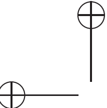
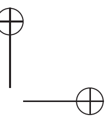



Module 27 Voorwaardelijke opdrachten en herhalingsopdrachten

Onderwerp Voorwaardelijke opdrachten en booleans, herhalingsopdrachten.
Voorkennis Module 3, 8.
Expressies `if, end, then, else, elif, piecewise, true, false, FAIL, is, testeq, and, or, not, evalb, testeq, for, from, to, by, while, in, do`
Zie ook Module 28, 29, 30

In deze module geven we aan de hand van voorbeelden een kort overzicht van het gebruik van voorwaardelijke opdrachten en herhalingsopdrachten.

27.1 Voorwaardelijke opdrachten

Een voorwaardelijke opdracht dient om, afhankelijk van het al of niet voldaan zijn van een voorwaarde, verschillende acties te kunnen laten plaatsvinden. Bijvoorbeeld,

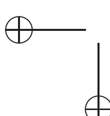
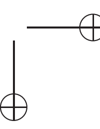
```
if (x>0) then x^2 else -x end if;
```

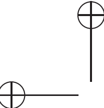
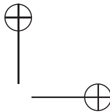
betekent: "*Als* $x > 0$ *dan*: x^2, *anders* (dus als x *niet* > 0): $-x$" en levert dus x^2 op als $x > 0$ en $-x$ als $x \le 0$.
Een voorwaardelijke opdracht begint met `if` en wordt afgesloten met `end if`. In het algemeen is de vorm

`if, then` — `if` *(voorwaarde)* `then`
(lijst Maple-statements)
`else` — `else`
(lijst Maple-statements)
`end if` — `end if;`

Hierbij mag het `else`-gedeelte[61] worden weggelaten: in dat geval doet Maple niets als aan de voorwaarde achter `if` niet is voldaan. Tussen `then` en `else`, respectievelijk tussen `else` en `end if` kunnen méér statements komen. Deze worden onderling gescheiden door puntkomma's of dubbele punten. Achter het laatste statement voor `else` of `end if` hoeft geen dubbele punt of puntkomma. In het bijzonder

[61] Daarmee bedoelen we het woord `else` en de *(lijst Maple-statements)* daarachter.

kunnen in de lijst Maple-statements na `then` of een eventuele `else` weer voorwaardelijke opdrachten voorkomen.
Een voorwaardelijke opdracht zullen we ook wel aangeven met de term *if-statement*.

> **!** We spreken van '*een* voorwaardelijke opdracht' of '*een* if-statement'. Daarmee brengen we tot uitdrukking dat alles wat tussen `if` en `end if` staat door Maple als één opdracht wordt beschouwd. Het begint pas met het verwerken ervan als na een `if` een `end if` is gevonden.
> Vaak zal voor het invoeren van zo'n 'samengestelde opdracht' meer dan één invoerregel nodig zijn. Zorg er dan voor dat de invoerregels samen in één 'execution group' komen (zie §2.2).

Een speciale vorm van voorwaardelijke opdrachten zijn we al tegen gekomen in §3.4 bij het definiëen van functies met `piecewise`. In deze module zullen we steeds voorbeelden geven die óók met `piecewise` kunnen. Het `if`-statement biedt veel meer mogelijkheden, die in Module 28 uitgebreid gebruikt worden.

Voorbeeldopgave

Formuleer in Maple de functie met het voorschrift

$$x \mapsto \begin{cases} x^2 & \text{als } x > 0 \\ -x & \text{als } -1 < x \leq 0 \\ x^3 & \text{als } x \leq -1. \end{cases}$$

Voorbeeldsessie

```
>  f := x -> if x>0 then x^2
             else if x<=-1 then x^3
                  else -x
                  end if
             end if:
```

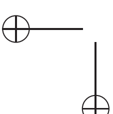
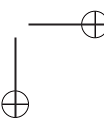
Ook in te voeren als:

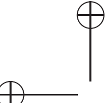
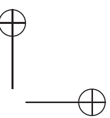
```
>  f := x -> if x>0 then x^2
             elif x<=-1 then x^3
             else -x
             end if:
```

```
>  f(4);
```

16

```
>  f(y);
```


```
Error, (in f) cannot determine if this expression is true or
false: 0 < y
```

Een in maple ingebouwde mogelijkheid voor dit soort functies:

```
>  g := x -> piecewise( x>0, x^2, x<=-1, x^3, -x );
```

$$g := x \to \text{piecewise}(0 < x,\, x^2,\, x \leq -1,\, x^3,\, -x)$$

```
>  g(y);
```

$$\begin{cases} y^2 & 0 < y \\ y^3 & y \leq -1 \\ -y & \textit{otherwise} \end{cases}$$

Toelichting

De gevraagde functie is in deze sessie op drie verschillende manieren gemaakt. De eerste twee leveren in feite precies hetzelfde op, alleen is in de tweede definitie de formulering iets korter en daardoor wellicht overzichtelijker. De constructie met `if`... `then`... `elif`... `then`... `elif`... `then`... ... `else`... `end if` is vooral nuttig als een betrekkelijk groot aantal 'gevallen' bekeken moet worden. Wat na de laatste `else` staat is dan wat in de 'overige gevallen' moet gebeuren.

`elif`

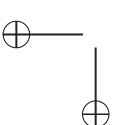
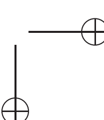
In deze sessie levert `f(4);` keurig het antwoord `16` op, maar `f(y)` resulteert in een foutmelding omdat `y` geen waarde heeft, zie §27.2. Dit probleem wordt ondervangen door gebruik te maken van de procedure `piecewise`. De volgorde van de argumenten is precies als in de if-then-elif...-else-end if-constructie; `g(y);` geeft nu een net overzicht van de wijze waarop `g` is gedefinieerd zolang `y` geen waarde heeft. ◇

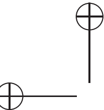
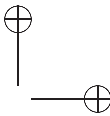
`piecewise`

27.2 Boolean-expressies

De voorwaarde in de `if`- of `elif`-clausule noemen we een *boolean*-expressie, dat is een uitdrukking die waar is of onwaar. Voorbeelden: $x > 0$, $n = 3$, $sin(x) = 0$, $x \in \mathbb{R}$ (in Maple: `type(x,realcons)`), `has(p,a)` enzovoort. Zo'n expressie evalueert tot `true`, `false` of `FAIL`. Het laatste geval (`FAIL`) treedt op als de (on)waarheid van zo'n expressie niet kan worden vastgesteld. Bijvoorbeeld, als `x` nog geen waarde heeft gekregen in een sessie, dan is `x<0` nog onbepaald. Maple reageert met de foutmelding '*cannot determine if this expression is true or false*' als zo'n onbepaalde boolean in de `if`-clausule staat.

`true, false`
`FAIL`

`evalb` Evaluatie van een booleaanse expressie kan zo nodig worden afgedwongen met `evalb`. De procedure `evalb` voert *geen* 'wiskundige berekeningen' uit. Als de expressie bijvoorbeeld een vergelijking is, dan wordt botweg alléén gekeken of het linkerlid letterlijk dezelfde formule is als het rechterlid. Zo niet, dan geeft het de expressie gewoon terug.

`is` Vaak zal het daarom nodig zijn om de expressie eerst met `simplify` te vereenvoudigen. Als het dan nog niet lukt, dan kan `is`[62] worden gebruikt, inplaats van `evalb`.

`testeq` In het geval de expressie een gelijkteken bevat, bijvoorbeeld als we willen nagaan of een zekere x-waarde aan een vergelijking voldoet, is `testeq` een paardenmiddel.

Voorbeeldsessie

```
> restart;
> if x>0 then x^2 else -x end if;
```

```
Error, cannot determine if this expression is true or false:
 0 < x
```

x heeft nog geen waarde. We geven x een waarde:

```
> x := 4:
> if x>0 then x^2 else -x end if;
```

$$16$$

De foutmelding kan echter ook op onverwachte plaatsen opduiken:

```
> if Pi < 4 then ja else nee end if;
```

```
Error, cannot determine if this expression is true or false:
 Pi < 4
```

```
> if evalf(Pi) < 4 then ja else nee end if;
```

$$ja$$

Alternatief

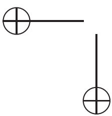
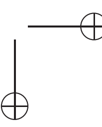
```
> if is(Pi<4) then ja else nee end if;
```

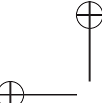
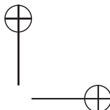
$$ja$$

Toelichting

In een `if...then`-constructie wordt automatisch `evalb` gedaan. Omdat daarin niets wordt 'uitgerekend', kan het de waarden van π en 4 niet met elkaar vergelijken! Dit geldt trouwens voor veel meer 'exacte' getallen: zelfs $\sqrt{2} < 2$ gaat óók niet goed ($1/3 < 1$ trouwens wél).

[62] Als u met `?is` nadere informatie opvraagt, dan merkt u dat dit commando hier eigenlijk helemaal niet voor bedoeld is. Het werkt echter heel vaak beter dan `evalb`.

Bij het testen van een ongelijkheid is het daarom een goede gewoonte om er standaard `evalf` om te zetten. ◇

Het gebruik van booleaanse expressies beperkt zich niet tot eenvoudige statements als `x>0`. Booleaanse expressies kunnen worden gecombineerd met de logische operatoren `and`, `or` en `not`. In de volgende sessie definiëren we de functie met het voorschrift

`and, or, not`

$$x \mapsto \begin{cases} x & \text{als } x \geq 0 \text{ of } x \leq -5 \\ x^2 & \text{als } x > -5 \text{ en } x < -2 \\ x^3 & \text{als } x \geq -2 \text{ en } x < 0. \end{cases}$$

Op zo'n functie kunnen we dan de `D`-operator laten werken om de afgeleide te berekenen. Maple geeft echter geen uitsluitsel of deze afgeleide daadwerkelijk bestaat!

Voorbeeldsessie

```
> f := x -> if (x <= -5 or x >= 0) then x
             elif (x > -5 and x < -2) then x^2
             else x^3
             end if;
```

$$f := x \rightarrow$$
$$\mathbf{if}\, x \leq -5 \;\mathbf{or}\; 0 \leq x \;\mathbf{then}\, x$$
$$\mathbf{elif} -5 < x \;\mathbf{and}\; x < -2 \;\mathbf{then}\, x^2$$
$$\mathbf{else}\, x^3$$
$$\mathbf{end\ if}$$

```
> D(f);
```

$$x \rightarrow$$
$$\mathbf{if}\, x \leq -5 \;\mathbf{or}\; 0 \leq x \;\mathbf{then}\, 1$$
$$\mathbf{elif} -5 < x \;\mathbf{and}\; x < -2 \;\mathbf{then}\, 2\,x$$
$$\mathbf{else}\, 3\,x^2$$
$$\mathbf{end\ if}$$

```
> D(f)(2);
```

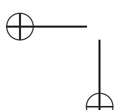
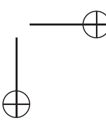
$$1$$

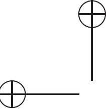
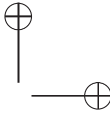
Toelichting

De afgeleide `D(f)` wordt keurig als een functie bepaald. ◇

27.3 Herhalingsopdrachten

Herhalingsopdrachten dienen om op een gemakkelijke manier een rij Maple-commando's een aantal keren achter elkaar uit te voeren. We

zullen drie vormen van herhalingsopdrachten behandelen, elk aan de hand van een voorbeeld.

For-loop, versie from - to: We geven eerst de algemene vorm (syntaxis) van deze variant van de herhalingsopdracht:

`for, from, to`
`do`
`end do`

initialisatie
`for` *(teller)* `from` *(beginwaarde)* `to` *(eindwaarde)*
`do`
(rij Maple-statements)
`end do;`

Hiermee wordt de rij Maple-statements tussen de `do` en `end do` achtereenvolgens uitgevoerd voor alle (gehele) waarden van de variabele *teller* die liggen tussen de beginwaarde en de eindwaarde.

Voorbeeldopgave

Gegeven de lijst `L = [3,1,0,5,7,8,6,2]`, met 8 elementen.
Bepaal de som van deze elementen.

Voorbeeldsessie

```
> L := [3,1,0,5,7,8,6,2]:
> som := 0; # Initialisatie
```

$$som := 0$$

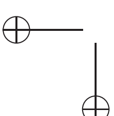
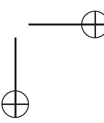
```
> for i from 1 to 8 do som := som + L[i]:  end do;
```

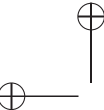
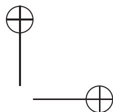
$$som := 3$$
$$som := 4$$
$$som := 4$$
$$som := 9$$
$$som := 16$$
$$som := 24$$
$$som := 30$$
$$som := 32$$

Als de initialisatie wordt vergeten, levert het opnieuw aktiveren van het statement:

```
> for i from 1 to 8 do som := som + L[i] end do;
```

$$som := 35$$
$$som := 36$$
$$som := 36$$
$$som := 41$$
$$som := 48$$
$$som := 56$$
$$som := 62$$

$$som := 64$$

De complete herhalingsopdracht is dus (nu met dubbele punt achter '`end do`'):

```
> som := 0:
  for i from 1 to 8 do som := som + L[i] end do:
  som;
```

$$32$$

Toelichting

De teller heet hier `i` en loopt van 1 tot en met 8. De *rij Maple-statements* bevat maar één opdracht, namelijk `som := som + L[i]`. Aan de voorbeeldsessie is ook te zien waarom de een of andere vorm van *initialisatie* in het algemeen een wezenlijk onderdeel van de herhalingsopdracht vormt. Zie ook de laatste voorbeeldsessie in §25.1 voor wat er gebeurt als `som` vóór het `for`-statement helemaal geen waarde heeft.
Overigens doet de herhalingsopdracht in dit voorbeeld precies hetzelfde als

```
som := add( L[i], i=1..8 );
```

zie Module 6. ◇

Als de teller bij 1 begint, dan mag `from 1` eventueel worden weggelaten. In de voorbeeldsessie hadden we dus ook kunnen doen:

```
for i to 8 do som := som + L[i] end do:
```

De waarde van de teller wordt steeds automatisch met 1 verhoogd. Dat hoeft niet. Men zou ook bijvoorbeeld alleen de elementen op de oneven plaatsen in `L` kunnen optellen door `by` te gebruiken:

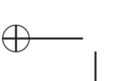
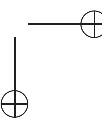
`by`

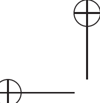
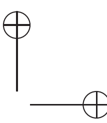
```
for i from 1 by 2 to 8 do...
```

Achter `by` mag ook een negatief getal staan.

Let ook op het gebruik van `:` en `;` Voor `end do` mag het worden weggelaten. Als de hele herhalingsopdracht wordt afgesloten met `end do;` (dus met puntkomma) heeft dat tot gevolg dat elke keer dat de statements tussen `do` en `end do` worden herhaald, de resultaten ervan op het scherm worden getoond. Meestal zal men die tussenresultaten alleen willen zien als vermoed kan worden dat er iets niet in de haak is. Maak er maar een gewoonte van elke herhalingsopdracht met een dubbele punt af te sluiten.

Een toepassing van het opbouwen van een expressierij (zie §8.1) hebben we als we, voor dezelfde lijst als die van de opgave, een lijst `S` willen maken die de partiële sommen (dus eigenlijk de tussenresultaten van de vorige sessie) bevat.

Voorbeeldsessie

```
> L := [3,1,0,5,7,8,6,2]:
> som := 0:  somrij := NULL;   # Initialisatie
```

$$somrij :=$$

```
> for i from 1 to nops(L) do
     som := som + L[i]:
     somrij := somrij, som
   end do;
```

$$som := 3$$
$$somrij := 3$$
$$som := 4$$
$$somrij := 3, 4$$
$$som := 4$$
$$somrij := 3, 4, 4$$
$$som := 9$$
$$somrij := 3, 4, 4, 9$$
$$som := 16$$
$$somrij := 3, 4, 4, 9, 16$$
$$som := 24$$
$$somrij := 3, 4, 4, 9, 16, 24$$
$$som := 30$$
$$somrij := 3, 4, 4, 9, 16, 24, 30$$
$$som := 32$$
$$somrij := 3, 4, 4, 9, 16, 24, 30, 32$$

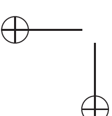
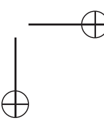
```
> S := [somrij];
```

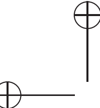
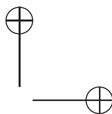
$$S := [3, 4, 4, 9, 16, 24, 30, 32]$$

Toelichting

Let op dat ook de expressierij `somrij` geïnitialiseerd moet worden, in dit geval als een lege rij.
Bij elke herhaling van de statements in de do-loop wordt er nu een getal aan `somrij` toegevoegd. Ten slotte maken we er de vereiste lijst van door er vierkante haken om te zetten. ◇

For-loop, versie 'x in': Deze versie is bijvoorbeeld handig als de 'teller' niet steeds met dezelfde waarde moet worden vehoogd.
De algemene vorm van deze herhalingsopdracht is:

`in`

(initialisatie)
`for` *(variabele)* `in` *(expressie)*
`do`
    (rij Maple-statements)
`end do;`

De *variabele* krijgt achtereenvolgens alle waarden die in *expressie* (meestal een lijst of verzameling) voorkomen. Voor al deze waarden wordt dan de rij Maple-statements tussen de `do` en de `end do` uitgevoerd.
Met deze versie zou een andere oplossing voor de opgave van blz. 418 zijn:

Voorbeeldsessie

```
> L := [3,1,0,5,7,8,6,2]:
> som := 0:
> for x in L do som := som + x end do;
```

$$som := 3$$
$$som := 4$$
$$som := 4$$
$$som := 9$$
$$som := 16$$
$$som := 24$$
$$som := 30$$
$$som := 32$$

Alternatief:

```
> Som := convert( L, `+` );
```

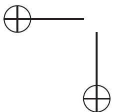
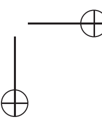
$$Som := 32$$

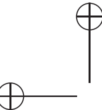
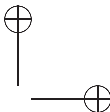
Toelichting

In dit voorbeeld neemt de variabele `x` achtereenvolgens alle waarden aan die in de lijst `L` voorkomen.
Het laatste statement is meer een illustratie bij §26.2: het converteert de expressie `L`, type `list` naar een expressie `S`, type `+`, met dezelfde operanden als `L`. ◇

Als `L` geen lijst of verzameling is, dan wordt de expressie `for x in L` opgevat als `for x in [op(L)]`, dat wil zeggen dat `x` achtereenvolgens de waarden van alle operanden van `L` krijgt. Maar het is moeilijk om hier een zinvolle toepassing voor te vinden.

De derde vorm van de herhalingsopdracht is:

While-loop: Deze herhalingsopdracht heeft de volgende algemene vorm:

while

> *(initialisatie)*
> `while` *(boolean)*
> `do`
> *(rij Maple-statements)*
> `end do;`

Maple voert hier de rij Maple-statements tussen `do` en `end do` net zo vaak uit tot de boolean na de `while` niet meer tot `true` evalueert. Deze vorm is vooral handig als er iets opgezocht moet worden, bijvoorbeeld in een lijst of verzameling. Zodra gevonden is waarnaar gezocht wordt kan de *boolean* op `false` worden gezet, en dan zijn we klaar.

Voorbeeldopgave

Gegeven de lijst `L = [3,1,4,5,7,8,6,2]`, met 8 elementen.
Bepaal het eerste element dat groter is dan 4.

Voorbeeldsessie

```
> L := [3,1,4,5,7,8,6,2]:
```

Eerste variant:

```
> gevonden := false:  i := 0:    x := 'x':    #initialisatie
> while not gevonden do
     i := i+1:
     if L[i] > 4 then x := L[i]:  gevonden := true end if:
  end do;
```

$$i := 1$$
$$i := 2$$
$$i := 3$$
$$i := 4$$

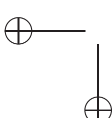
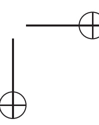
```
> `resultaat:  op plaats`, i, `staat`, x;
```

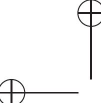
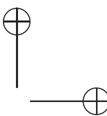
$$resultaat:\ op\ plaats,\ 4,\ staat,\ 5$$

Tweede variant:

```
> x := L[1]:  i := 1:    # initialisatie
> while x <= 4 do i := i+1:  x := L[i] end do;
```

$$i := 2$$
$$x := 1$$
$$i := 3$$
$$x := 4$$
$$i := 4$$
$$x := 5$$


```
> `resultaat:  op plaats`, i, `staat`, x;
```

$$\mathit{resultaat:\ op\ plaats},\ 4,\ \mathit{staat},\ 5$$

Derde variant:

```
> i := 1:  # initialisatie
> while L[i] <= 4 do i := i+1 end do;
```

$$i := 2$$
$$i := 3$$
$$i := 4$$

```
> `resultaat:  op plaats`, i, `staat`, L[i];
```

$$\mathit{resultaat:\ op\ plaats},\ 4,\ \mathit{staat},\ 5$$

Toelichting

We hebben drie, steeds kortere varianten gemaakt. De eerste sluit het best aan bij de beschrijving van de `while`-loop, en heeft (daarom) de meeste hulpvariabelen nodig.
Let goed op de gebruikte initialisaties die bij elk van de drie versies verschillen. ◇

Herhalingsopdrachten waarin een `do - end do`-clausule voorkomt, zullen we aangeven met de term *do-loop*. Herhalingsopdrachten mogen ook in een gecombineerde `for-in-while` vorm voorkomen. Raadpleeg hiervoor `?for`.

Geneste do-loops. Als laatste opmerking nog dit: do-loops mogen ook binnen de `do-end do` van andere do-loops voorkomen. Dit wordt het *nesten* van do-loops genoemd. Dit hebben we vaak nodig bij het 'vullen' van een matrix of een meerdimensionale Array. We geven daarvan een voorbeeld.

Voorbeeldopgave

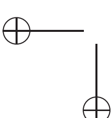
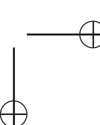
Gegeven een matrix A met natuurlijke getallen.
Maak een nieuwe matrix B, waarin $b_{ij} = e$ als de overeenkomstige a_{ij} een even getal is, en $b_{ij} = o$ als a_{ij} oneven is.

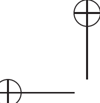
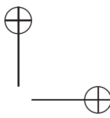
Voorbeeldsessie

```
> with(LinearAlgebra):
> A := RandomMatrix(3,5, generator=0..99);
```

$$A := \begin{bmatrix} 71 & 36 & 91 & 9 & 56 \\ 12 & 26 & 95 & 58 & 71 \\ 5 & 98 & 61 & 49 & 72 \end{bmatrix}$$

```
> n,m := Dimension(A);
```

$$n, m := 3, 5$$

```
> B := Matrix(n,m):    # initialisatie
> for i to n do
    for j to m do
      if type( A[i,j], even )
        then B[i,j] := e
        else B[i,j] := o
      end if
    end do
  end do:
> B;
```

$$\begin{bmatrix} o & e & o & o & e \\ e & e & o & e & o \\ o & e & o & o & e \end{bmatrix}$$

Alternatief zonder (expliciete) loop:

```
> map( x -> if type(x,even) then e else o end if, A );
```

$$\begin{bmatrix} o & e & o & o & e \\ e & e & o & e & o \\ o & e & o & o & e \end{bmatrix}$$

Toelichting

In dit voorbeeld is de initialisatie het maken van de (uit louter nullen bestaande) matrix B. In het for-statement wordt B gevuld.
Het alternatief is aardig voor degenen die er een sport van maken het gebruik van expliciete herhalingsopdrachten zoveel mogelijk te vermijden. ◇

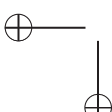
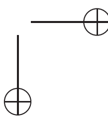
27.4 Iteratieve processen

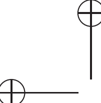
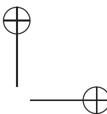
In veel situaties worden do-loops (of while-loops) met voorwaardelijke opdrachten gecombineerd. Typische voorbeelden zijn zogenaamde *iteratieve processen*.

Nulpuntsbepaling door intervalhalvering. Om een numerieke benadering van een nulpunt van een continue functie te benaderen kunnen we gebruikmaken van de *tussenwaardestelling* die in haar eenvoudigste vorm luidt:

> Als f continu is op het interval $[a, b]$, en als $f(a) > 0$ en $f(b) < 0$ of andersom, dan is er minstens één c tussen a en b met $f(c) = 0$.

Zo'n nulpunt c kan als volgt worden benaderd.

We beginnen met $x_l = a$ (linker eindpunt van het interval) en $x_r = b$ (rechter eindpunt). Er geldt nu dat het teken van $f(x_l)$ verschillend is van het teken van $f(x_r)$, dus $f(x_l)\,f(x_r) < 0$.
We berekenen nu $x_m = \frac{1}{2}(x_l + x_r)$, het midden van het interval $[x_l, x_r]$, en de bijbehorende functiewaarde $f(x_m)$.
Als $|f(x_m)| < \epsilon$ dan zijn we klaar. Zo niet, dan kijken we naar het teken van $f(x_m)$.
Als dat verschillend is van het teken van $f(x_l)$, dan vervangen we x_r door x_m en dan weten we dat het nulpunt c op het *nieuwe*, half zo lange interval $[x_l, x_r]$ moet liggen.
Als het teken van $f(x_m)$ gelijk is aan dat van x_l, dan moet het verschillend zijn van dat van x_r en kunnen we x_l vervangen door x_m.
Op het nieuwe interval berekenen we een nieuwe x_m enzovoort. De lengte van het interval waarop c moet liggen wordt steeds gehalveerd, dus we kunnen c zo nauwkeurig benaderen als we willen.

Voorbeeldopgave

Benader het nulpunt c van

$$f(x) = 9x^3 + 5x^2 + 8x + 6$$

op het interval $[-1, 1]$, zó dat $|f(c)| < 0.01$.

Voorbeeldsessie

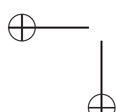
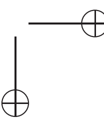
```
> f := x -> 9*x^3 + 5*x^2 + 8*x + 6;
```

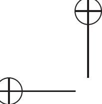
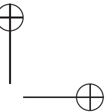
$$f := x \to 9\,x^3 + 5\,x^2 + 8\,x + 6$$

```
> xl := -1:  xr := 1:  xm := 0.5*(xr+xl):    # initialisatie
> eps := 0.01:
> while abs( f(xm) ) > eps do   #1
    if f(xm)*f(xl) < 0          #2
      then xr := xm             #3
      else xl := xm             #4
    end if:                    #5
    xm := 0.5*(xr+xl)           #6
  end do;                       #7
```

$$xm := -0.5$$
$$xm := -0.75$$
$$xm := -0.625$$
$$xm := -0.6875$$
$$xm := -0.65625$$
$$xm := -0.671875$$
$$xm := -0.6796875$$
$$xm := -0.68359375$$

Controle:

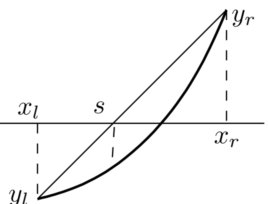

```
> f(xm);
```

$$-0.007240713$$

Toelichting

Na de initialisatie (vastleggen van de beginwaarden voor x_l, x_r en x_m) volgen we de bovenstaande redenering op de voet. De statements tussen `#1` en `#7` worden herhaald zolang $f(x_m)$ groter dan ϵ is. Elke keer wordt bekeken of het nulpunt in de linkerhelft van het interval ligt (`#2`). Zo ja (`#3`), dan gaan we verder met het linker interval; zo nee (`#4`), dan gaan we verder met de rechterhelft. In `#6` wordt het nieuwe midden berekend voordat alles bij `#1` weer opnieuw begint.
We hebben de loop niet met een dubbele punt maar met een puntkomma afgesloten. Daardoor wordt elke in regel `#6` berekende waarde voor x_m achtereenvolgens weergegeven. ◇

Regula Falsi. Een efficiëntere algoritme om een nulpunt te bepalen krijgen we door in plaats van het midden van het interval $[x_l, x_r]$ het snijpunt van de lijn door de punten $(x_l, f(x_l))$ en $(x_r, f(x_r))$ als volgende benadering te nemen. Als we dit snijpunt s noemen, dan nemen we weer, afhankelijk van het teken van $f(s)$ als het volgende interval hetzij $[x_l, s]$, hetzij $[s, x_r]$. Zie figuur 68.



FIGUUR 68. Regula Falsi

Als voorbeeld nemen we de functie uit de opgave van blz. 425.

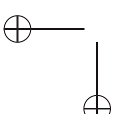
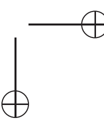
Voorbeeldsessie

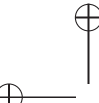
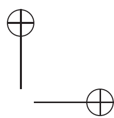
Afleiding formule snijpunt van de lijn door (x_l, y_l) en (x_r, y_r) met de x-as:

```
> lijn := x -> a*x + b:
> stelsel := { lijn(xl)=yl, lijn(xr)=yr };
```

$$stelsel := \{a\,xl + b = yl,\ a\,xr + b = yr\}$$

```
> opl := solve( stelsel, {a,b} );
```

$$opl := \{a = \frac{yl - yr}{-xr + xl}, b = \frac{xl\,yr - yl\,xr}{-xr + xl}\}$$

```
> s := solve( subs( opl, lijn(x) ) = 0, x );
```

$$s := -\frac{xl\,yr - yl\,xr}{yl - yr}$$

We benaderen een nulpunt van:

```
> f := x -> 9*x^3 + 5*x^2 + 8*x + 6;
```

$$f := x \to 9\,x^3 + 5\,x^2 + 8\,x + 6$$

Initialisatie:

```
> xl := -1:  xr := 1:
  yl := evalf(f(xl)):  yr := evalf(f(xr)):
```

```
> eps := 0.01:
```

Regula Falsi:

```
> while abs(f(s)) > eps do
    if f(s)*f(xl) < 0
    then xr := s else xl := s
    end if:
  yl := evalf(f(xl)):  yr := evalf(f(xr))
  end do;
```

$$yl := -6.$$

$$yr := 0.478729901$$

$$yl := -6.$$

$$yr := 0.135383517$$

$$yl := -6.$$

$$yr := 0.037305879$$

$$yl := -6.$$

$$yr := 0.010205130$$

Resultaat:

```
> s;
```

$$-0.6828656635$$

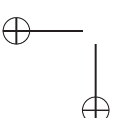
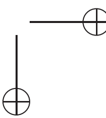
```
> f(s);
```

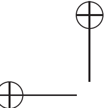
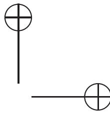
$$0.002786044$$

Toelichting

Uiteraard gebruiken we Maple ook om een uitdrukking voor s in termen van x_l, y_l, x_r en y_r te vinden. We gaan daarbij te werk als in de voorbeeldopgave op blz. 62; s is dan het nulpunt van de lijn $ax + b$.

Merk op dat we de gevonden uitdrukking voor s niet hoeven over te nemen in de while-loop. Telkens als Maple een `s` tegenkomt, dan wordt deze direct helemaal geëvalueerd.

We zien dat nu de benadering met de vereiste nauwkeurigheid in vier stappen in plaats van acht (zoals bij de intervalhalvering) wordt gevonden. ◇

Opgave 27.1

Beschouw de functie gedefinieerd door

```
f := x -> if (x<0) then -1 else 1 end if;
```

Als men de grafiek van deze functie wil laten tekenen, moet

```
plot(f, -10..10);
```

worden opgegeven, en niet

```
plot(f(x), x=-10..10);
```

Verklaar dit en verbeter deze plotopdracht door toevoegen van (gewone) quotes. (Aanwijzing: zie ook Module 25.)
Definieer `f` ook met `piecewise`, en ga na dat `plot(f(x), x=-10..10);` dan wél goed gaat.

Opgave 27.2

Zie de voorbeeldsessie op blz. 422. Test de drie varianten ook met andere lijsten, bijvoorbeeld:

(a) Het eerste getal > 4 staat op de eerste plaats in de lijst;
(b) Het eerste getal > 4 staat op de laatste plaats in de lijst;
(c) Alle getallen in de lijst zijn ≤ 4.

Wat gaat er mis, en hoe zou dat kunnen worden opgelost?

Opgave 27.3

Maak varianten van de voorbeeldsessies op blz. 425 en blz. 426 door als criterium te nemen dat de benaderde waarde van c minder dan 0.01 van de exacte waarde van c mag verschillen.

Opgave 27.4

Maak een variant van de Regula Falsi door voor s te nemen: het nulpunt van de tweedegraadskromme door de punten $(x_l, f(x_l))$, $(x_m, f(x_m))$ en $(x_r, f(x_r))$. Hierbij is $x_m = \frac{1}{2}\,(x_l + x_r)$.

Kies zelf een f (of neem de f van de voorbeeldopgave op blz. 425).
Aanwijzing: Pas op dat een tweedegraadspolynoom meestal *twee* nulpunten heeft. Zorg dat er een nulpunt in het beschouwde interval wordt gekozen.