1j}}{2},$$

$$\mathrm{th}^2\,\Delta y_j = \frac{-k_{0j} + k_{1j}}{2}.$$

Derde orde:

$$
(5.2)\qquad
\begin{cases}
k_{0j} = hf_j\,(t_0, Y_1, \ldots, Y_n), \\
k_{1j} = hf_j\,(t_0 + \frac{h}{3},\ Y_1 + \frac{k_{01}}{3}, \ldots,\ Y_n + \frac{k_{0n}}{3}), \\
k_{2j} = hf_j\,(t_0 + \frac{2}{3}h,\ Y_1 + \frac{2}{3}k_{11}, \ldots,\ Y_n + \frac{2}{3}k_{1n}), \\
y_j^*(t_0+h) = Y_j + \frac{k_{0j} + 3k_{2j}}{4}, \\
k_{3j} = hf_j\,(t_0+h, y_1^*(t_0+h), \ldots, y_n^*(t_0+h)), \\
th^3\,\Delta y_j = \frac{k_{0j} - 3k_{2j} + 2k_{3j}}{2}\ .
\end{cases}
$$

Als extra punt wordt dus k_{3j} berekend. Uit formule (5.2) volgt echter dat dit tevens het beginpunt is voor de volgende integratiestap en dit punt moest dus toch al berekend worden.
Hieruit kunnen we concluderen dat na het accepteren van een stapgrootte h de functie f_j slechts drie maal berekend behoeft te worden (zie [1]).

Vierde orde:

$$
(5.3)\qquad
\begin{cases}
k_{0j} = hf_j\,(t_0, Y_1, \ldots, Y_n), \\
k_{1j} = hf_j\,(t_0 + \frac{h}{2},\ Y_1 + \frac{k_{01}}{2}, \ldots,\ Y_n + \frac{k_{0n}}{2}), \\
k_{2j} = hf_j\,(t_0 + \frac{h}{2},\ Y_1 + \frac{k_{11}}{2}, \ldots,\ Y_n + \frac{k_{1n}}{2}), \\
k_{3j} = hf_j\,(t_0 + h,\ Y_1 + k_{21}, \ldots,\ Y_n + k_{2n}), \\
y_j^*(t_0+h) = Y_j + \frac{k_{0j} + 2(k_{1j}+k_{2j}) + k_{3j}}{6}, \\
k_{4j} = hf_j\,(t_0 + \frac{3}{4}h, Y_1 + \frac{5k_{01}+7k_{11}+13k_{21}-k_{31}}{32}, \ldots, \\
\qquad\qquad Y_n + \frac{5k_{0n}+7k_{1n}+13k_{2n}-k_{3n}}{32}), \\
th^4\,\Delta y_j = \frac{2(-k_{0j}+3(k_{1j}+k_{2j}+k_{3j})-8k_{4j})}{3},\ j = 1(1)n.
\end{cases}
$$

In [1] kan men bovendien de eerste- en vijfde-orde formules vinden.

Hoofdstuk II

§1. Inleiding

In dit hoofdstuk zullen we een numerieke integratiemethode bespreken, die ontwikkeld is door Fowler en Warten [6].
We beschouwen hiertoe weer het stelsel gewone differentiaalvergelijkingen gegeven in formule (2.1) van het vorige hoofdstuk.
In vector notatie kunnen we dit stelsel als volgt schrijven:

$$(1.1)\qquad \begin{cases} \vec{y}' = \vec{f}(t,\vec{y}), \\ \vec{y}(t_0) = \vec{Y}, \\ \text{met } \vec{y} = \vec{y}(t),\ \vec{y} = (y_1,\ldots,y_n). \end{cases}$$

Dit stelsel kunnen we als volgt linearizeren:

Zij $h(h > 0)$ de stapgrootte, dan is

$$(1.2)\qquad \vec{y}'(t_0+h) = \vec{f}(t_0,\vec{Y}) + h\vec{f}_t(t_0,\vec{Y}) + h\vec{f}_y(t_0,\vec{Y})\vec{f}(t_0,\vec{Y}) + O(h^2)$$

en

$$(1.3)\qquad \vec{y}(t_0+h) = \vec{Y} + h\vec{f}(t_0,\vec{Y}) + O(h^2).$$

Formule (1.3) levert:

$$\frac{\vec{y}(t_0+h) - \vec{Y}}{h} = \vec{f}(t_0,\vec{Y}) + O(h).$$

Dit gesubstitueerd in (1.2) geeft:

$$\vec{y}'(t_0+h) = \vec{f}(t_0,\vec{Y}) + h\vec{f}_t(t_0,\vec{Y}) + \vec{f}_y(t_0,\vec{Y})\{\vec{y}(t_0+h)-\vec{Y}\} + O(h^2)$$

of ook

$$\vec{y}'(t_0+h) = D(t_0)\,\vec{y}(t_0+h) + \vec{f}(t_0,\vec{Y}) + h\vec{f}_t(t_0,\vec{Y}) - D(t_0)\,\vec{Y} + O(h^2),$$

waarin $D(t_0) = \left\{ \frac{\partial f_i(t_0, \vec{Y})}{\partial y_j} \right\}$ een $n \times n$ matrix is.

De door Fowler en Warten ontwikkelde methode is er op gericht om die stelsels differentiaalvergelijkingen numeriek op te lossen, waarvan de eigenwaarden van de matrix $D(t_0)$ in absolute waarde sterk in grootte varieren. Zij hebben hiertoe een tweede-orde tweestap methode ontwikkeld, die grotere waarden van de stapgrootte h toelaat, dan bij de Runge-Kutta methodes het geval is, terwijl de stabiliteit gehandhaafd blijft.

§2. De Fowler-Warten methode

Om de achtergrond van de formules van Fowler en Warten toe te lichten gaan we uit van het homogene lineaire stelsel:

$$(2.1) \qquad \begin{cases} \vec{y}' = D\vec{y}, \ \vec{y} = \vec{y}(t), \\ \vec{y}(t_0) = \vec{Y}. \end{cases}$$

Hierin is D een $n \times n$ matrix, onafhankelijk van t, met reële verschillende eigenwaarden, die in absolute waarde sterk in grootte verschillen.
De exacte oplossing van (2.1) is:

$$\vec{y}(t) = e^{Dt}\,\vec{Y}.$$

Dit levert

$$(2.2) \qquad \begin{cases} \vec{y}(t_0+h) = e^{Dh}\,\vec{Y} \\ \qquad\qquad = \vec{Y} + \frac{e^{Dh}-1}{D}\,\vec{Y}'. \end{cases}$$

Het essentiële verschil tussen de Runge-Kutta methode en die van Fowler en Warten is dat de eerstgenoemde methode gebaseerd is op

een <u>polynoom-benadering</u> van de operator e^{Dh} en de laatstgenoemde op een <u>diagonaal-matrix</u> benadering van de operator $\frac{e^{Dh}-1}{D}$, d.w.z.

$$\frac{e^{Dh}-1}{D} \to \frac{e^{\vec{\lambda}h}-1}{\vec{\lambda}}, \quad \vec{\lambda} = (\lambda_1,\ldots,\lambda_n). \tag{2.3}$$

Indien we de numeriek berekende oplossing van (2.1) voorstellen door $\vec{y}^*(t)$ krijgen we m.b.v. (2.2) en (2.3):

$$\vec{y}^*(t_0+h) = \vec{Y} + \frac{e^{\vec{\lambda}h}-1}{\vec{\lambda}}\ \vec{Y}'. \tag{2.4}$$

Op grond van het bovenstaande kunnen we concluderen dat de componenten van $\vec{\lambda}$ in absolute waarde sterk in grootte zullen varieren.
Laten $(\lambda_1,\ldots,\lambda_r)$ de in absolute waarde kleine componenten van $\vec{\lambda}$ zijn en stel $\vec{\lambda}_p = (\lambda_{r+1},\ldots,\lambda_n)$.
Voor λ_i, i = 1, ..., r krijgen we de <u>asymptotische oplossing</u> van (2.4).
Deze kunnen we schrijven als

$$\vec{y}_A(t_0+h) = \vec{y}_A(t_0) + h\vec{y}'_A(t_0), \tag{2.5}$$

daar voor de genoemde λ_i geldt: $\frac{e^{\lambda_i h}-1}{\lambda_i} \simeq h$.
Voor $\vec{\lambda}_p = (\lambda_{r+1},\ldots,\lambda_n)$ krijgen we de <u>storings oplossing</u> van (2.4).
Deze schrijven we als

$$\vec{y}_{PE}(t_0+h) = \vec{y}_{PE}(t_0) + \frac{e^{\vec{\lambda}_p h}-1}{\vec{\lambda}_p}\ \vec{y}'_{PE}(t_0). \tag{2.6}$$

Uit het bovenstaande volgt direkt dat $\vec{y}^*(t_0+h)$ te schrijven is als:

$$\vec{y}^*(t_0+h) = \vec{y}_A(t_0+h) + \vec{y}_{PE}(t_0+h),$$

en dit levert m.b.v. (2.5) en (2.6)

$$\vec{y}^*(t_0+h) = \vec{y}_A(t_0) + \vec{y}_{PE}(t_0) + h\vec{y}'_A(t_0) + \frac{e^{\vec{\lambda}_p h}-1}{\vec{\lambda}_p}\ \vec{y}'_{PE}(t_0). \tag{2.7}$$

Hieruit volgt dat als h_k de stapgrootte is van t_k tot $t_{k+1} = t_k + h_k$, $k = 0, 1, 2, \ldots$ we (2.7) kunnen schrijven als:

$$(2.8)\quad \vec{y}^{*}(t_{k+1}) = \vec{y}_A(t_k) + \vec{y}_{PE}(t_k) + h_k\vec{y}'_A(t_k) + \frac{e^{\vec{\lambda}_p h_k} - 1}{\vec{\lambda}_p}\vec{y}_{PE}(t_k).$$

Opmerking

Uit het bovenstaande blijkt dat we de numerieke oplossing van het beginwaarde-probleem (1.1) kunnen schrijven als formule (2.8).

In formule (2.8) komen vijf onbekenden voor (zie [6]) nl.
$\vec{y}_A(t_k)$, $\vec{y}_{PE}(t_k)$, $\vec{y}'_A(t_k)$, $\vec{y}'_{PE}(t_k)$ en $\vec{\lambda}_p$.
Om deze eenduidig te kunnen bepalen hebben we vijf voorwaarden nodig.
Hiertoe ontwikkelen we de uitdrukking $\frac{e^{\vec{\lambda}_p h_k} - 1}{\vec{\lambda}_p}$ uit formule (2.8) in een Taylorreeks.
Dit geeft

$$(2.9)\quad \begin{cases} \vec{y}\,(t_{k+1}) = \vec{y}_A(t_k) + \vec{y}_{PE}(t_k) + h_k\{\vec{y}'_A(t_k) + \vec{y}'_{PE}(t_k)\} + \frac{h_k^2}{2}\vec{\lambda}_p\vec{y}'_{PE}(t_k) + \\ \qquad \frac{h_k^3}{6}\vec{\lambda}_p^2\,\vec{y}_{PE}(t_k) + O(h_k^4). \end{cases}$$

Als we formule (2.9) vergelijken met de Taylorreeks ontwikkeling van de exacte oplossing van (1.1) in het punt $t_{k+1} = t_k + h_k$ krijgen we voor een tweede-orde integratieproces de volgende drie voorwaarden:

$$(2.10)\quad \begin{cases} \vec{y}_A(t_k) + \vec{y}_{PE}(t_k) = \vec{y}(t_k), \\ \vec{y}'_A(t_k) + \vec{y}'_{PE}(t_k) = \vec{y}'(t_k) \\ \text{en} \\ \vec{\lambda}_p\,\vec{y}'_{PE}(t_k) = \vec{y}''(t_k). \end{cases}$$

Opmerking

De laatste formule van (2.10) wordt door Fowler en Warten [6] echter vervangen door

(2.11) $$\vec{\lambda}_p \, \vec{y}'_{PE}(t_k) = \vec{y}''_p(t_k).$$

Hierin is $\vec{y}''_p(t_k)$ een benadering voor $\vec{y}''(t_k)$.
De methode, die hiervoor is gebruikt, kan men vinden in [6].

De door Fowler en Warten genoemde vijf onbekenden zijn echter afhankelijk. Dit volgt direkt uit de eerste formule van (2.10).
De volgende keuze is dus geen beperking voor de algemene oplossing.

(2.12) $$\vec{y}_A(t_k) = \vec{y}(t_k).$$

De laatste voorwaarde is nu direkt af te leiden uit (2.5) en (2.12).
Formule (2.5) levert:

(2.13) $$\vec{y}_A(t_{k+1}) = \vec{y}_A(t_k) + h_k \, \vec{y}'_A(t_k).$$

Indien we voor dezelfde waarde van k, h_{k-1} definieren als de stapgrootte van het punt $t_{k-1} = t_k - h_{k-1}$ tot t_k kunnen we op grond van (2.13) ook schrijven:

(2.14) $$\vec{y}_A(t_{k-1}) = \vec{y}_A(t_k) - h_{k-1} \, \vec{y}'_A(t_k).$$

De formules (2.14) en (2.12) leveren dan de vijfde voorwaarde:

(2.15) $$\vec{y}'_A(t_k) = \frac{\vec{y}(t_k) - \vec{y}(t_{k-1})}{h_{k-1}}.$$

Opmerking

Een schatting voor de truncation error en een beschrijving van de strategie, die gebruikt wordt om de verschillende stapgroottes te controleren kan men vinden in [6].

§3. De Fowler-Warten methode in detail

In deze paragraaf zullen we een beschrijving geven van de details waaruit de Fowler-Warten methode is opgebouwd.

We onderscheiden:

(3.1) $$\vec{y}'_A(t_k) = \frac{\vec{y}(t_k) - \vec{y}(t_{k-1})}{h_{k-1}},$$

(3.2) $$\vec{d}_1 = \vec{y}'(t_k) - \vec{y}'_A(t_k),$$

(3.3) $$\vec{y}_p(t_k+\delta_k) = \vec{y}(t_k) + \delta_k \vec{y}'(t_k), \text{ met } \delta_k \leq \frac{h_k}{4},$$

(3.4) $$\vec{y}'_p(t_k+\delta_k) = \vec{f}\{t_k+\delta_k, \vec{y}_p(t_k+\delta_k)\},$$

(3.5) $$\vec{d}_2 = \frac{\vec{y}'_p(t_k+\delta_k) - \vec{y}'(t_k)}{\delta_k},$$

(3.6)
$$\vec{\lambda} = \begin{cases} \dfrac{\vec{d}_2}{\vec{d}_1} & \text{als } \vec{d}_1 \neq \underline{0}, \\ \underline{0} & \text{als } \vec{d}_1 = \underline{0}, \end{cases}$$

hierin is $\underline{0}$ de nulvector.

Indien elke component van $\vec{\lambda}$ negatief is stellen we:

$$\vec{c}_1 = \frac{e^{\vec{\lambda}h_k} - 1}{\vec{\lambda}h_k} \text{ en } \vec{c}_0 = e^{\vec{\lambda}h_k},$$

is dit niet het geval dan schrijven we

$$\vec{c}_1 = 1 + \frac{\vec{\lambda}h_k}{2} \text{ en } \vec{c}_0 = 1 + \vec{\lambda}h_k,$$

(3.7) $$\vec{y}^*(t_{k+1}) = \vec{y}(t_k) + h_k \vec{y}'_A(t_k) + h_k \vec{c}_1 \vec{d}_1,$$

(3.8) $$\vec{y}'^*(t_{k+1}) = \vec{f}(t_{k+1}, \vec{y}^*(t_{k+1})),$$

(3.9) $$\vec{E} = h_k [\vec{y}'^*(t_{k+1}) - (\vec{y}'_A(t_k) + \vec{c}_0 \vec{d}_1)].$$

Opmerking

Onder het in bovenstaande formules voorkomende produkt c.q. quotiënt van vectoren verstaan we het componentsgewijs genomen produkt c.q. quotiënt.

We zullen nu een verklaring geven van bovenstaande formules. Hierbij zullen we onderscheid maken tussen de volgende modificaties.

(3.10) Modificatie 1

Formule (3.1) is identiek aan (2.15). Bij de start van het integratieproces stellen we h_{k-1} en $\vec{y}'_A(t_k)$ gelijk aan nul.

Formule (3.2) is dezelfde als de tweede formule van (2.10) met $\vec{d}_1 = \vec{y}'_{PE}(t_k)$.

Een verklaring voor de formules (3.3), (3.4), (3.5) en (3.6) kan men vinden in [6]. In (3.5) is $\vec{d}_2$ gelijk aan $\vec{y}''_p(t_k)$ (zie (2.11)). Voor $\vec{\lambda}$ zie formule (2.3).

Formule (3.7) is direkt af te leiden m.b.v. (2.9), (2.10) en (3.6).

Formule (3.8) spreekt voor zichzelf.

Een verklaring voor (3.9) vindt men in [6].

(3.11) Modificatie 3

Deze modificatie verschilt slechts van de vorige door een andere keuze voor de startwaarde van $\vec{y}'_A(t_k)$.
$\vec{y}'_A(t_k)$ wordt hierin gelijk gesteld aan $\vec{y}'(t_k)$.

(3.12) Modificatie 4

Beschouw formule (3.6).
Indien elke component van $\vec{\lambda}$ positief is vervangen we $\vec{\lambda}$ door $-\vec{\lambda}$ en $\vec{d}_1$ door $-\vec{d}_1$.
M.b.v. de nieuwe waarden van $\vec{\lambda}$ en $\vec{d}_1$ berekenen we $\vec{y}_A(t_k)$, $\vec{c}_1$ en $\vec{c}_0$ op de volgende manier:

$$\vec{y}_A(t_k) = \vec{y}'(t_k) - \vec{d}_1,\ \vec{c}_1 = \frac{e^{\vec{\lambda}h_k} - 1}{\vec{\lambda}h_k} \text{ en } \vec{c}_0 = e^{\vec{\lambda}h_k}.$$

De overige formules worden behandeld als in modificatie 3.

(3.13) Modificatie 2

Het is eenvoudig in te zien dat de beschreven methode van Fowler en Warten niet exact is voor lineaire beginwaarde-problemen.

Beschouw de differentiaalvergelijking:

$$y' = \lambda y + b,\ y(t_0) = y_0,$$

λ en b constant.

De algemene oplossing hiervan is

(3.14) $$y = ce^{\lambda t} - \frac{b}{\lambda},$$

hierin is c een constante, die uit de beginvoorwaarde is te berekenen.

Daar λ negatief moet zijn (zie §4) kunnen we (3.14) als volgt meetkundig voorstellen:

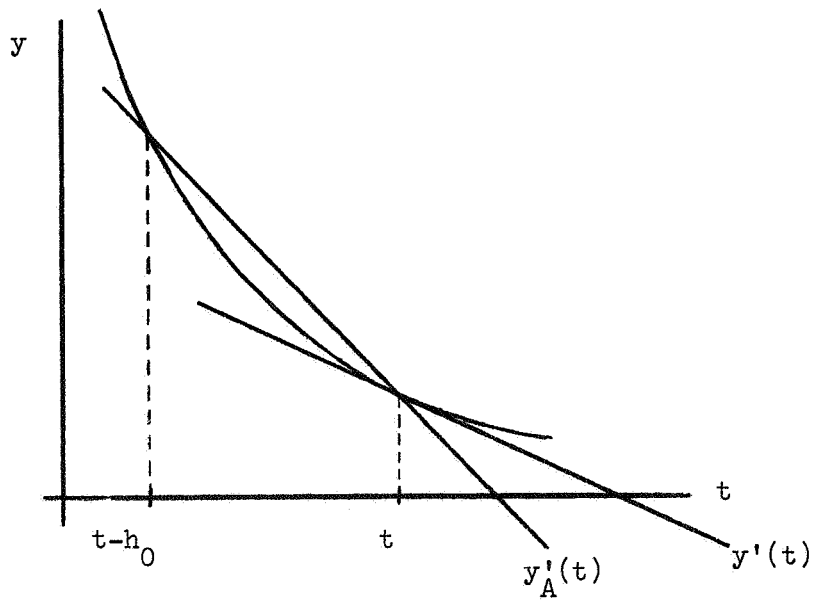

fig. 1

Uit fig. 1 blijkt dat:

(3.15) $$d_1 = y'(t_k) - y'_A(t_k) > 0$$

en

(3.16) $$d_2 = \lambda y'(t_k) > 0.$$

daar $\lambda < 0$.

In de besproken modificaties berekenden we λ zoals in (3.6) wordt aangegeven.
Voor lineaire beginwaarde-problemen echter zullen we, indien d_1 en d_2 hetzelfde teken hebben, modificatie 2 gebruiken, d.w.z. $y'_A(t_k)$ gelijk aan nul en d_1 gelijk aan $y'(t_k)$ stellen.
Uit (3.16) volgt dan dat we op deze wijze λ exact kunnen berekenen.
Geldt het bovenstaande voor een of andere lineair beginwaarde-probleem echter niet, dan berekenen we de oplossing m.b.v. een der andere modificaties.

Opmerking

Uit het bovenstaande volgt tevens dat modificatie 2 ook wordt gebruikt voor lineaire beginwaarde-problemen als λ, d_1 en $d_2 > 0$.

§4. De stabiliteit van de Fowler-Warten methode

In deze paragraaf zullen we de stabiliteitsanalyse geven voor de methode van Fowler en Warten zoals deze door van der Houwen ongepubliceerd gegeven werd.

Laten h_k en h_{k-1} de stapgroottes zijn van t_k tot $t_{k+1} = t_k + h_k$, $t_{k-1} = t_k - h_{k-1}$ tot t_k en stel $\vec{y}_k = \vec{y}(t_k)$, $\vec{y}_{k+1} = \vec{y}(t_{k+1})$ en $\vec{y}_{k-1} = \vec{y}(t_{k-1})$, $k = 1, 2, \ldots$.
Verder stellen we dat $\vec{y}^*_k$ de numeriek berekende waarde is van $\vec{y}_k$.
Dan geldt dat de fout $\vec{\varepsilon}_k$ in de k-de stap gelijk is aan:

(4.1) $$\vec{\varepsilon}_k = \vec{y}^*_k - \vec{y}_k.$$

We zullen nu onderzoeken of $\vec{\varepsilon}_k$ begrensd blijft.
Dit hangt af van:

(4.2) De <u>stabiliteits eigenschappen</u> van het beginwaarde-probleem,

(4.3) de <u>stabiliteit</u> van de gebruikte numerieke benaderings-methode.

De definitie van (4.2) kan men vinden in Liniger [7].
We zullen (4.3) nader bespreken.

Substitutie van (4.1) in (2.8) levert m.b.v. (2.10), (2.15) en (3.6):

$$(4.4)\begin{cases} \vec{y}_{k+1} + \vec{\varepsilon}_{k+1} = \vec{y}_k + \vec{\varepsilon}_k + \dfrac{h_k}{h_{k-1}} (\vec{y}_k - \vec{y}_{k-1} + \vec{\varepsilon}_k - \vec{\varepsilon}_{k+1}) + \\ \qquad h_k \vec{c}_1 \{\vec{f}(t,\vec{y}_k+\vec{\varepsilon}_k) - \dfrac{1}{h_{k-1}} (\vec{y}_k-\vec{y}_{k-1}+\vec{\varepsilon}_k-\vec{\varepsilon}_{k-1})\}. \end{cases}$$

Nu geldt:

$$\vec{y}_{k+1} = \vec{y}_k + h_k \vec{y}'_k + \tfrac{1}{2} h_k^2 \vec{y}''_k + 0(h_k^2),$$

$$\vec{y}_{k-1} = \vec{y}_k - h_{k-1} \vec{y}'_k + \tfrac{1}{2} h_{k-1}^2 \vec{y}''_k + 0(h_{k-1}^3).$$

Dit gesubstitueerd in (4.4) geeft:

$$\vec{y}_k + h_k \vec{y}'_k + \tfrac{1}{2} h_k^2 \vec{y}''_k + 0(h_k^3) + \vec{\varepsilon}_{k+1} =$$

$$\vec{y}_k + \vec{\varepsilon}_k + \frac{h_k}{h_{k-1}} (\vec{y}_k-\vec{y}_k+h_{k-1} \vec{y}'_k-\tfrac{1}{2} h_{k-1}^2 \vec{y}''_k+0(h_{k-1}^3) +$$

$$\vec{\varepsilon}_k-\vec{\varepsilon}_{k-1}) + h_{k-1} \vec{c}_1 \{\vec{f}(t_k,\vec{y}_k)+\vec{\varepsilon}_k f_y(t_k,\vec{y}_k)+0(\vec{\varepsilon}_k^2) -$$

$$\frac{1}{h_{k-1}} (\vec{y}_k-\vec{y}_k+h_{k-1} \vec{y}'_k-\tfrac{1}{2} h_{k-1}^2 \vec{y}''_k+0(h_{k-1}^3)+\vec{\varepsilon}_k-\vec{\varepsilon}_{k-1})\},$$

hierin is $f_y(t_k,\vec{y}_k)$ de Jacobiaan.

Hieruit volgt:

$$\vec{\varepsilon}_{k+1} = \vec{\varepsilon}_k + \frac{h_k}{h_{k-1}}\vec{\varepsilon}_k - \frac{h_k}{h_{k-1}}\vec{\varepsilon}_{k-1} - \vec{c}_1\frac{h_k}{h_{k-1}}(\vec{\varepsilon}_k - \vec{\varepsilon}_{k-1}) -$$
$$\tfrac{1}{2}\vec{y}''_k(h_k^2 + h_k h_{k-1}) + O(h_k^3) + O(h_k h_{k-1}^2) +$$
$$h_k \vec{c}_1 \vec{y}'_k + h_k \vec{c}_1 f_y(t_k, \vec{y}_k)\vec{\varepsilon}_k + \vec{c}_1 O(h_k \vec{\varepsilon}_k^2) -$$
$$\vec{c}_1 h_k \vec{y}'_k + \tfrac{1}{2}\vec{c}_1 h_k h_{k-1}\vec{y}''_k - \vec{c}_1 O(h_k h_{k-1}^2),$$

of ook

$$(4.5)\qquad \begin{cases} \vec{\varepsilon}_{k+1} = (1 + \dfrac{h_k}{h_{k-1}} - \vec{c}_1\dfrac{h_k}{h_{k-1}} + h_k \vec{c}_1 f_y(t_k, \vec{y}_k))\vec{\varepsilon}_k + \\ (\vec{c}_1 - 1)\dfrac{h_k}{h_{k-1}}\vec{\varepsilon}_{k-1} + \tfrac{1}{2}(\vec{c}_1 h_{k-1} - h_{k-1} - h_k) h_k \vec{y}''_k + \\ O(h_k^3) + O(h_k h_{k-1}^2) + O(h_k \vec{\varepsilon}_k^2). \end{cases}$$

We kunnen (4.5) ook schrijven als:

$$(4.6)\qquad \begin{pmatrix} \vec{\varepsilon}_{k+1} \\ \vec{\varepsilon}_k \end{pmatrix} = \begin{pmatrix} 1 + (1 - \vec{c}_1)\dfrac{h_k}{h_{k-1}} + h_k \vec{c}_1 f_y(t_k, \vec{y}_k) & (\vec{c}_1 - 1)\dfrac{h_k}{h_{k-1}} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \vec{\varepsilon}_k \\ \vec{\varepsilon}_{k-1} \end{pmatrix} + O(h_k^2).$$

Hieruit kunnen we concluderen dat de fout $\vec{\varepsilon}_k$ begrensd blijft indien de eigenwaarden van de matrix operator uit (4.6) in absolute waarde kleiner of gelijk zijn aan 1.

Dit levert (zie van der Houwen [10]):

$$(4.8)\qquad \begin{cases} \lambda_j \leq 0, \\ h_k \left[c_1 \lambda_j + \dfrac{2(1-c_1)}{h_{k-1}} \right] \geq -2, \\ (1-c_1)\, h_k \leq h_{k-1}, \end{cases}$$

hierin is λ_j een eigenwaarde van de Jacobiaan $f_y(t_k, \vec{y}_k)$.

Opmerking

De Fowler-Warten methode is stabiel als voldaan is aan (4.8).

We kunnen (4.8) als volgt meetkundig voorstellen:

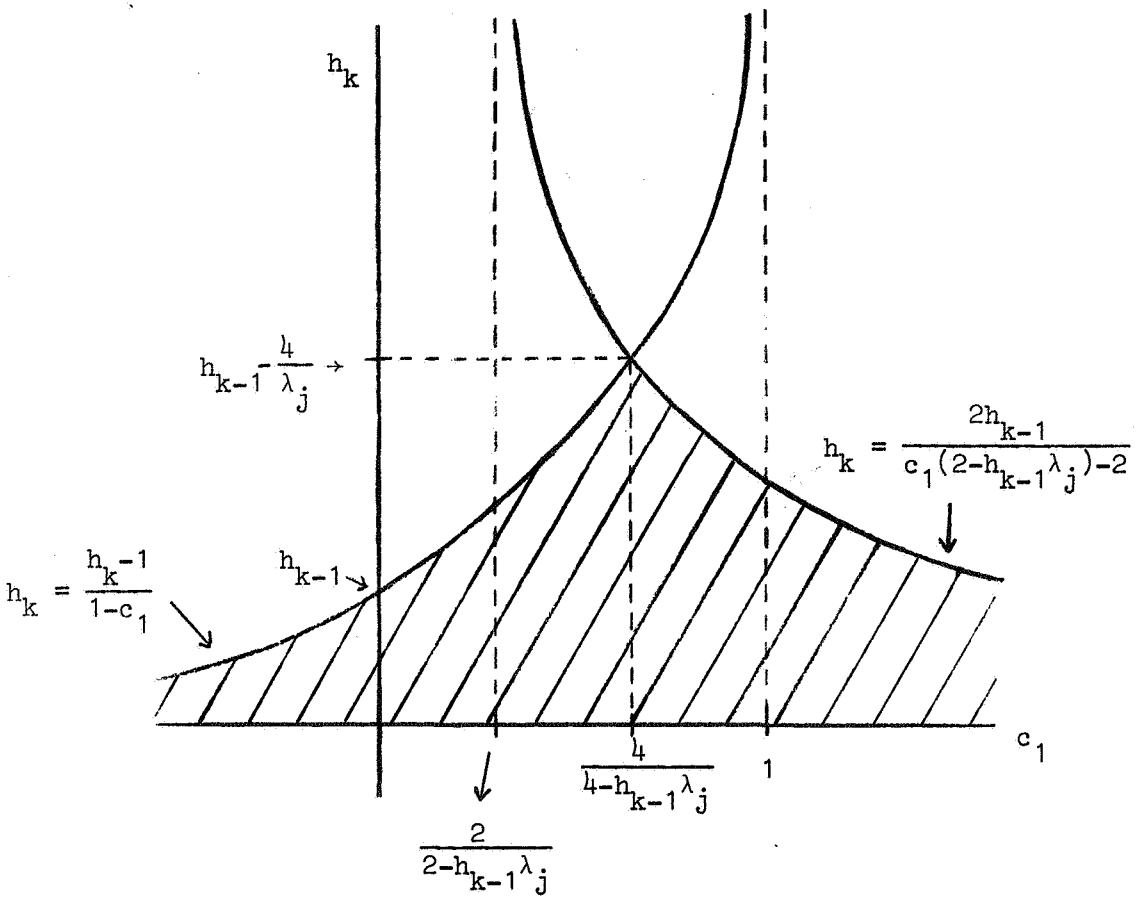

fig. 2.

Opmerking

De waarden van h_k en c_1 moeten in het gearceerde gebied liggen opdat aan de in (4.8) gestelde stabiliteitseisen kan worden voldaan.

§5. Enige kritische opmerkingen

Op een op het Mathematisch Centrum te Amsterdam gehouden werkbespreking [8] is door de Vogelaere (University of Berkeley, California) en Spijker (Rijks Universiteit van Leiden) kritiek geleverd op de in [6] beschreven numerieke integratiemethode.
Hoewel wij het beoordelen van deze kritieken aan de lezer van deze scriptie overlaten, willen we toch nader ingaan op enkele details uit [8].
In de eerste plaats wijzen we er op dat het door de Vogelaere gebruikte voorbeeld in [8] geenszins aantoont dat de Fowler-Warten methode voor een lineaire differentiaalvergelijking met één beginvoorwaarde spaak loopt, hoewel dit in [8] wel wordt verondersteld.
Wij zullen het in [8] behandelde voorbeeld bespreken en daarbij dezelfde notatie gebruiken (zie ook [6]).
De Vogelaere stelt:

Los de differentiaalvergelijking

$$y' = \alpha y,\ y = y(t),$$

$$y(0) = 1$$

op met modificatie 3.
Zij verder

$$-\alpha = 2000,$$

$$h_0 = .01.$$

Dan is:

$$-\alpha h_0 = 20.$$

Verder geldt:

$$y'(0) = \alpha,$$

$$y(t-h_0) = e^{-\alpha h_0} \text{ voor } t = 0.$$

De formules (3.1) - (3.6) leveren:

(5.1) $$y'_A(t) = \frac{y(t) - y(t-h_0)}{h_0} = \frac{1 - e^{-\alpha h_0}}{h_0},$$

(5.2) $$d_1 = y'(t) - y'_A(t) = \alpha - \frac{1 - e^{-\alpha h_0}}{h_0},$$

(5.3) $$y_p(t+\delta) = y(t) + \delta y'(t) = 1 + \delta\alpha,$$

(5.4) $$y'_p(t+\delta) = f(t+\delta, y_p(t+\delta)) = \alpha(1+\delta\alpha) = \alpha + \delta\alpha^2$$

(5.5) $$d_2 = \frac{y'_p(t+\delta) - y'(t)}{\delta} = \frac{\alpha + \delta\alpha^2 - \alpha}{\delta} = \alpha^2$$

(5.6)
$$\lambda = \frac{d_2}{d_1} = \frac{\alpha^2}{\alpha - \frac{1-e^{-\alpha h_0}}{h_0}} = \alpha \frac{1}{1 - \frac{1-e^{-\alpha h_0}}{\alpha h_0}} =$$

$$\alpha \frac{1}{1 - \frac{1-e^{20}}{20}} \simeq \frac{1}{1 - \frac{1-10^9}{20}} \simeq \alpha . 2.10^{-8}.$$

In [8] wordt uit (5.6) geconcludeerd, dat de Fowler-Warten methode voor een eenvoudige lineaire differentiaalvergelijking spaak loopt, daar α en λ volgens [6] dezelfde waarden moeten hebben.

Dat de Fowler-Warten methode in dit geval onjuist is geïnterpreteerd blijkt uit modificatie 2 (zie (3.13)).
Uit (3.13) volgt nl. dat wij lineaire differentiaalvergelijkingen met modificatie 3 kunnen oplossen indien d_1 en d_2 niet hetzelfde teken hebben.

Uit (5.2) en (5.5) volgt echter dat wij in dit geval modificatie 2 moeten gebruiken.
Hierdoor krijgen wij i.p.v. (5.6):

$$y_A'(t) = 0,$$

$$d_1 = y'(t),$$

$$\lambda = \frac{d_2}{d_1} = \frac{\alpha^2}{\alpha} = \alpha.$$

Hieruit zien we, dat λ exact wordt berekend.

Opmerking

Wij willen er toch op wijzen, dat de kritiek van de Vogelaere in feite behelst dat de Fowler-Warten methode niet de mogelijkheid open heeft gelaten om van de ene modificatie over te gaan op de andere indien dit noodzakelijk is.
Uit het zojuist besproken voorbeeld blijkt dat zijn kritiek volkomen gerechtvaardigd is.

Tot besluit willen wij er op wijzen dat de door Spijker in [8] geleverde kritiek volkomen juist is, d.w.z. de Fowler-Warten methode is geen tweede- maar een eerste-orde integratieproces.
Wij zullen dit nader toelichten.
Formule (3.6) geeft:

$$\lambda = \frac{\vec{d}_2}{\vec{d}_1}.$$

Verder geldt (zie [6]):

$$\vec{d}_2 = \vec{y}''(t_k) + O(\delta_k),\ k = 1, 2, \ldots .$$

Uit (3.2) en (3.3) volgt na Taylorreeks ontwikkeling van $\vec{y}(t_{k-1})$ dat:

$$\vec{d}_1 \simeq \frac{h_{k-1}}{2}\vec{y}''(t_k) + O(h_{k-1}^2).$$

Indien h_k en $h_{k-1} \to 0$ krijgen we dus:

$$\vec{\lambda} \simeq \frac{2}{h_{k-1}} = O(h_k^{-1}). \tag{5.7}$$

De truncation error $\vec{E}$ is gelijk aan (zie [6]):

$$\vec{E} = \frac{h_k^3}{6}\{\vec{y}'''(t_k) - \vec{\lambda}_p \vec{y}'_{PE}(t_k)\} - \frac{h_k^2}{4}\delta_k\{\vec{y}'''(t_k) - \vec{y}''(t_k)\vec{f}_y\} + O(h_k^4). \tag{5.8}$$

Uit (5.7) en (5.8) volgt direkt dat de Fowler-Warten methode een eerste-orde integratieproces is.

Hoofdstuk III

§1. Inleiding

In dit hoofdstuk zullen wij een viertal procedures bespreken, die geschreven zijn in ALGOL 60.
De procedures RUNGE 2n, RUNGE 3n en RUNGE 4n zijn gebaseerd op de formules (5.1), (5.2) en (5.3) uit §5 van hoofdstuk I.
De details waaruit de procedure DIFFSYS is opgebouwd kan men vinden in §3 van het vorige hoofdstuk.
In §4 zullen we enige test resultaten van de genoemde integratie-procedures geven.
Vervolgens zullen we die formele parameters bespreken, die de vier bovengenoemde procedures gemeenschappelijk hebben.

t: Dit is de onafhankelijke variabele. Na een aanroep van elk der genoemde procedures is t gelijk aan b.

a: De startwaarde van t.

b: Een parameter, die in de value lijst voorkomt en die de laatste waarde, die t gekregen heeft, aflevert.

y: Een array met elementen y[1], ..., y[n]. Dit zijn de afhankelijke variabelen.

ya: Een array met elementen ya[1], ..., ya[n]. Dit zijn de startwaarden van y[j], j = 1(1)n.

fi: Een variabele van het type boolean.
Indien fi de waarde true heeft dan start de integratie bij a met een stapgrootte, die gelijk is aan b - a.
Is dit niet het geval dan wordt t.a.v. de procedures RUNGE 2n, RUNGE 3n en RUNGE 4n de integratie voortgezet met de beginvoorwaarden:

$$t = d[3],$$

$$y[j] = d[j+3],\ j = 1(1)n$$

en de stapgrootte h wordt dan berekend volgens

(1.1) $$h = d[2] * \text{sign}\,(b-d[3]).$$

Voor DIFFSYS geldt echter dat de integratie wordt voortgezet met de beginvoorwaarden:

$$t = d[5],$$

$$y[j] = d[j+5],\ j = 1(1)n$$

en dat de stapgrootte h_k, k = 1, 2, ... wordt berekend m.b.v. de relatie:

$$h_k = d[2] * \text{sign}\,(b-d[5]).$$

De waarden, die a en ya[j] hadden worden dan niet meer in het integratie proces betrokken (voor de betekenis van het array d zie §2 en 3 van dit hoofdstuk).

n: Het aantal differentiaal vergelijkingen dat opgelost moet worden.

§2. De procedures RUNGE 2n, RUNGE 3n en RUNGE 4n

De genoemde procedures worden gebruikt om stelsels differentiaalvergelijkingen

$$\frac{dy_j}{dt} = f_j(t,y_1,\ldots,y_n),$$

$$y_j(t_0) = Y_j,\ j = 1(1)n$$

op te lossen m.b.v. de formules (5.1), (5.2) en (5.3) uit hoofdstuk I.

Eerst zullen wij de formele parameters, die in deze procedures voorkomen bespreken (zie ook §1 van dit hoofdstuk).

ftyj: Een expressie die afhangt van t, y[j] en j en die de waarde van $\frac{dy_j}{dt}$ aflevert.

j: Een variabele van het type integer. In de actuale parameterlijst correspondeert deze met ftyj en wel om aan te geven welke van de n differentiaal vergelijkingen er opgelost moeten worden.

e: Een array met elementen e[1], ..., e[2*n].
e[2*j-1] is een relatieve en e[2*j] een absolute tolerantie voor y[j].
De tolerantie wordt gedefinieerd als:

tol = (abs(ftyj) * e[2*j-1] + e[2*j] * abs(h)) / int,

met int = abs(b-(if fi then a else d[3])).

d: Een array met elementen d[1], ..., d[n+3].
Na een aanroep van elk der genoemde procedures hebben de elementen van d de volgende waarden:
De waarde van d[1] is gelijk aan het aantal verworpen stappen, d[2] levert de stapgrootte, d[3] de waarde van b en d[4], ..., d[n+3] zijn gelijk aan y[1], ..., y[n] voor t = b.

Tenslotte willen wij er op wijzen dat de stapgrootte h wordt verworpen als geldt:

$$th^2 \Delta y_j > tol,$$

voor elke waarde van j, j = 1(1)n.

```
procedure RUNGE2n(t,a,b,y,ya,ftyj,j,e,d,fi,n); value b,fi,n;
integer j,n; real t,a,b,ftyj; Boolean fi; array y,ya,e,d;
begin integer jj;
      real tl,h,fhm,int,hl,absh,discr,tol,mu,mu1,fh;
      Boolean last,first,reject;
      array yl,k0,k1[1:n],ee[1:2×n];
      if fi then begin d[3]:= a;
                       for jj:= 1 step 1 until n do
                       d[jj+3]:= ya[jj]
                 end;
      d[1]:= 0; tl:= d[3];
      for jj:= 1 step 1 until n do yl[jj]:= d[jj+3];
      if fi then d[2]:= b-d[3]; absh:= h:= abs(d[2]);
      if b-tl < 0 then h:= -h; int:= abs(b-tl);
      for jj:= 1 step 1 until 2×n do ee[jj]:= e[jj]/int;
      first:= true;
      if fi then begin last:= true; goto step
                 end;
test: absh:= abs(h);
      if h ≥ b-tl = h ≥ 0 then begin d[2]:= h;
                                     last:= true;
                                     h:= b-tl;
                                     absh:= abs(h)
                                end
                          else last:= false;
step: t:= tl;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj];
      for j:= 1 step 1 until n do
      k0[j]:= ftyj×h;
      t:= if last then b else tl+h;
      for jj:= 1 step 1 until n do
      yl[jj]:= yl[jj]+k0[jj];
      for j:= 1 step 1 until n do
      k1[j]:= ftyj×h;
      reject:= false; fhm:= 0;
```

```
        for jj:= 1 step 1 until n do
        begin discr:= abs(-k0[jj]+k1[jj])/2;
              tol:= abs(k0[jj])×ee[2×jj-1]+abshx×ee[2×jj];
              reject:= discr > tol ∨ reject;
              fh:= discr/tol;
              if fh > fhm then fhm:= fh
        end;
        mu:= .95/fhm;
        if reject then begin h:= mu×h; d[1]:= d[1]+1;
                             goto test
                       end;
        if first then begin first:= false; hl:= h;
                            h:= mu×h; goto acc
                      end;
        fh:= mu×h/hl+mu-mu1; hl:= h; h:= fh×h;
acc:    mu1:= mu;
        for jj:= 1 step 1 until n do
        y[jj]:= y1[jj]+(k0[jj]+k1[jj])/2;
        if b ≠ t then begin tl:= t;
                            for jj:= 1 step 1 until n do
                            y1[jj]:= y[jj];
                            goto test
                      end;
        if ¬ last then d[2]:= h; d[3]:= t;
        for jj:= 1 step 1 until n do d[jj+3]:= y[jj]
end     RUNGE2n;
```

```
procedure RUNGE3n(t,a,b,y,ya,ftyj,j,e,d,fi,n); value b,fi,n;
integer j,n; real t,a,b,ftyj; Boolean fi; array y,ya,e,d;
begin integer jj;
      real tl,h,fhm,int,hl,absh,discr,tol,mu,mu1,fh;
      Boolean last,first,reject;
      array yl,k0,k1,k2,k3[1:n],ee[1:2×n];
      if fi then begin d[3]:= a;
                       for jj:= 1 step 1 until n do
                       d[jj+3]:= ya[jj]
                 end;
      d[1]:= 0; tl:= d[3];
      for jj:= 1 step 1 until n do yl[jj]:= d[jj+3];
      if fi then d[2]:= b-d[3]; absh:= h:= abs(d[2]);
      if b-tl < 0 then h:= -h; int:= abs(b-tl);
      for jj:= 1 step 1 until 2×n do ee[jj]:= e[jj]/int;
      first:= true;
      if fi then begin last:= true; goto step
                 end
            else reject:= true;
test: absh:= abs(h);
      if h ≥ b-tl = h ≥ 0 then begin d[2]:= h;
                                     last:= true;
                                     h:= b-tl;
                                     absh:= abs(h)
                                end
                           else begin last:= false;
                                     if ¬ reject then
                                     goto fast
                                end;
step: t:= tl;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj];
      for j:= 1 step 1 until n do
      k0[j]:= ftyj×h;
fast: t:= tl+h/3;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+k0[jj]/3;
      for j:= 1 step 1 until n do
      k1[j]:= ftyj×h;
      t:= tl+2×h/3;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+2×k1[jj]/3;
      for j:= 1 step 1 until n do
      k2[j]:= ftyj×h;
      t:= if last then b else tl+h;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+(k0[jj]+3×k2[jj])/4;
      for j:= 1 step 1 until n do
      k3[j]:= ftyj×h;
      reject:= false; fhm:= 0;
```

```
      for jj:= 1 step 1 until n do
      begin discr:= abs(k0[jj]-3×k2[jj]+2×k3[jj])/2;
            tol:= abs(k0[jj])×ee[2×jj-1]+abs×ee[2×jj];
            reject:= discr > tol ∨ reject;
            fh:= discr/tol;
            if fh > fhm then fhm:= fh
      end;
      mu:= 1/(1+fhm×fhm)+.45;
      if reject then begin h:= mu×h; d[1]:= d[1]+1;
                           goto test
                     end;
      if first then begin first:= false; hl:= h;
                           h:= mu×h; goto acc
                    end;
      fh:= mu×h/hl+mu-mu1; hl:= h; h:= fh×h;
acc:  mu1:= mu;
      for j:= 1 step 1 until n do
      k0[j]:= k3[j]×h/hl;
      if b ≠ t then begin tl:= t;
                           for jj:= 1 step 1 until n do
                           y1[jj]:= y[jj];
                           goto test
                    end;
      if ¬ last then d[2]:= h; d[3]:= t;
      for jj:= 1 step 1 until n do d[jj+3]:= y[jj]
end   RUNGE3n;
```

```
procedure RUNGE4n(t,a,b,y,ya,ftyj,j,e,d,fi,n); value b,fi,n;
integer j,n; real t,a,b,ftyj; Boolean fi; array y,ya,e,d;
begin integer jj;
      real tl,h,fhm,int,hl,absh,discr,tol,mu,mu1,fh;
      Boolean last,first,reject;
      array yl,k0,k1,k2,k3,k4[1:n],ee[1:2×n];
      if fi then begin d[3]:= a;
                       for jj:= 1 step 1 until n do
                       d[jj+3]:= ya[jj]
                 end;
      d[1]:= 0; tl:= d[3];
      for jj:= 1 step 1 until n do yl[jj]:= d[jj+3];
      if fi then d[2]:= b-d[3]; absh:= h:= abs(d[2]);
      if b-tl < 0 then h:= -h; int:= abs(b-tl);
      for jj:= 1 step 1 until 2×n do ee[jj]:= e[jj]/int;
      first:= true;
      if fi then begin last:= true; goto step
                 end;
test: absh:= abs(h);
      if h ≥ b-tl = h ≥ 0 then begin d[2]:= h;
                                     last:= true;
                                     h:= b-tl;
                                     absh:= abs(h)
                                end
                          else last:= false;
step: t:= tl;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj];
      for j:= 1 step 1 until n do
      k0[j]:= ftyj×h;
      t:= tl+h/2;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+k0[jj]/2;
      for j:= 1 step 1 until n do
      k1[j]:= ftyj×h;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+k1[jj]/2;
      for j:= 1 step 1 until n do
      k2[j]:= ftyj×h;
      t:= tl+h;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+k2[jj];
      for j:= 1 step 1 until n do
      k3[j]:= ftyj×h;
      t:= if last then b else tl+3×h/4;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+(5×k0[jj]+7×k1[jj]+13×k2[jj]-k3[jj])/32;
      for j:= 1 step 1 until n do
      k4[j]:= ftyj×h;
      if ¬ last then t:= tl+h;
      reject:= false; fhm:= 0;
```

```
      for jj:= 1 step 1 until n do
      begin discr:= abs(-k0[jj]+3×(k1[jj]+k2[jj]+k3[jj])-8×k4[jj])×2/3);
            tol:= abs(k0[jj])×ee[2×jj-1]+abshׯee[2×jj];
            reject:= discr > tol ∨ reject;
            fh:= discr/tol;
            if fh > fhm then fhm:= fh
      end;
      mu:= .3+1.27/(1+fhm);
      if reject then begin h:= mu×h; d[1]:= d[1]+1;
                           goto test
                     end;
      if first then begin first:= false; hl:= h;
                           h:= mu×h; goto acc
                    end;
      fh:= mu×h/hl+mu-mu1; hl:= h; h:= fh×h;
acc:  mu1:= mu;
      for jj:= 1 step 1 until n do
      y[jj]:= yl[jj]+(k0[jj]+2×(k1[jj]+k2[jj])+k3[jj])/6;
      if b ≠ t then begin tl:= t;
                           for jj:= 1 step 1 until n do
                           yl[jj]:= y[jj];
                           goto test
                    end;
      if ¬ last then d[2]:= h; d[3]:= t;
      for jj:= 1 step 1 until n do d[jj+3]:= y[jj]
end   RUNGE4n;
```

§3. De procedure DIFFSYS

De procedure DIFFSYS wordt gebruikt om die stelsels differentiaalvergelijkingen op te lossen die beschreven zijn in het vorige hoofdstuk m.b.v. de in §4 van dat hoofdstuk beschreven formules en modificaties.
Eerst zullen wij de formele parameters, die in deze procedure voorkomen bespreken.

Een verklaring voor de parameters f, a, b, y, ya, fi en n kan men in §1 vinden.

d: Een array met elementen d[1], ..., d[n+5].
Na een aanroep van DIFFSYS hebben de elementen van d de volgende waarden:
d[1] levert het aantal verworpen stappen, d[2], d[3] en d[4] zijn respectievelijk gelijk aan de stapgroottes h_k, h_{k-1} en δ_k voor $k = 1, 2, \ldots$, d[5] is gelijk aan b en d[6], ..., d[n+5] zijn gelijk aan y[1], ..., y[n] voor t = b.

dd: Een array met elementen dd[1], ..., dd[n].
dd[1], ..., dd[n] zijn gelijk aan y[1], ..., y[n] voor $t = b - h_{k-1}$, $k = 1, 2, \ldots$.

f: Een procedure met t, y, dy als parameters.
Hierin hebben t en y dezelfde betekenis als in §1, de variabele t komt echter in de value lijst voor.
Verder is dy een array met elementen dy[1], ..., dy[n] en is

$$dy[j] = \frac{dy_j}{dt},\ j = 1(1)n.$$

e: Een array met elementen e[1], ..., e[4].
e[1], ..., e[4] zijn respectievelijk gelijk aan u_1, u_2, l_1 en l_2 (zie [6]).

mod: Een variabele van het type integer.
In de actuale parameter lijst moet deze variabele vervangen worden door één der cijfers 1, 2, 3 of 4 hetgeen bewerkstelligt dat een

stelsel differentiaalvergelijkingen met één der vier modificaties wordt opgelost.

Tenslotte willen wij er op wijzen dat de in deze procedures voorkomende controle van de stapgroottes h_k en δ_k beschreven is in [6].

```
procedure DIFFSYS(t,a,b,y,ya,d,dd,fi,f,e,mod,n);
value b,fi,n; integer mod,n; real t,a,b;
boolean fi; array y,ya,d,dd,e; procedure f;
begin integer i;
      real p,q,tl,tol1,tol2,h0,discr,delta,h,
           absh,absd,L,d2,c1;
      boolean last,acc1,acc2,A,B,C,D;
      array r,yl,yp0,yp1,yc0,yc1,yA,dy,d1,c0[1:n];
      A:= mod = 1; B:= mod = 2; C:= mod = 3; D:= mod = 4;
      if fi then begin d[3]:= 0; d[5]:= a;
                       for i:= 1 step 1 until n do
                       begin if A ∨ B then yA[i]:= 0;
                             d[i+5]:= ya[i]; dd[i]:= 0
                       end
                 end;
      d[1]:= 0; tl:= d[5]; h0:= d[3];
      if fi then begin d[2]:= b-d[5]; d[4]:= .95×d[2]/4 end;
      absh:= h:= abs(d[2]); absd:= delta:= abs(d[4]);
      if b-tl < 0 then begin h:= -h; delta:= -delta end;
      for i:= 1 step 1 until n do
      begin y[i]:= d[i+5]; yl[i]:= dd[i] end;
      f(tl,y,dy);
      if fi then begin for i:= 1 step 1 until n do
                       begin if A ∨ B then d1[i]:= dy[i] else
                             if C ∨ D then begin yA[i]:= dy[i];
                                                 d1[i]:= 0
                                           end
                       end;
                       last:= true; goto STEP
                 end;
START: for i:= 1 step 1 until n do
       begin yA[i]:= (y[i]-yl[i])/h0;
             d1[i]:= dy[i]-yA[i]
       end;
TEST:  absh:= abs(h); absd:= abs(delta);
       if h ≥ b-tl ≥ h ≥ 0 then begin last:= true; d[2]:= h;
                                      h:= b-tl; absh:= abs(h);
                                      d[4]:= delta;
                                      delta:= h×delta/d[2];
                                      absd:= abs(delta)
                                end
                           else last:= false;
STEP:  for i:= 1 step 1 until n do yp0[i]:= y[i]+delta×dy[i];
       f(tl+delta,yp0,yp1);
       for i:= 1 step 1 until n do
       begin d2:= (yp1[i]-dy[i])/delta;
             if B ∧ sign(d2) = sign(d1[i]) then begin yA[i]:= 0;
                                                      d1[i]:= dy[i]
                                                end;
             L:= if d1[i] = 0 then 0 else d2/d1[i];
             p:= L×h; q:= exp(p);
             if L > 0 ∧ D then begin L:= -L;
                                     d1[i]:= -d1[i];
                                     yA[i]:= dy[i]-d1[i];
                                     p:= L×h; q:= exp(p)
                               end;
             if L < 0 then begin c0[i]:= q; c1:= (q-1)/p end
                      else begin c0[i]:= 1+p; c1:= 1+p/2 end;
             yc0[i]:= y[i]+h×(yA[i]+c1×d1[i]);
             r[i]:= abs(yc0[i])
       end;
```

```
        f(if last then b else tl+h,yc0,yc1);
        acc1:= true;
        for i:= 1 step 1 until n do
        begin discr:= abs(yp1[i]-dy[i])×absd/2;
              tol1:= (e[1]+e[2]×r[i])/2;
              tol2:= (e[3]+e[4]×r[i])/2;
              if discr > .75×tol1 then begin delta:= delta/2;
                                             goto CONTR
                                       end;
              acc1:= discr < tol2 ∧ acc1
        end;
        if acc1 then delta:= delta×2;
CONTR:  acc1:= acc2:= false;
        for i:= 1 step 1 until n do
        begin discr:= abs(yc1[i]-yA[i]-c0[i]×d1[i])×absh;
              tol1:= e[1]+e[2]×r[i];
              tol2:= e[3]+e[4]×r[i];
              if discr > 1.5×tol1 then begin h:= h/2;
                                             d[1]:= d[1]+1;
                                             if delta > h/4
                                             then delta:= h/4;
                                             goto TEST
                                       end
                                  else
              if discr > .75×tol1 then acc1:= true
                                  else
              if discr < tol2 then acc2:= true
        end;
        t:= if last then b else tl+h; h0:= h;
        for i:= 1 step 1 until n do
        begin yl[i]:= y[i]; y[i]:= yc0[i];
              dy[i]:= yc1[i]
        end;
        if b ≠ t then begin tl:= t;
                            if acc1 then h:= h/2
                                    else if acc2 then h:= h×2;
                            if delta > h/4 then delta:= h/4;
                            goto START
                      end;
        if ¬ last then begin d[2]:= h; d[4]:= delta end;
        d[5]:= t; d[3]:= h0;
        for i:= 1 step 1 until n do
        begin d[i+5]:= y[i]; dd[i]:= yl[i] end
end     DIFFSYS;
```

§4. Enige testresultaten

In deze paragraaf zullen we enige testresultaten geven van de besproken integratieprocedures..
Voordat we echter hiertoe overgaan zullen we eerst enige algemene begrippen behandelen.
Hiertoe gaan we uit van het stelsel differentiaalvergelijkingen uit hoofdstuk II, formule (1.1).
Laat $\vec{y}(t)$ de exacte oplossing van dit stelsel zijn en $\vec{y}^{*}(t)$ de numerieke berekende.
We introduceren nu de volgende begrippen:

(4.1) Onder de globale fout ε verstaan we de maximum-norm van het verschil tussen $\vec{y}(t)$ en $\vec{y}^{*}(t)$,

(4.2) onder het aantal functieevaluaties het getal, dat aangeeft hoevaak de rechterleden van dit stelsel berekend zijn.

In het vervolg zullen we nu van elk van de te behandelen voorbeelden het aantal functieevaluaties en de globale fout die daarbij hoort in een tabel en in een grafiek geven.
Uit de grafieken kan men dan direkt aflezen welke integratieprocedure voor een bepaald beginwaardeprobleem de voorkeur verdient boven de andere indien men een bepaalde nauwkeurigheid van de oplossing eist.

(4.3) Opmerking

In de grafieken hebben de merktekens betrekking op de volgende procedures:

\+ : RUNGE 2n,
* : RUNGE 3n,
Y : RUNGE 4n,
X : DIFFSYS.

Voorbeeld 1

Beschouw de differentiaalvergelijking:

$$(4.4)\qquad \begin{cases} \dfrac{dy}{dt} = -y,\ y = y(t), \\ y(0) = 1. \end{cases}$$

De exacte oplossing van (4.4) is: $y = e^{-t}$.

We losten (4.4) op voor t = 10. Als startwaarde voor t kozen we t = 0.

Uit §1 van dit hoofdstuk blijkt dat de beginwaarde voor de stapgrootte voor de vier besproken procedures gelijk is aan 10. We kozen deze differentiaalvergelijking dan ook om aan te tonen dat DIFFSYS onmiddellijk, d.w.z. zonder de stapgrootte te wijzigen, de exacte oplossing moet geven indien we modificatie 2 gebruiken (zie §3 en §5 van het vorige hoofdstuk). Uit tabel I blijkt inderdaad dat dit het geval is, want het minimum aantal functieevaluaties voor DIFFSYS is gelijk aan drie (zie ook §3 van dit hoofdstuk).

TABEL I

procedure	globale fout		aantal functieevaluaties
RUNGE 2n	2.82	10^{-2}	26
	8.11	10^{-3}	120
	3.70	10^{-4}	1066
	1.98	10^{-5}	10540
RUNGE 3n	4.93	10^{-2}	33
	2.04	10^{-3}	69
	1.57	10^{-4}	115
	1.97	10^{-5}	291
	7.23	10^{-6}	843
RUNGE 4n	1.97	10^{-2}	40
	2.98	10^{-3}	40
	3.02	10^{-4}	95
	1.70	10^{-5}	150
	7.66	10^{-6}	265
DIFFSYS	2.96	10^{-13}	3

Een grafische voorstelling van deze waarden kan men in fig. 3 vinden.
In fig. 3 zijn echter de waarden van DIFFSYS niet getekend, daar uit tabel I blijkt dat deze niet te vergelijken zijn met die van de andere procedures.

(4.5) Opmerking

Voor alle grafieken geldt dat op de horizontale as de $^{10}\log$ van de globale fout is uitgezet.

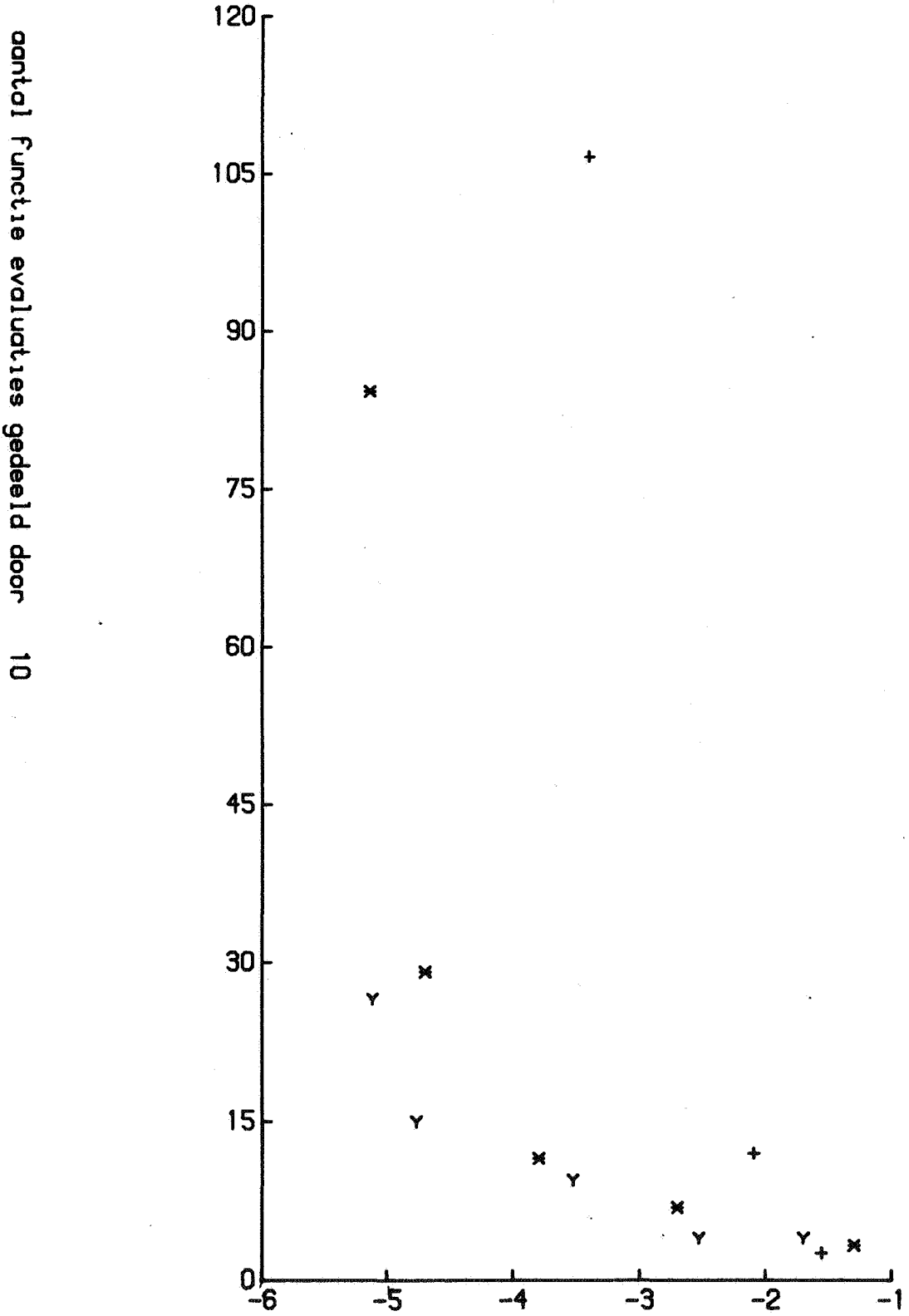
aantal functie evaluaties gedaeld door 10
120
105
90
75
60
45
30
15
0
-6
-5
-4
-3
-2
-1
log eps globaal

fig. 3

Voorbeeld 2

Als tweede voorbeeld behandelen we de differentiaalvergelijking van van der Pol voor mu = 10:

$$y'' - 10(1-y^2)\, y' + y = 0,\ y = y(t).$$

Deze differentiaalvergelijking werd als volgt als stelsel geschreven:

$$\frac{dy_1}{dt} = y_2,$$

$$\frac{dy_2}{dt} = 10(1-y_1^2)\, y_2 - y_1.$$

Als startwaarden kozen we:

$$y_1(0) = 2,$$

$$y_2(0) = 0,$$

en de integratie werd uitgevoerd tot t = 18.86305053.
In dit punt geldt nl.(zie [1]):

$$y_1 \simeq 2.01428536,$$

$$y_2 \simeq 0.$$

We zullen nu zowel de tabel als de grafiek geven voor y_1 en voor y_2.
Bij y_1 behoort tabel II en fig. 4, bij y_2 tabel III en fig. 5.

TABEL II

procedure	globale fout	aantal functieevaluaties
RUNGE 2n	$2.51\ 10^{-2}$	7010
	$1.84\ 10^{-4}$	9016
	$7.88\ 10^{-5}$	10980
	$4.08\ 10^{-5}$	13892
	$8.50\ 10^{-6}$	20686
	$1.88\ 10^{-6}$	100498
RUNGE 3n	$5.13\ 10^{-5}$	1578
	$1.43\ 10^{-5}$	1902
	$6.37\ 10^{-6}$	2272
	$3.70\ 10^{-6}$	2709
	$1.39\ 10^{-6}$	3471
	$4.72\ 10^{-7}$	4568
	$4.02\ 10^{-8}$	9354
RUNGE 4n	$1.15\ 10^{-5}$	1350
	$6.55\ 10^{-6}$	1500
	$4.54\ 10^{-6}$	1920
	$2.54\ 10^{-6}$	2210
	$3.00\ 10^{-7}$	3155
	$1.40\ 10^{-7}$	3755
	$2.22\ 10^{-8}$	5975
DIFFSYS	$3.83\ 10^{-5}$	2219
	$6.49\ 10^{-6}$	4971
	$2.19\ 10^{-4}$	1099
	$1.39\ 10^{-6}$	11403
	$3.55\ 10^{-7}$	24471

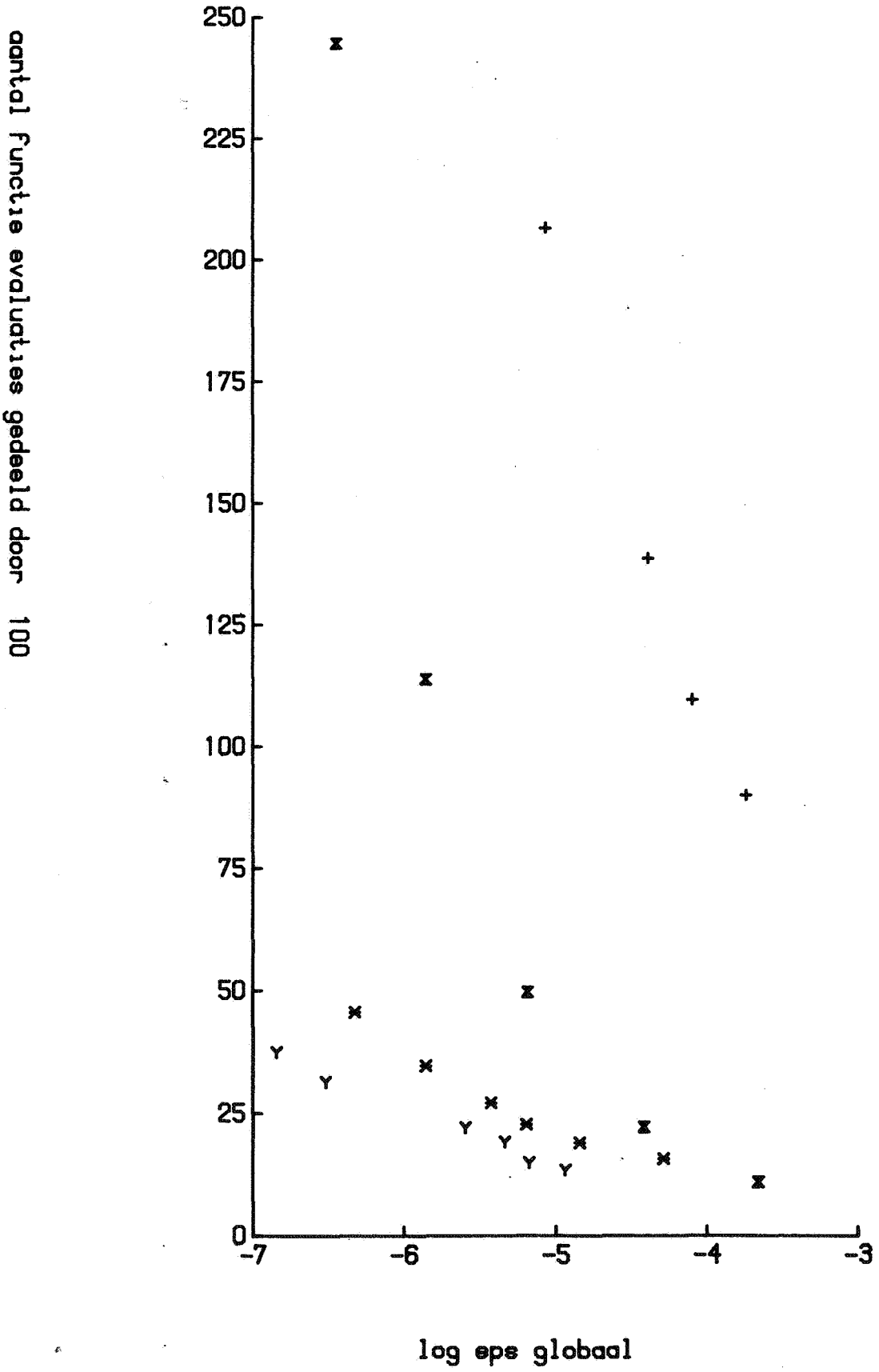
aantal functie evaluaties gedeeld door 100
250
225
200
175
150
125
100
75
50
25
0
-7
-6
-5
-4
-3
log eps globaal

fig. 4

TABEL III

procedure	globale fout	aantal functieevaluaties
RUNGE 2n	$5.61\ 10^{-2}$	2760
	$2.90\ 10^{-2}$	9016
	$2.07\ 10^{-2}$	10980
	$1.77\ 10^{-2}$	13892
	$8.15\ 10^{-3}$	20686
	$2.83\ 10^{-4}$	100498
RUNGE 3n	$1.47\ 10^{-2}$	1578
	$5.60\ 10^{-3}$	1902
	$3.11\ 10^{-3}$	2272
	$2.36\ 10^{-3}$	2709
	$1.33\ 10^{-3}$	3471
	$6.71\ 10^{-4}$	4568
	$1.22\ 10^{-4}$	9354
RUNGE 4n	$1.05\ 10^{-3}$	1500
	$7.80\ 10^{-4}$	1605
	$5.18\ 10^{-4}$	1700
	$2.21\ 10^{-4}$	1920
	$8.28\ 10^{-5}$	2210
	$6.32\ 10^{-3}$	1350
DIFFSYS	$2.56\ 10^{-2}$	1099
	$6.03\ 10^{-3}$	2219
	$2.01\ 10^{-3}$	4971
	$4.09\ 10^{-4}$	11403
	$1.26\ 10^{-4}$	24471

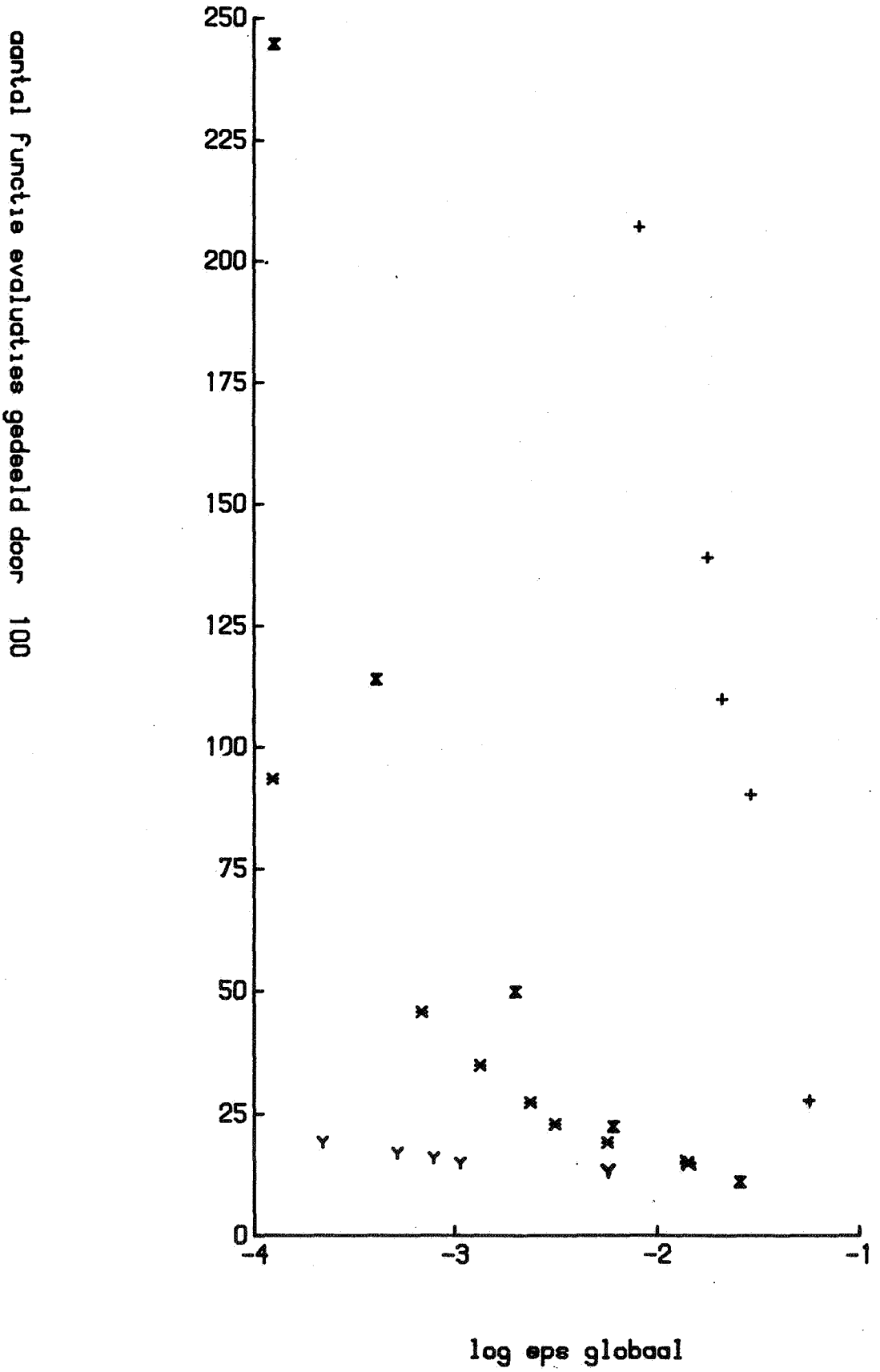
aantal functie evaluaties gedaan door 100
250
225
200
175
150
125
100
75
50
25
0
-4
-3
-2
-1
log eps globaal

fig. 5

Opmerking

Uit de figuren 4 en 5 kunnen we de conclusie trekken dat de Fowler-Warten methode niet te prefereren is boven RUNGE 3n en RUNGE 4n. De oorzaak hiervan is dat de eigenwaarden van de differentiaalvergelijking van van der Pol, die gelijk zijn aan 1 en $2m\ y_1y_2$, niet altijd in absolute waarde sterk in grootte verschillen (zie ook [1] blz. 94).

Voorbeeld 3

Als derde voorbeeld kozen we het stelsel differentiaalvergelijkingen:

$$(4.6)\qquad \begin{cases} \dfrac{dy_1}{dt} = -2000\ y_1 + 1000\ y_2 + 1,\ y_1 = y_1(t), \\ \dfrac{dy_2}{dt} = y_1 - y_2,\ y_2 = y_2(t), \\ y_1(0) = y_2(0) = 0. \end{cases}$$

De exacte oplossing van (4.6) is:

$$y_1 = 0.00100025 + 1.25\ 10^{-8}\ e^{-0.499875t} + 1.9995\ e^{-2000.500125t},$$

$$y_2 = 0.00100025 + 2.5\ 10^{-7}\ e^{-0.499875t} - 10^{-3}\ e^{-2000.500125t}.$$

Formule (4.6) kunnen we ook schrijven als

$$\frac{d\vec{y}}{dt} = A\vec{y} + \vec{c},$$

$$\vec{y}(0) = \underline{0},$$

hierin is:

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix},$$

$$\vec{e} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

$$A = \begin{pmatrix} -2000 & 1000 \\ 1 & -1 \end{pmatrix}.$$

De eigenwaarden van A zijn zoals uit de exacte oplossing van (4.6) blijkt ongeveer gelijk aan - 0.5 en - 2000.5.
Uit §1 van het vorige hoofdstuk volgt direkt dat de methode van Fowler-Warten er op gericht is om beginwaarde-problemen van dit type op te lossen.
Dat deze methode i.h.a. inderdaad de voorkeur verdient boven de Runge-Kutta methodes zal uit de gevonden resultaten blijken.
We losten (4.6) op voor t = 8. Uit §1 van dit hoofdstuk blijkt dat de beginwaarde van de stapgrootte voor de vier besproken procedures dan gelijk is aan 8.
Uit de stabiliteitseisen voor bijvoorbeeld de vierde orde Runge-Kutta methode blijkt dan dat de maximale stapgrootte gelijk gemaakt wordt aan $2.78 / 2000.5 \simeq 1.35\ 10^{-3}$ (zie hoofdstuk I, §4).
Hieruit volgt, dat er minstens 6000 stappen of 30000 functieevaluaties (zie §2 van dit hoofdstuk) nodig zijn om (4.6) in het punt t = 8 op te lossen, indien we t = 0 als beginpunt kiezen.
Bovendien weten we dat de exacte oplossing zich ongeveer gedraagt als $e^{-\frac{1}{2}t}$.
Indien we echter de grafiek van $e^{-\frac{1}{2}t}$ tekenen blijkt dat deze veel sterker daalt indien $t \in [0,1]$, dan voor bijvoorbeeld $t \in [7,8]$.
Hieruit volgt dat er meer functieevaluaties nodig zijn om de oplossing van (4.7) in het punt t = 1 te berekenen, als we t = 0 als beginpunt kiezen, dan in het punt t = 8, indien we uitgaan van t = 7 als beginwaarde.
Op grond hiervan gaan we, daar we de oplossing y_1 van (4.6) in het punt t = 8 willen berekenen niet uit van t = 0, doch bijvoorbeeld van t = 7.9 (zie tabel V en fig. 7).
Als beginvoorwaarden van y_1 en y_2 in t = 7.9 kozen we:

$$y_1 = y_2 = 0.0009.$$

Bovendien losten we (4.6) nog op in het punt t = 0.1, terwijl we t = 0 als startwaarde voor t kozen. In de tabellen IV en V en de figuren 6 en 7 zijn alleen de waarden voor y_1 gegeven.

Opmerking

In de twee voorafgaande voorbeelden gebruikten we deze efficiente wijze van oplossen niet, daar voorbeeld 1 gebruikt werd om aan te tonen, dat modificatie 2 in de praktijk zeer goed werkt en we van het stelsel differentiaalvergelijkingen uit voorbeeld 2 de exacte oplossing niet kenden.

TABEL IV

procedure	globale fout	aantal functieevaluaties
RUNGE 2n	$2.58\ 10^{-5}$	322
	$9.60\ 10^{-5}$	324
	$1.19\ 10^{-5}$	344
	$1.70\ 10^{-6}$	420
	$1.50\ 10^{-6}$	536
	$2.31\ 10^{-7}$	1372
RUNGE 3n	$1.84\ 10^{-5}$	386
	$1.48\ 10^{-5}$	401
	$6.15\ 10^{-7}$	405
	$1.10\ 10^{-6}$	410
	$9.85\ 10^{-5}$	441
	$2.00\ 10^{-7}$	445
	$2.92\ 10^{-7}$	462
	$2.90\ 10^{-8}$	488
RUNGE 4n	$3.97\ 10^{-7}$	490
	$1.01\ 10^{-5}$	495
	$8.21\ 10^{-7}$	505
	$6.70\ 10^{-6}$	525
	$8.94\ 10^{-5}$	540
	$7.09\ 10^{-8}$	550
DIFFSYS	$1.50\ 10^{-6}$	53
	$2.81\ 10^{-6}$	69
	$2.78\ 10^{-7}$	97
	$1.15\ 10^{-7}$	131
	$1.22\ 10^{-8}$	1181
	$1.31\ 10^{-8}$	543
	$3.86\ 10^{-8}$	217

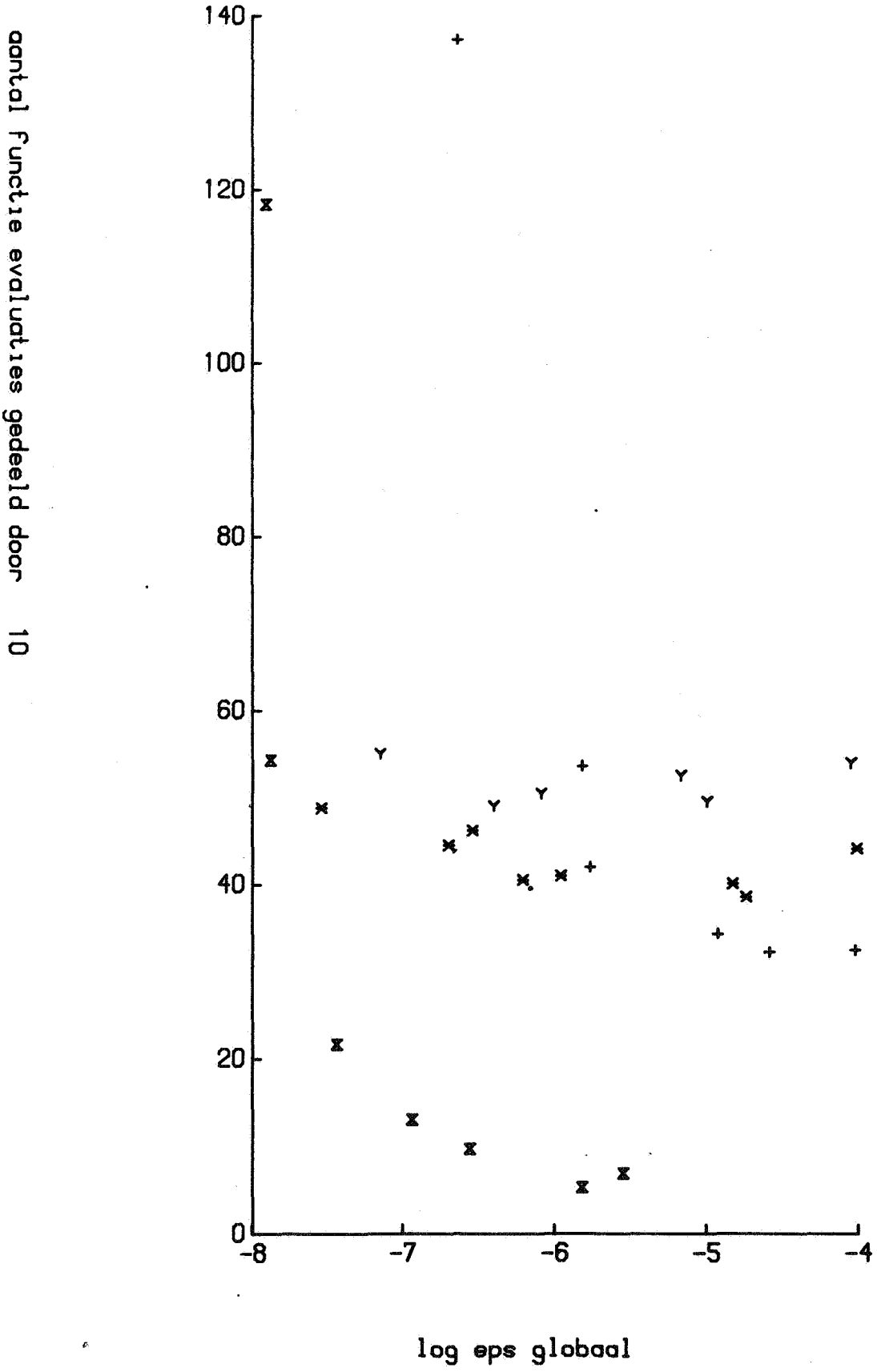
aantal functie evaluaties gedeeld door 10
140
120
100
80
60
40
20
0
-8
-7
-6
-5
-4
log eps globaal

fig. 6

TABEL V

procedure	globale fout		aantal functieevaluaties
RUNGE 2n	1.00	10^{-3}	268
	1.69	10^{-4}	320
	4.23	10^{-5}	322
	9.21	10^{-5}	322
	4.83	10^{-5}	324
	4.29	10^{-4}	326
	3.98	10^{-5}	344
	3.93	10^{-5}	422
	3.89	10^{-5}	536
	3.87	10^{-5}	1372
RUNGE 3n	5.60	10^{-4}	379
	6.10	10^{-5}	394
	4.83	10^{-5}	407
	6.09	10^{-3}	408
	1.28	10^{-4}	409
	3.79	10^{-5}	413
	2.75	10^{-5}	418
	8.73	10^{-4}	421
	5.35	10^{-4}	452
	3.98	10^{-5}	452
	3.86	10^{-5}	506
RUNGE 4n	1.50	10^{-3}	485
	4.16	10^{-3}	485
	3.87	10^{-5}	550
	3.87	10^{-5}	610
DIFFSYS	3.89	10^{-5}	69
	3.87	10^{-5}	97
	3.87	10^{-5}	217
	3.87	10^{-5}	543

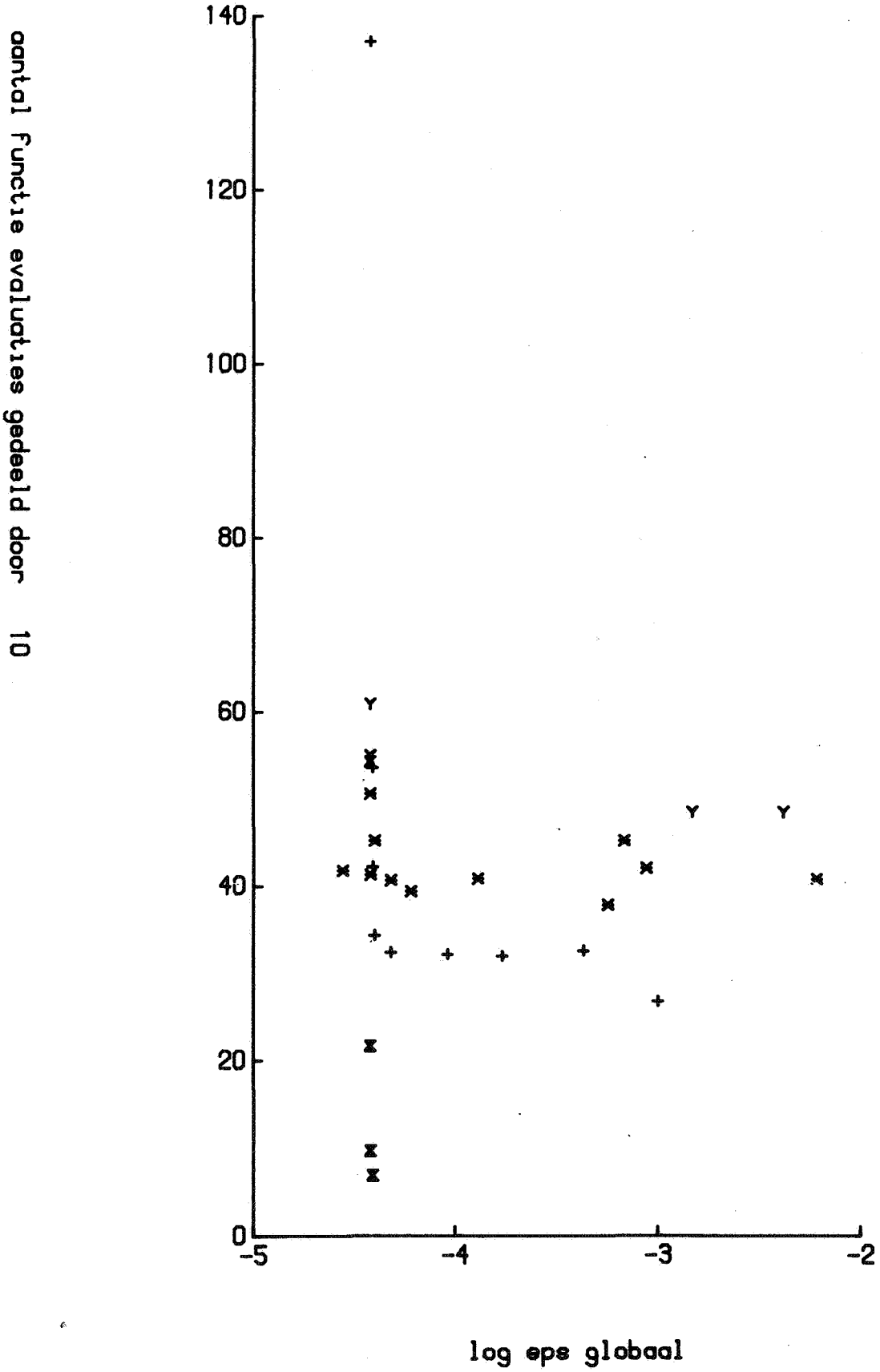
aantal functie evaluaties gedeeld door 10
140
120
100
80
60
40
20
0
-5
-4
-3
-2
log eps globaal

fig. 7

Literatuur

[1] J.A. Zonneveld — Automatic numerical integration. Mathematical Centre Tracts 8 (1964).

[2] T.J. Dekker — Cursus Wetenschappelijk Rekenen A: Numerieke Wiskunde deel II. Mathematisch Centrum (1966).

[3] A. van Wijngaarden — Cursus Wetenschappelijk Rekenen B: Proces Analyse. Mathematisch Centrum (1965).

[4] S. Wilf — A stability criterion for numerical integration. Journal of the ACM 6, no. 3. blz. 363-365 (July 1959).

[5] A.I. Abdel Karim — The stability of the fourth order Runge-Kutta method for the solution of systems of differential equations. Communications of the ACM 9, no. 2, blz. 113-116 (February 1966).

[6] M.E. Fowler en R.M. Warten — A numerical integration technique for ordinary differential equations with widely separated eigenvalues. I.B.M. Journal of Research and Development (September 1967).

[7] W. Liniger — Zur Stabilität der numerischen Integrationsmethoden für Differentialgleichungen. Thesis, University of Lausanne (1967).

[8] R.J. de Vogelaere — Werkbespreking Rekenafdeling No. 46. Mathematisch Centrum (1970).

[9] F.E.J. Kruseman-Aretz — Het MC-ALGOL 60 systeem voor de EL X8. Mathematisch Centrum MR 81.

[10] P.J. van der Houwen — Finite difference methods for solving partial differential equations. Mathematical Centre Tracts 20 (1968).

[11] W.E. Baanstra — Werkbespreking Rekenafdeling No. 31. Mathematisch Centrum (1968).

[12] J. Certaine — The solution of ordinary differential equations with large time constants. Ralston and Wilf: Mathematical methods for digital computers, blz. 128-132 (1960).

[13] C.F. Curtiss en J.O. Hirschfelder — Integration of stiff equations. Proc. Natl. Acad. Sci. USA 38, blz. 235-243 (1952).

[14] G. Emanuel — Numerical analysis of stiff equations. Report No. TDR-269(4230-20)-3, Aerospace corp. El Segundo, California (1964).

[15] C.W. Gear — The automatic integration of stiff ordinary differential equations. Information Processing (IFIP, 1968).

[16] P. Henrici — Discrete variable methods in ordinary differential equations. Wiley (1962).

[17] F.B. Hildebrand — Introduction to numerical analysis. McGraw-Hill (1956).

[18] A. Huťa — Une amélioration de la méthode Runge-Kutta-Nyström pour la résolution numérique des équations differentielles du premier ordre. Act. Fac. Rerum Nat. Univ. Comenianae Math. 1, blz. 201-224 (1956).

[19] A. Huťa — Contribution à la formule de sixième ordre dans la méthode de Runge-Kutta-Nyström. Act. Fac. Rerum Nat. Univ. Comenianae Math. 2, blz. 21-23 (1957).

[20] H.B. Lee — Matrix filtering as an aid to numerical integration. Proc. of the IEEE, volume 55, no. 11, blz. 1826-1831 (November 1967).

21 P. Naur (ed.) — Revised report on the algorithmic language ALGOL 60 (1962).