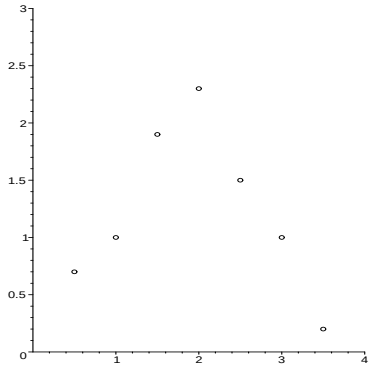
Module 18 Benaderende en interpolerende functies

Onderwerp	Continue en differentieerbare functies door gegeven punten; kleinste kwadraten-benaderingen
Voorkennis	Module 8
Expressies	`piecewise, Spline, LeastSquares, readdata`
Bibliotheken	`CurveFitting`
Bestanden	meting2.dat
Zie ook	Module 7, 8, 9, 13, 16, 28

18.1 Inleiding

In deze module construeren we functies van één variabele die zo goed mogelijk passen bij een gegeven rij van punten (x_i, y_i). Men kan hierbij denken aan metingen die zijn verricht om het verband tussen twee grootheden te onderzoeken. Op diverse plaatsen is zoiets als oefening of illustratie al eens gedaan, zie bijvoorbeeld §7.4, verder uitgewerkt in §28.6 (n^{de} graads polynoom door $n+1$ meetpunten) en Module 16 (opgave 16.4, kleinste kwadratenmethode).

We zullen twee methoden behandelen aan de hand van de volgende gefingeerde meetserie.



FIGUUR 17. De 'meetpunten' in deze module

Voorbeeldsessie

```
> restart; with(plots):
```

```
> XY :=
  [[0.5,0.7],[1,1],[1.5,1.9],[2,2.3],[2.5,1.5],[3,1],[3.5,0.2]]:
> pointplot(XY, color=black, symbol=CIRCLE, symbolsize=15,
  view=[0..4,0..3]);  pp := %:
```

(zie figuur 17)

Toelichting

De lijst `XY` bevat de 7 meetpunten; in `pp` wordt het plaatje bewaard. In de volgende voorbeeldsessies zullen deze gebruikt worden. ◇

18.2 Continue functies door gegeven punten

De allereenvoudigste manier om een functie $f : \mathbb{R} \to \mathbb{R}$ te maken waar de gegeven punten aan voldoen, is een functie die constant is op de intervallen $(x_i, x_{i+1}]$. Laten we voor die constante op het genoemde interval de waarde van y_{i+1} nemen. We definiëren f bij een gegeven rij punten $(x_1, y_1), \ldots, (x_n, y_n)$ als:

$$f(x) = \begin{cases} y_1 & \text{als } x \le x_1 \\ y_2 & \text{als } x_1 < x \le x_2 \\ \vdots & \vdots \\ y_n & \text{als } x_{n-1} < x \le x_n \\ 0 & \text{als } x > x_n\,. \end{cases}$$

`piecewise` Om zo'n functie te maken hebben we het commando `piecewise`, zie §3.4

Voorbeeldsessie

(de lijst XY en de `pointplot pp` nog bekend van de vorige sessie)

We maken een stuksgewijs constante functie:

```
> hulplijst := map( a -> (x <=a[1],a[2]), XY );
```

$$\textit{hulplijst} := [x \le 0.5,\ 0.7,\ x \le 1,\ 1,\ x \le 1.5,\ 1.9,\ x \le 2,\ 2.3,\ x \le 2.5,\ 1.5,\ x \le 3,\ 1,\ x \le 3.5,\ 0.2]$$

```
> f := unapply(piecewise( op(hulplijst), 0 ), x);
```

$$f := x \to \text{piecewise}(x \le 0.5,\ 0.7,\ x \le 1,\ 1,\ x \le 1.5,\ 1.9,\ x \le 2,\ 2.3,\ x \le 2.5,\ 1.5,\ x \le 3,\ 1,\ x \le 3.5,\ 0.2,\ 0)$$

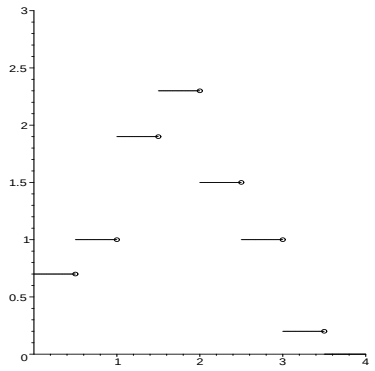
Nu hebben we ook direct de beschikking over de waarde op tussengelegen punten:

```
> f(1.25);
```

1.9

```
> ps0 := plot( f, 0..4, 0..3, color=black, discont=true ):
  display(pp,ps0);
```

(zie figuur 18)



FIGUUR 18. Een stuksgewijs constante functie

Toelichting

Het nut van een dergelijke functie is niet zozeer dat we er een plaatje van kunnen tekenen, maar vooral dat we met de aanroep $f(a)$ de beschikking hebben over een benaderde waarde van functiewaarden die niet in de lijst staan. We noemen dat *interpolatie*. ◇

De hierboven gedefinieerde functie f is niet continu. Een continue interpolatiefunctie krijgen we door alle opeenvolgende punten (x_i, y_i), (x_{i+1}, y_{i+1}) door rechte lijnen met elkaar te verbinden.
De grafiek van een dergelijke functie is wel heel gemakkelijk te tekenen, namelijk met `plot(XY)` (zie §9.3). De interpolatiefunctie kunnen we weer met `piecewise` maken, waarbij we voor het functievoorschrift op het interval $(x_i, x_{i+1}]$ de vergelijking voor de lijn (eerstegraadspolynoom) door de punten (x_i, y_i), (x_{i+1}, y_{i+1}) moeten nemen.

`Spline` Dit kunnen we de Maple-procedure `Spline` voor ons laten doen. Deze bevindt zich in de bibliotheek `CurveFitting`.

Voorbeeldsessie

(de lijst XY en de `pointplot pp` nog bekend van de vorige sessie)

```
> with(plots):  with(CurveFitting):
```

We verbinden de 'meetpunten' door rechte lijnen:

```
> s1 := Spline(XY, x, degree=1 );
```

$$s1 := \begin{cases} 0.4000000000 + 0.6000000000\,x & x < 1 \\ -0.8000000000 + 1.800000000\,x & x < 1.5 \\ 0.7000000000 + 0.8000000000\,x & x < 2 \\ 5.500000000 - 1.600000000\,x & x < 2.5 \\ 4.000000000 - 1.000000000\,x & x < 3 \\ 5.800000000 - 1.600000000\,x & otherwise \end{cases}$$

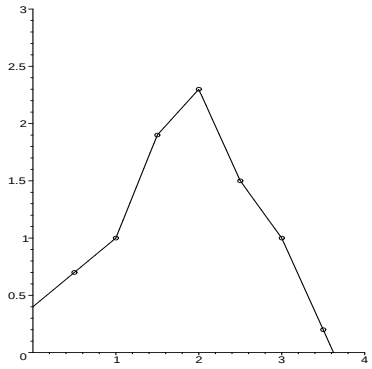
Functiewaarden op tussengelegen punten:

```
> p1 := unapply( s1, x ):    p1(1.25);
```

1.450000000

```
> ps1 := plot(p1, 0..4, 0..3, color=black):
  display(pp,ps1);
```

(zie figuur 19



FIGUUR 19. Lineaire interpolatiefunctie

Toelichting

Het eerste argument waarmee `Spline` wordt aangeroepen is een lijst van coördinaatparen, of een $n \times 2$-matrix waarvan de eerste kolom de x-coördinaten en de tweede de bijbehorende functiewaarden bevat[46]. Het tweede argument is de naam die we als onafhankelijke variabele willen gebruiken. Het derde argument heeft de vorm van een optie, en geeft de graad aan van de polynoomfuncties tussen opvolgende punten; in dit geval dus `degree=1` omdat we lineaire (eerstegraads) functies willen.
Het resultaat van een aanroep van `Spline` is een uitdrukking van de vorm `piecewise(x<x1,...)`. ◇

[46]Mag ook met twee lijsten of kolomvectoren, `X`, `Y`, waarbij de eerste een lijst van x-waarden en de tweede een lijst met bijbehorende y-waarden.

Zoals de optie in het `Spline`-commando al suggereert kunnen we ook interpolatiepolynomen van hogere graad maken. Een tweedegraadspolynoom bijvoorbeeld heeft drie parameters, en kan dus aan drie voorwaarden voldoen. De eerste twee voorwaarden zijn dat hij door de punten (x_i, y_i) en (x_{i+1}, y_{i+1}) moet gaan. De derde voorwaarde is dat de opvolgende polynomen *differentieerbaar* op elkaar moeten aansluiten, dus dat de afgeleide van de polynoom door (x_i, y_i) en (x_{i+1}, y_{i+1}) in het punt x_i gelijk is aan die van de polynoom door (x_{i-1}, y_{i-1}) en (x_i, y_i). Gebruikelijk is dan om voor de afgeleide in het beginpunt 0 te nemen.
Naarmate we een hogere graad kiezen kunnen, we continuïteit van de tweede, derde en hogere afgeleiden bereiken.

Voorbeeldsessie

(de lijst XY en de `pointplot pp` nog bekend van de vorige sessie)

```
> with(plots): with(CurveFitting):
```

We verbinden de 'meetpunten' door tweedegraadspolynomen:

```
> s2 := Spline(XY, x, degree=2 ):
  p2 := unapply( s2, x ):    p2(1.25);
```

1.432878788

```
> ps2 := plot(p2, 0..4, 0..3, color=black):
> display(pp,ps2, title="Tweedegraads-spline");
```

(zie figuur 20)

```
> s3 := Spline(XY, x, degree=3 ):
  p3 := unapply( s3, x ):    p3(1.25);
```

1.418125000

```
> ps3 := plot(p3, 0..4, 0..3, color=black):
> display(pp,ps3, title="Derdegraads-spline");
```

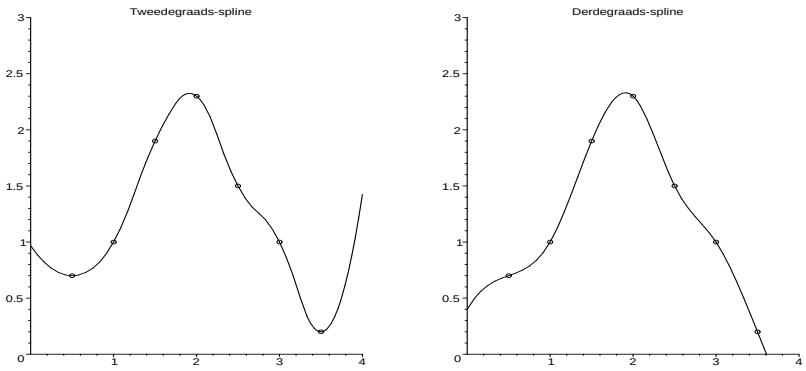
(zie figuur 20)

```
> plot( [D(p2),D(p3)], 0..4, -3..3, color=black,
    linestyle=[SOLID,DASH] );
```

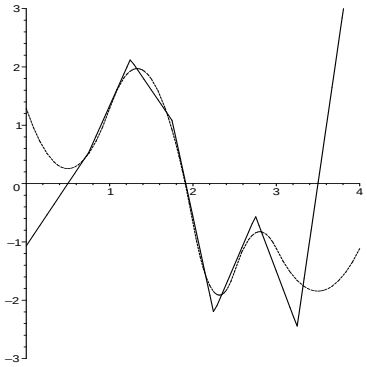
(zie figuur 21)

Toelichting

De derdegraads polynomen worden het vaakst gebruikt omdat ze meestal het meest lijken op een vloeiende kromme die men met de hand door de losse punten zou tekenen.
In figuur 21 zijn de afgeleiden van de beide splines afgebeeld. De afgeleiden van de tweedegraadspolynomen sluiten wél continu, maar niet differentieerbaar op elkaar aan; de afgeleiden van de derdegraadspolynomen hebben een 'vloeiend' verloop. ◇



FIGUUR 20. Tweede- en derdegraads interpolatiepolynomen



FIGUUR 21. Afgeleide van een tweedegraads spline (doorlopende lijn; met knikken) en van een derdegraads spline (streeplijn)

18.3 Kleinste kwadratenfuncties

Vaak zal men bij een serie meetgegevens $(x_1, y_1), \ldots, (x_n, y_n)$ op grond van (fysische) theorie een zeker verband veronderstellen van de vorm $y = f(x)$, waarbij f een formule is met nog een aantal parameters erin, bijvoorbeeld $f(x) = ax^2 + bx + c$, $f(x) = a\,e^{bx}$ enzovoort. Voor de meetwaarden zal dan gelden

$$y_i = f(x_i) + \varepsilon_i, \quad i - 1, \ldots, n$$

met ε_i de *meetfout*. De *kleinste kwadratenbenadering* van f is dan een zodanige keuze van de parameters in f dat $\sum_{i=1}^{n} \varepsilon_i^2$ minimaal is.

Leastsquares

In Module 16, opgave 16.4, zijn de parameters berekend voor het geval dat f een eerste- of tweedegraads polynoom is. Via de bibliotheek `CurveFitting` hebben we ook de beschikking over de procedure `LeastSquares` waarmee dat in één commando kan. We gebruiken weer dezelfde lijst gefingeerde meetgegevens als in de vorige sessies.

Voorbeeldsessie

```
> restart; with(plots):
> XY :=
  [[0.5,0.7],[1,1],[1.5,1.9],[2,2.3],[2.5,1.5],[3,1],[3.5,0.2]]:
> pp := pointplot(XY, color=black, symbol=CIRCLE,
                  symbolsize=15, view=[0..4,0..3]):
> with(CurveFitting):
> ls1 := LeastSquares( XY, x );
```

$$ls1 := 1.5000000000 - 0.1357142857\,x$$

```
> pls1 := plot( ls1, x=0..4, color=black ):
> ls2 := LeastSquares( XY, x, curve=a*x^2+b*x+c );
```

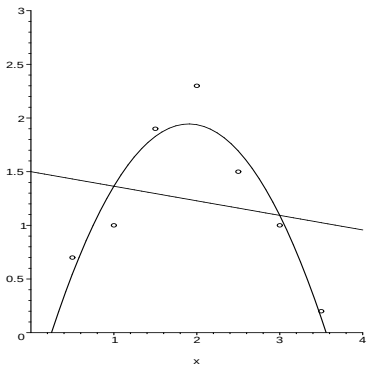
$$ls2 := -0.6285714286 + 2.7023809524\,x - 0.7095238095\,x^2$$

```
> pls2 := plot( ls2, x=0..4, color=black ):
  display(pp,pls1,pls2);
```

(zie figuur 22)

```
> LeastSquares( XY, x, curve=a*x^b );

Error, (in CurveFitting:-LeastSquares) curve to fit is not
linear in the parameters
```



FIGUUR 22. Eerste- en tweedegraads kleinste kwadratenpolynoom door de 'meetpunten'

Toelichting

Als `Leastsquares` met twee argumenten wordt aangeroepen, dan

wordt aangenomen dat $f(x)$ van de gedaante $ax + b$ is (*lineaire regressie*). Een functie van een andere gedaante kan worden opgegeven door de optie `curve=...` .
Het resultaat van de laatste aanroep, met `curve=a*x^b`, is enigszins teleurstellend. Als u zich opgave 16.4 in herinnering roept is dat echter wel te begrijpen. Het bepalen van k parameters komt neer op het oplossen van k vergelijkingen met k onbekenden. Dat gaat alleen gegarandeerd goed als deze vergelijkingen lineair zijn, en dat is het geval als $f(x)$ lineair is in de parameters, dat wil zeggen: $f(x) = a_1\, f_1(x) + \cdots + a_k\, f_k(x)$. ◇

Dat is natuurlijk lang niet altijd het geval. Vaak gebruikt worden bijvoorbeeld exponentiële functies, dus $f(x) = \alpha e^{\beta x}$. Ook functies van de vorm $f(x) = ax^b$ (zogenaamde *Cobb-Douglas functies*) worden in de praktijk zeer vaak gebruikt, namelijk als we eisen dat moet gelden: $f(0) = 0$ en f strikt stijgend voor alle $x \geq 0$.
Dergelijke functies kunnen lineair in de parameters worden gemaakt door logaritmen te nemen. Bijvoorbeeld voor de Cobb-Douglas functie geldt:

$$y = ax^b \quad \Leftrightarrow \quad \ln y = \ln a + b \ln x\,. \tag{18.1}$$

Door dus $\ln x_i$ en $\ln y_i$ als nieuwe variabelen te nemen wordt de functie lineair in de parameters. U kunt dit toepassen in opgave 18.1.
Een andere belangrijke categorie wordt gevormd door *periodieke functies*. Daar gaan we in Module 19 verder op in.

18.4 Import van gegevens uit een bestand

Meetgegevens waarvoor we een interpolatiepolynoom of een kleinste kwadratenfunctie willen maken, zullen in het algemeen van een ander programma dan Maple afkomstig zijn. Ze zullen dan ook meestal niet de vorm van Maple-lijsten `[[x1,y1],[x2,y2],...]` of `[x1,x2,...], [y1,y2,...]` hebben. Voor 'normale', eenvoudige gevallen bestaat het commando `readdata`, dat veronderstelt dat de gegevens in de volgende vorm in een bestand zitten:

`readdata`

```
   .981    4.231   1.2345
-12.1      8      125
  3.1      1      3E-10
```

In dit voorbeeld zijn dat dus drie getallen per regel. Als het bestand zich met de naam `meting.dat` op een diskette zou bevinden, dan is het resultaat van het commando

```
XY := readdata( "a:meting.dat", 2 );
```

dat `XY` de lijst `[[.981,4.231],[-12.1,8.0],[3.1,1.0]]` wordt. Het tweede argument van `readdata` geeft dus aan hoeveel getallen er per regel moeten worden gelezen. Er worden automatisch getallen met een decimale punt van gemaakt, maar er zijn opties waardoor wat subtieler met de data kan worden omgesprongen. Raadpleeg `?readdata` voor meer informatie

In veel gevallen kan heel goed worden volstaan met de Import Data... *assistant*, zie §14.6 (blz. 196). Een voordeel hiervan is dat je door je folders kunt 'bladeren', dus geen ellenlange padnamen hoeft in te typen. Een nadeel zou kunnen zijn dat het *hele* bestand gelezen wordt, en dat het resultaat in een matrix van kommagetallen (*floats*) komt. Dat is trouwens niet zo erg, zie §13.2 voor het weglaten van kolommen of rijen uit een matrix. Met

`convert, listlist`

```
L := convert( A, listlist )
```

wordt matrix `A` omgezet in een lijst van lijsten `B`, iedere rij van de matrix wordt een sublijst van `B`. Als u de gegevens bijvoorbeeld wilt ordenen, is zo'n lijst van lijsten handiger dan een matrix.

Voor ingewikkelder gevallen.kunt u gebruik maken van `fscanf`. Deze procedure kan in principe alles aan, dus bijvoorbeeld ook bestanden die naast numerieke gegevens ook andere zaken bevat; raadpleeg `?fscanf` voor de details.

Opgave 18.1

Het bestand `meting2.dat` bevat twee getallen (x_i, y_i) per regel. Er wordt verondersteld dat tussen de x- en y-waarden een verband $y = f(x)$ bestaat, met $f(0) = 0$ en f strikt stijgend.

(a) Lees het bestand met `readdata` (zie Appendix 1).

(b) Voeg het punt $(0, 0)$ toe aan de lijst. Maak achtereenvolgens splines van de graad $1, 2, 3, 4$ en vergelijk de grafieken.

(c) Maak de eerste- en tweedegraads kleinste kwadratenpolynoom. Denk aan de voorwaarde $f(x) = 0$. Maak een plaatje als figuur 22.

(d) Maak een kleinste kwadraten Cobb-Douglas-benadering van de gegevens.