MATLAB Quickstart

Versie 1.11 voor MATLAB 6.x

G.J. Bex

October 6, 2005

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

1.1 Doel en doelpubliek van deze tekst

Deze tekst is een inleiding tot het gebruik van MATLAB en heeft een tweeledig doel: (1) de student vertrouwd maken met de basis van MATLAB en (2) als referentie dienst doen voor later gebruik.

Een relatief beknopte tekst zoals deze kan niet hopen volledig te zijn, er wordt echter geprobeerd de beginnende gebruiker van MATLAB een startpunt te bieden. De behandelde wiskunde komt ongeveer overeen met het niveau behaald na een eerste jaar van een beta-wetenschapsopleiding.

1.2 Wat is MATLAB?

MATLAB is een software pakket dat de gebruiker toelaat op een relatief eenvoudige manier diverse numerieke berekeningen uit te voeren. Het is uiteraard geen vervanging voor een gedegen kennis van de wiskunde, wel een hulpmiddel dat vervelend rekenwerk kan overnemen. Daarnaast heeft MATLAB zeer veel mogelijkheden voor visualisatie van functies en data, hetgeen kan bijdragen tot een beter begrip.

1.3 Overzicht

Het eerste deel geeft een kort overzicht over de grafische user interface van Matlab. Er wordt een overzicht gegeven van de belangrijkste vensters en hoe data uitgewisseld kan worden met andere programma's of bewaard voor later gebruik in MATLAB.

Het tweede deel behandelt het rekenen met Matlab. In de eerste sectie komen eenvoudige numerieke berekeningen aan bod, maar daarna hebben we het over het "hart" van MATLAB: vectoren en matrices. Functies spelen een centrale rol in de wiskunde, hierop wordt dan ook uitgebreid in gegaan in sectie 3.4. Secties 3.5 en 3.6 behandelen visualisatie van functies in twee en drie dimensies. Integratie, differentiatie en minimalisatie van functies in één en meerdere veranderlijken komen aan bod in de volgende secties. Er wordt hier ook kort ingegaan op vectorvelden. In sectie 3.11 wordt het oplossen vergelijkingen in één veraderlijke en van stelsels vergelijkingen besproken, terwijl de laatste sectie differentiaalvergelijkingen behandelt.

In het derde deel wordt oppervlakkig ingegaan op het schrijven van programma's in MATLAB. De klemtoon ligt hier vooral op het schrijven van functies waarin controlestructuren zoals voorwaardelijke en herhalingsopdrachten gebruikt worden.

Tot slot is er nog een bibliografie voorzien waarin enkele boeken over MATLAB vermeld staan. Merk echter op dat het help systeem van MATLAB van zeer goede kwaliteit is zodat extra literatuur niet noodzakelijk is.

Hoofdstuk 2

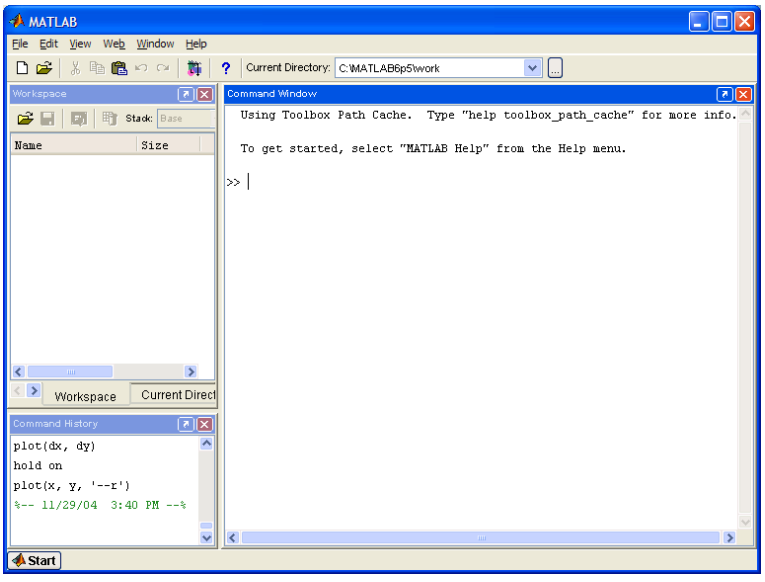
User interface

2.1 MATLAB venster

Wanneer je MATLAB voor het eerst opstart krijg je een scherm te zien zoals in figuur 2.1. De rechterkant ervan, het '`Command Window`', is het belangrijkst, hierin worden de MATLAB commando's ingegeven en de resultaten van berekeningen getoond. Figuur 2.2 toont hetzelfde venster nadat enkele commando's ingegeven werden aan de prompt '>>'.

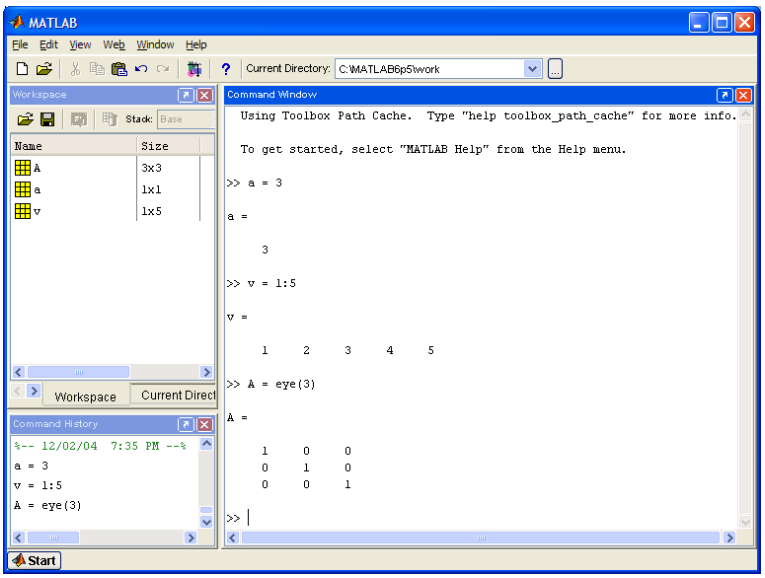
Vaak wil je een commando dat je net hebt gebruikt aanpassen en opnieuw uitvoeren. Als je in het '`Command Window`' de pijltjestoetsen gebruikt krijg je het voorgaande commando (m.b.v. '↑') of het volgende commando (m.b.v. '↓').

Wanneer je een ingewikkelde uitdrukking op meerdere invoerlijnen aan het opbouwen bent, e.g. een '**for**' en het blijkt dat je in één van de vorige lijnen vergist hebt, dan kan je de invoer afbreken en een nieuwe prompt krijgen door '`Ctrl-c`' te drukken. Deze toetsencombinatie onderbreekt ook een berekening, hetgeen nuttig is wanneer je een te ingewikkelde berekening probeert te maken die meer tijd kost dan je eraan kan of wil spenderen.



Figuur 2.1: MATLAB's hoofdscherm

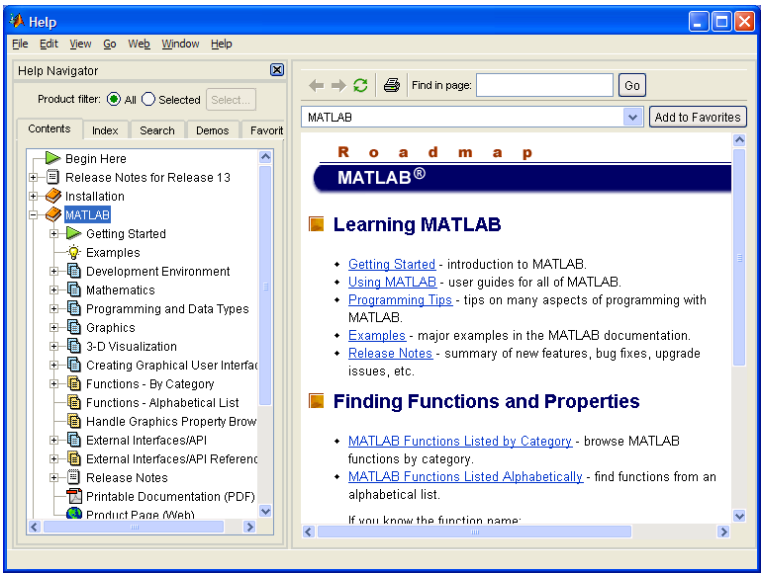
Het bovenste linkerdeel van dit venster, de '`Workspace`', toont een lijst met variabelen (zie ook sectie 3.1.2) die in gebruik zijn. Naast de naam wordt ook de grootte en type weergegeven. Het onderste linkerdeel, de '`Command History`', toont de lijst van commando's die recent ingegeven werden. Merk op dat die lijst bewaard blijft tussen verschillende MATLAB sessies.



Figuur 2.2: MATLAB's hoofdscherm na enkele berekeningen

2.2 Help venster

Een ander venster dat je veel zal gebruiken is MATLAB help, zie figuur 2.3. Dit lijkt zeer sterk op de helpsystemen van de meeste Microsoft Windows toepassingen. Je kan de inhoud bekijken, zoeken op namen van functies of de inhoud van de help bestanden, via de index het commando vinden dat je zoekt en zo verder.



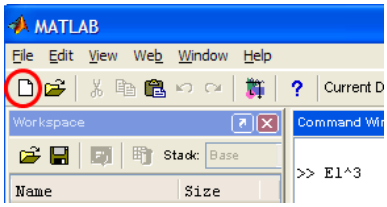
Figuur 2.3: MATLAB's helpvenster

2.3 Figure venster

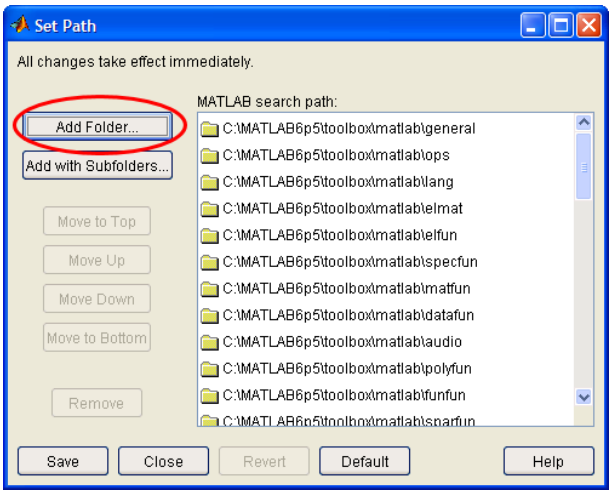
Een derde soort venster gebruikt MATLAB om plots te tonen, zowel voor 2D- als voor 3D-figuren. Voorbeelden hiervan zijn te zien in figuren 3.1 en 3.4. In dit venster kunnen de eigenschappen van de plots gemakkelijk aangepast worden. Ze kunnen ook herschaald en — voor 3D-plots — geroteerd worden. Dit kan natuurlijk ook via de command line.

2.4 Editor venster

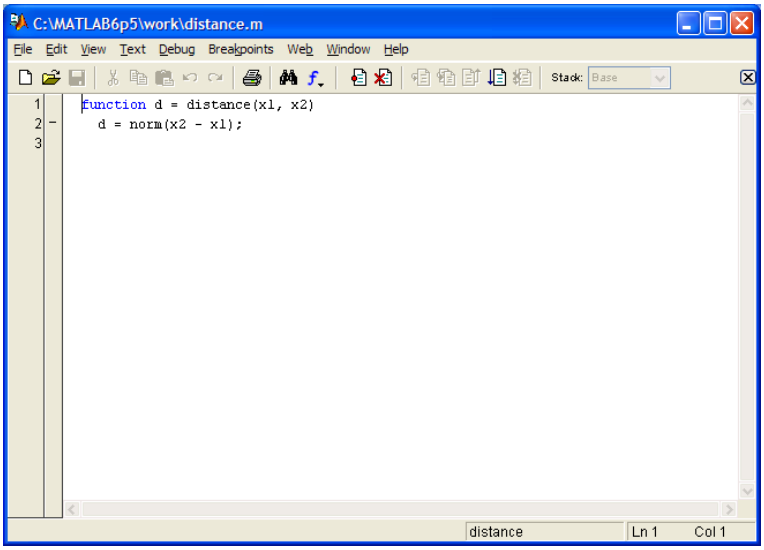
Een laatste type venster in figuur 2.6 laat toe M-files te editeren (zie sectie 3.4.2 voor toelichting over M-files). Je vindt er de functionaliteit in die je kent van een tekstverwerker of een editor. Een nieuwe M-file maak je aan door op de button te klikken die rood omcirkeld is in figuur 2.4 zodat een venster zoals getoond in figuur 2.6 opent.



Figuur 2.4: Een nieuwe M-file openen



Figuur 2.5: Een directory toevoegen aan MATLAB's path



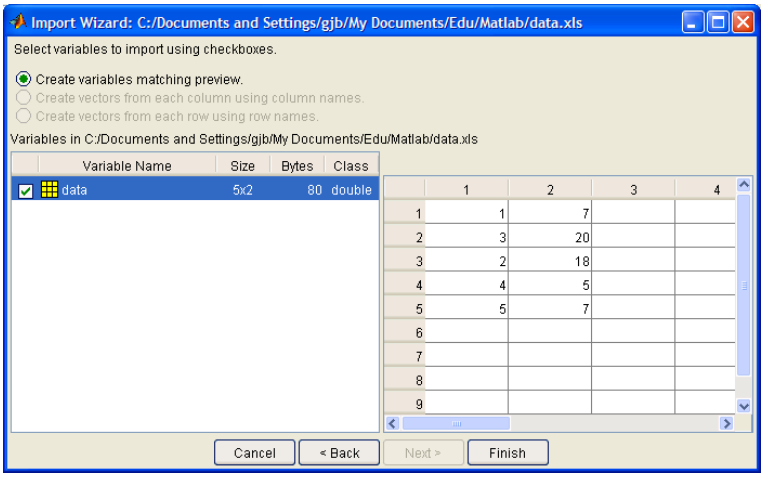
Figuur 2.6: MATLAB's editor venster

Merk op dat je M-files op de harde schijf moet opslaan vooraleer je ze kan gebruiken. Dit kan in om het even welke folder, maar MATLAB moet deze wel weten te vinden. MATLAB zoekt files in een lijst van folders, het zogenaamde 'path'. Je kan eenvoudig een folder toevoegen aan het 'path' door uit het '`File`' menu '`Set Path...`' te kiezen zodat het dialoogvenster in figuur 2.5 opent. Je kan de folder waarin je de M-file bewaard hebt toevoegen met behulp van de button '`Add Folder`'.

2.5 Data in- en uitvoer

Er werd al een aantal keer op gewezen dat MATLAB bij uitstek geschikt is voor data-analyse en -visualisatie. Het spreekt dus vanzelf dat de nodige mogelijkheden voorzien zijn om externe gegevens zoals resultaten van experimenten in te voeren in MATLAB. Het gaat hierbij wel te verstaan over informatie die voorgesteld is als een tabel.

De eenvoudigste manier om dit soort data te importeren maakt gebruik van de data import wizard. In figuur 2.7 wordt bij wijze van voorbeeld een Microsoft Excel bestand geladen. Er wordt een groot aantal bestandsformaten ondersteund waaronder een aantal spreadsheet formaten zoals CSV (Comma Separated Values) naast Excel, maar ook grafische bestanden in GIF en JPEG formaat. Zelfs geluid kan ingelezen worden voor analyse.



Figuur 2.7: MATLAB's data import wizard voor het inlezen van gegevens uit externe bestanden, hier gebruikt om een Microsoft Excel bestand '`data.xls`' aan de variabele 'data' toe te kennen

Naast de invoer met de import wizard kan men ook data importeren met behulp van functies. Voor elk formaat is er een specifieke functie, e.g. 'xlsread' en '**csvread**' voor Excel en CSV. Een voorbeeld van het gebruik hiervan wordt hieronder getoond:

```
>> T = xlsread('c:/Experiments/Experiment3/data.xls')
T =
     1     7
     3    20
     2    18
     4     5
     5     7
```

Voor gebruik in Microsoft Excel kan data weggeschreven worden in CSV formaat. Hiervoor gebruikt men de functie '**csvwrite**', e.g. '**csvwrite**('c:/Data/T.csv', T)' zodat de matrix 'T opgeslagen wordt in het bestand '`T.csv`' dat er als volgt uitziet:

```
1,7
3,20
2,18
4,5
5,7
```

Merk op dat er geen equivalent bestaat voor het schrijven van Microsoft Excel bestanden.

Men kan berekeningen tussen MATLAB-sessies bewaren in een zogenaamde MAT-file. Daarin worden alle of geselecteerde variabelen opgeslagen die men in een sessie gebruikt heeft. De

eenvoudigste manier om dit te doen is het via het `File`-menu '`Save Workspace As...`' te selecteren om de informatie te bewaren, '`Open...`' om ze weer te laden. Het is eenvoudig een aantal variabelen te bewaren en weer te laden met behulp van de functies '**save**' en '**load**'. De variabelen 'a' en 'v' kunnen geschreven worden naar en gelezen uit het bestand '`data.mat`' met behulp van respectievelijk de commando's '**save** 'data.mat' a v' en '**load** 'data.mat' a v'.

Hoofdstuk 3

Rekenen met MATLAB

3.1 Eenvoudige berekeningen

MATLAB kan gebruikt worden als gesofisticeerde rekenmachine. Vanzelfsprekend worden de rekenkundige bewerkingen ondersteund:

```
>> 5*3 + 2/3 - 0.5*(2^3 - 1)
ans =
   12.1667
```

Hier stellen $*$ en $\hat{}$ respectievelijk de vermenigvuldiging en de machtverheffing voor. Er zijn ook een groot aantal wiskundige functies voorgedefinieerd zoals de vierkantswortel:

```
>> sqrt(3.0)
ans =
    1.7321
>> 1/sqrt(2)
ans =
    0.7071
```

de trigoniometrische functies:

```
>> sin(pi/4)
ans =
    0.7071
>> tan(pi/2)
ans =
  1.6331e+016
```

Merk op dat de constante π onder de naam **pi** voorgedefinieerd is. Uit de laatste berekening blijkt ook duidelijk dat je bij het maken van numerieke berekeningen voorzichtig moet zijn. Immers, de $\lim_{x\to\pi/2}\tan(x) = +\infty$. Het MATLAB resultaat is weliswaar een groot getal, maar niet exact.
De exponentiële functie (**exp**), de logaritme met basis e (**log**), 10 (**log10**) en 2 (**log2**) respectievelijk:

```
>> exp(3.0)
ans =
   20.0855
>> log(ans)
ans =
     3
```

```
>> log10(1000)
ans =
     3
>> log2(32)
ans =
     5
```

3.1.1 Formaat

Hoewel er bij deze berekeningen maar 5 beduidende cijfers getoond worden werkt MATLAB inwendig in dubbele precisie. Je kan het formaat waarin MATLAB resultaten weergeeft zelf instellen met behulp van het commando **format**:

```
>> sqrt(2)
ans =
    1.4142
>> format long
>> ans
ans =
   1.41421356237310
>> format short
>> ans
ans =
    1.4142
```

Zoals je ziet beïnvloedt **format** niet de nauwkeurigheid van het resultaat, enkel de vorm waarin het op het scherm getoond wordt.

3.1.2 Variabelen

Dit laatste voorbeeld illustreert nog een interessant concept. Het resultaat van de berekening wordt automatisch toegekend aan een variabele met als naam **ans**. Je kan — zoals hierboven — later de waarde van deze variabele opnieuw opvragen of ze gebruiken bij verdere berekeningen:

```
>> sqrt(3)
ans =
    1.7321
>> ans*ans
ans =
    3.0000
```

Je kan een variabele zien als een geheugenplaats met een naam, **ans** in dit geval. Je kan echter ook zelf het resultaat van een berekening toekennen aan een variabele zoals hieronder:

```
>> x = sin(0.5)
x =
    0.4794
>> y = cos(0.5)
y =
    0.8776
>> x^2 + y^2
ans =
     1
```

De laatste toekenning aan de variabele bepaalt de waarde van de variabele, met andere woorden, de waarde die een variabele had gaat verloren bij een nieuwe toekenning:

```
>> x = 10
x =
    10
>> x = 11
x =
    11
>> x
x =
    11
```

De naam van een variabele moet beginnen met een hoofd- of kleine letter, de overige karakters mogen maximaal 63 letters, cijfers of '_' zijn. Merk op dat hoofd- en kleine letters verschil maken voor de namen van variabelen:

```
>> X = 10
X =
    10
>> x
x =
    11
>> X
X =
    10
```

3.1.3 Complexe getallen

Rekenen met complexe getallen is zeer eenvoudig in MATLAB, hieronder staan enkele voorbeelden:

```
>> a = 3 + 5*i; b = 7 + 11*i;
>> a + b
ans =
  10.0000 +16.0000i
>> a*b
ans =
 -34.0000 +68.0000i
>> a/b
ans =
   0.4471 + 0.0118i
```

Om het reëel, het imaginair deel, het complex toegevoegde en de norm van een complex getal te bepalen heeft men respectievelijk de functies '**real**', '**imag**', '**conj**' en '**abs**'.

```
>> real(a)
ans =
     3
>> imag(a)
ans =
     5
>> conj(a)
ans =
```

```
   3.0000 - 5.0000i
>> abs(a)
ans =
    5.8310
```

De trigoniometrische en expontiële functies zijn gedefinieerd op complexe getallen.

3.2 Vectoren

Vectoren spelen een erg belangrijke rol in MATLAB, ze vormen het hart van het systeem.

3.2.1 Constructie

Zoals je weet uit de cursus wiskunde zijn er twee soorten vectoren: rij- en kolomvectoren. Beide types bestaan ook in MATLAB. Een rijvector kan je eenvoudig definiëren zoals hieronder:

```
>> u = [1  2  4]
u =
     1     2     4
```

Deze vector heeft drie componenten met als waarden 1, 2 en 4 respectievelijk. Bij een kolomvector worden de componenten door ; van elkaar gescheiden:

```
>> v = [2;  3;  6]
v =
     2
     3
     6
```

Vaak wil men vectoren waarvan de componenten opeenvolgende waarden hebben. Deze zijn zeer eenvoudig te definiëren in MATLAB. Het volgende voorbeeld maakt een vector met vijf componenten:

```
>> w = 1 : 5
w =
     1     2     3     4     5
```

De volgende vector heeft ook vijf componenten, maar de waarden hebben een interval van 2 in plaats van 1.

```
>> w = 1 : 2 : 10
w =
     1     3     5     7     9
```

De componenten zowel als het interval kunnen reële getallen zijn:

```
>> w = 0.0 : 0.5 : 2.0
w =
         0    0.5000    1.0000    1.5000    2.0000
```

Individuele componenten van een vector kunnen opgevraagd of veranderd worden door een index te gebruiken.

```
>> w(2)
ans =
    0.5000
>> w(3) = -1
```

```
w =
         0    0.5000   -1.0000    1.5000    2.0000
```

Het is ook mogelijk een vector "slice" te maken, i.e. een vector die bestaat uit een gedeelte van de componenten van een andere.

```
>> w(2 : 4)
ans =
    0.5000   -1.0000    1.5000
```

Je kan zeer eenvoudig een vector construeren die enkel componenten bevat met de waarde 0 door middel van de functie '**zeros**'. Merk op dat als deze functie opgeroepen wordt met één parameter er een vierkante matrix met die dimensies aangemaakt wordt. De functie 'ones' werkt analoog, maar levert een vector die enkel 1 als componenten bevat. Soms is het nuttig een vector met willekeurige componenten te creëren, hiervoor kan men de functie '**rand**' gebruiken. De componenten van de vector zullen pseudo-random getallen tussen 0 en 1 zijn. Je krijgt bij elke aanroep een vector met andere waarden voor de componenten zoals hieronder geïllustreerd wordt.

```
>> zeros(1, 3)
ans =
     0     0     0
>> ones(1,3)
ans =
     1     1     1
>> rand(1, 3)
ans =
    0.9501    0.2311    0.6068
>> rand(1, 3)
ans =
    0.4860    0.8913    0.7621
```

3.2.2 Bewerkingen

Er zijn een groot aantal bewerkingen met vectoren gedefinieerd waaronder de optelling en aftrekking:

```
>> u = [3 5 7]
u =
     3     5     7
>> v = [2 4 6]
v =
     2     4     6
>> u + v
ans =
     5     9    13
>> u - v
ans =
     1     1     1
```

Bij elke component van een vector een getal optellen of ermee vermenigvuldigen is erg eenvoudig:

```
>> 2 + u
ans =
```

```
     5     7     9
>> 2*u
ans =
     6    10    14
```

Een vector transponener, i.e. een kolomvector van een rijvector maken of omgekeerd, kan met behulp van de operator '.'':

```
>> v.'
ans =
     2
     4
     6
```

Naast de getransponeerde vector kan men ook de geconjugeerde berekenen met behulp van de operator '', naast een transpositie wordt hierbij ook nog het complex toegevoegde van iedere component genomen. Uiteraard is 'u'' identiek aan 'u.'' voor vectoren met reële componenten.

```
>> z = [1 + i  3 - 2*i  4]
z =
   1.0000 + 1.0000i   3.0000 - 2.0000i   4.0000
>> z'
ans =
   1.0000 - 1.0000i
   3.0000 + 2.0000i
   4.0000
```

Het scalair product van twee vectoren $\vec{u}.\vec{v} = \sum_{i=1}^{i=3} u_i v_i$ kan op twee manieren berekend worden:

```
>> dot(u, v)
ans =
    68
```

of nog:

```
>> u*v.'
ans =
    68
```

Vooraleer het scalaire product te berekenen hebben we de vector v eerst getransponeerd. Indien we dit niet zouden doen krijgen we bij het uitvoeren van het scalair product de volgende foutmelding:

```
>> u*v
??? Error using ==> *
Inner matrix dimensions must agree.
```

Naast het scalaire product (of in-product) kan je ook het uitproduct van twee vectoren berekenen:

```
>> A = u.'*v
A =
     6    12    18
    10    20    30
    14    28    42
```

De componenten van deze matrix A worden gegeven door $A_{ij} = u_i v_j$.

Het vectorieel product $\vec{u} \times \vec{v}$ van twee vectoren $\vec{u}$ en $\vec{v}$ dat gedefinieerd is als

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{e}_x & \vec{e}_y & \vec{e}_z \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}$$

kan ook eenvoudig berekend worden met behulp van de functie '**cross**':

```
>> cross(u, v)
ans =
     2     -4      2
```

Het aantal componenten van een vector kan bepaald worden met behulp van de functie **length**. De norm van een vector $\vec{u}$ is gedefinieerd als $\sqrt{u_1^2 + u_2^2 + u_3^2}$ en kan in MATLAB eenvoudig berekend worden:

```
>> length([3 5 7])
ans =
     3
>> norm([3 5 7])
ans =
    9.1104
```

Merk op dat de terminologie verwarrend kan zijn: de functie '**norm**' berekend de lengte van een vector $\vec{u}$ als $\sqrt{u_1^2 + \cdots + u_n^2}$ waarbij n het aantal componenten is gegeven door de functie '**length**'.

Er zijn nog een aantal functies die het vermelden waard zijn: '**min**', '**max**', **sum**' en '**prod**' die respectievelijk de kleinste en grootste component, de som en het product van de vector componenten berekenen.

```
>> min([3 5 7])
ans =
     3
>> max([3 5 7])
ans =
     7
>> sum([3 5 7])
ans =
    15
>> prod([3 5 7])
ans =
   105
```

Voor statistische berekeningen heeft Matlab ook een aantal functies die nuttig kunnen zijn, we vermelden hier enkel '**mean**' en '**median**'.

Aangezien in MATLAB het gebruik van vectoren centraal staat zijn er operatoren die componentsgewijs werken. Zo is het bijvoorbeeld mogelijk een vector $\vec{w}$ te bekomen uit de vectoren $\vec{u}$ en $\vec{v}$ zo dat $w_i = u_i v_i$ of $w_i = u_i / v_i$, respectievelijk met behulp van de operatoren '.*' en './'.

```
>> u = [1 3 5]; v = [2 4 6];
>> u.*v
ans =
     2     12     30
>> u./v
```

```
ans =
    0.5000    0.7500    0.8333
```

Analoog kan je een vector $\vec{v}$ bekomen waarvan de componenten die zijn van een vector $\vec{u}$ verheven tot een zekere macht n, i.e. $v_i = u_i^n$. De operator hiervoor is '.^'.

```
>> [1 3 5].^3
ans =
     1    27   125
```

Het is belangrijk het verschil in te zien tussen de operator voor het scalair product '$*$' en de componentsgewijze vermenigvuldiging '$.*$'. Het eerste geeft als resultaat een getal, tweede tweede een vector.

```
>> [1 2 3]*[2 3 4]
??? Error using ==> *
Inner matrix dimensions must agree.
>> [1 2 3]*[2; 3; 4]
ans =
    20
>> [1 2 3].*[2 3 4]
ans =
     2     6    12
>> [1 2 3].*[2; 3; 4]
??? Error using ==> .*
Matrix dimensions must agree.
```

Ook de meeste functies werken componentsgewijs op vectoren, ter illustratie:

```
>> w = [1 4 9 16 25]
w =
     1     4     9    16    25
>> sqrt(w)
ans =
     1     2     3     4     5
```

De functie '**sqrt**' wordt dus toegepast op elke component van de vector en de resulterende vector heeft dit resultaat voor elk van zijn componenten. Dit werkt voor de meeste functies met één parameter zoals '**sin**', '**cos**',..., maar ook voor functies die je zelf definieert (zie sectie 3.4).

3.3 Matrices

Werken met matrices en vectoren is zeer analoog in MATLAB. Veel operaties zijn speciaal geoptimaliseerd voor matrices. Ook deze datastrukturen spelen een zeer belangrijke rol.

3.3.1 Constructie

Het definiëren van een matrix A met twee rijen en drie kolommen zowel als een matrix B met drie rijen en kolommen:

```
>> A = [1 4 9; 3 9 27]
A =
     1     4     9
     3     9    27
>> B = [1 2 3; 4 5 6; 7 8 9]
```

```
B =
     1     2     3
     4     5     6
     7     8     9
```

De rijen worden van elkaar gescheiden door ';', de kolommen door spaties. Het kan overzichtelijker zijn een matrix op de volgende manier in te geven:

```
>> C = [6  5  4;
        3  2  1]
C =
     6     5     4
     3     2     1
>> D = [6  5;
        3  2]
D =
     6     5
     3     2
```

Net zoals je individuele componenten van vectoren kan opvragen en aanpassen kan dit ook voor matrices.

```
>> C(1,2)
ans =
     5
>> C(2,1) = 8
C =
     6     5     4
     8     2     1
```

Een kolom of een rij van een matrix is kan in zijn geheel geselecteerd worden met behulp van de ':' operator.

```
>> C(:,2)
ans =
     5
     2
>> C(2,:)
ans =
     8     2     1
```

In sectie 3.2.1 maakten we al kennis met de functies '**zeros**', 'ones' en '**rand**' die als speciaal geval vectoren opleverden. Het is duidelijk dat deze functies matrices van willekeurige vorm kunnen aanmaken. Hieronder geven we een voorbeeld van elk, telkens voor een 2×3 matrix.

```
>> zeros(2, 3)
ans =
     0     0     0
     0     0     0
>> ones(2, 3)
ans =
     1     1     1
     1     1     1
>> rand(2, 3)
ans =
```

```
    0.4565    0.8214    0.6154
    0.0185    0.4447    0.7919
```

Er zijn nog enkele functies die toelaten eenvoudig speciale matrices te construeren. De functie '**eye**' levert de identiteitsmatrix op:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

Meer algemeen produceert de functie '**diag**' een diagonaalmatrix gegeven de vector met de waarden van de componenten op de diagonaal:

```
>> diag([3  5  7])
ans =
     3     0     0
     0     5     0
     0     0     7
```

3.3.2 Bewerkingen

De optelling en aftrekking van twee matrices is gedefinieerd wanneer deze hetzelfde aantal rijen en kolommen hebben:

```
>> A + C
ans =
     7     9    13
     6    11    28
>> A - C
ans =
    -5    -1     5
     0     7    26
```

Net zoals bij vectoren is het ook bij matrices eenvoudig een getal bij de componenten op te tellen of ze ermee te vermenigvuldigen:

```
>> 3 + A
ans =
     4     7    12
     6    12    30
>> 2*A
ans =
     2     8    18
     6    18    54
```

Matrices kunnen, net zoals vectoren, getransponeerd en geconjugeerd worden met de operatoren '.'' en ''', e.g.

```
>> [1 + i    -2 - 3*i;
    2          1 - i].'
ans =
   1.0000 + 1.0000i   2.0000
  -2.0000 - 3.0000i   1.0000 - 1.0000i
```

```
>> [1 + i    -2 - 3*i;
    2          1 - i]'
ans =
   1.0000 - 1.0000i   2.0000
  -2.0000 + 3.0000i   1.0000 + 1.0000i
```

Twee matrices kunnen met elkaar vermenigvuldigd worden wanneer het aantal kolommen van de eerste gelijk is aan het aantal rijen van de tweede:

```
>> A*B
ans =
    80    94   108
   228   267   306
```

Wanneer dit niet het geval is krijgen we de volgende foutmelding:

```
>> A*C
??? Error using ==> *
Inner matrix dimensions must agree.
```

Wanneer we een vierkante een aantal keer met zichzelf willen vermenigvuldigen dan kan dit eenvoudig met behulp van de operator '^'. Zo is e.g. 'B^3' equivalent met 'B*B*B'.

In sommige gevallen — zie bijvoorbeeld sectie 3.6 — is het nodig twee matrices componentsgewijs te vermenigvuldigen, te delen of tot een macht te verheffen. Het resultaat is dus een matrix met evenveel rijen en kolommen als de oorspronkelijke matrices. Hiervoor voorziet MATLAB de nodige operatoren, '.*', './' en '.^' respectievelijk.

```
>> A .* C
ans =
     6    20    36
     9    18    27
>> A ./ C
ans =
    0.1667    0.8000    2.2500
    1.0000    4.5000   27.0000
>> A.^2
ans =
     1    16    81
     9    81   729
```

Merk op dat de resultaten uiteraard verschillen bij gebruik van de operatoren '*' en '.*', immers:

```
>> E1 = [1  2; 3  4]; E2 = [5  6; 7  8];
>> E1*E2
ans =
    19    22
    43    50
>> E1.*E2
ans =
     5    12
    21    32
```

Dit geldt uiteraard ook voor '^' en '.^':

```
>> E1^3
ans =
```

```
    37     54
    81    118
>> E1.^3
ans =
     1      8
    27     64
```

Voor een vierkante matrix kunnen we de determinant uitrekenen met behulp van de functie **det**:

```
>> det(B)
ans =
     0
>> det(D)
ans =
    -3
```

Van een vierkante matrix waarvan de determinant niet nul is kunnen we de inverse matrix berekenen:

```
>> inv(B);
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.541976e-018.
>> inv(D)
ans =
   -0.6667    1.6667
    1.0000   -2.0000
```

De functies '**min**', '**max**', '**sum**' en '**prod**' die besproken werden in sectie 3.2.2 in het kader van vectoren zijn ook gedefinieerd voor matrices, zij werken echter kolomsgewijs.

```
>> min([1  4;
        3  2])
ans =
     1     2
>> sum([1  4;
        3  2])
ans =
     4     6
```

Net zoals bij vectoren werken ook voor matrices de meeste functies componentsgewijs:

```
>> H = [[1  10  100; 1000  10000  100000]]
H =
           1          10         100
        1000       10000      100000
>> log10(H)
ans =
     0     1     2
     3     4     5
```

3.3.3 Eigenwaarden en -vectoren

MATLAB heeft een functie '**eig**' die zowel eigenwaarden als -vectoren van een matrix kan berekenen. Beschouw de volgende matrix als voorbeeld.

```
>> A = [1 -3 2; 0 -2 -1; -1 0 2]
A =
     1    -3     2
     0    -2    -1
    -1     0     2
```

De eigenwaarden kunnen als volgt berekend worden:

```
>> eig(A)
ans =
   1.5974 + 1.5685i
   1.5974 - 1.5685i
  -2.1948
```

De matrix 'A' heeft drie eigenwaarden, twee ervan zijn elkaars complex toegevoegde: $1.5974 + 1.5685i$ en $1.5974 - 1.5685i$; daarnaast is er één reële eigenwaarde: -2.1948.
Dezelfde functie kan echter ook gebruikt worden om de bijbehorende eigenvectoren te berekenen:

```
>> [V, D] = eig(A)
V =
   0.8433                  0.8433                  0.6257
  -0.0816 - 0.1046i       -0.0816 + 0.1046i       0.7657
   0.1295 + 0.5044i        0.1295 - 0.5044i       0.1492

D =
   1.5974 + 1.5685i            0                       0
        0                  1.5974 - 1.5685i            0
        0                       0                  -2.1948
```

De matrix 'V' bevat de eigenvectoren in de kolommen, de matrix 'D' heeft de bijbehorende eigenwaarde op de diagonaal. Ter controle berekenen we $A \cdot \vec{v}_1 - \lambda_1 \vec{v}_1$:

```
>> A*V(:,1) - D(1,1)*V(:,1)
ans =
  1.0e-014 *
  -0.1332 + 0.0888i
   0.0028
   0.1110 - 0.0222i
```

Dit is inderdaad gelijk aan de nulvector binnen de gebruikte precisie; analoog voor de andere twee eigenwaarden en -vectoren.

3.4 Functies

Zoals duidelijk geworden is in de vorige secties bevat MATLAB een zeer uitgebreide bibliotheek van functies. Voorbeelden hiervan zijn o.a. de vierkantswortel '**sqrt**', de trigoniometrische functies '**sin**', '**cos**', '**tan**', hun inverse '**asin**', '**acos**' en '**atan**', de exponentiële functie '**exp**' en de natuurlijke logaritme '**log**', de logaritme met basis 2 en 10, '**log2**' en '**log10**' respectievelijk. Zoals reeds vermeld in secties 3.2.2 en 3.3.2 werken deze functies niet enkel op getallen, maar ook op vectoren en matrices. In deze secties stonden ook diverse voorbeelden van functies die typisch zijn voor het gebruik in de context van vectoren en matrices; hiervan zullen in de volgende hoofdstukken nog een groot aantal voorbeelden gegeven worden. Al deze functies staan uitgebreid gedocumenteerd in help systeem van MATLAB.

Vaak echter is het nuttig zelf nieuwe functies te definiëren die niet voorzien zijn in MATLAB, hetgeen uiteraard voorzien is. Het definiëren van functies in MATLAB gebeurt op een op het eerste zicht nogal bizarre manier. Er zijn twee mogelijkheden die in de volgende subsecties toegelicht worden. De resultaten van functies gedefinieerd volgens beide methoden zijn uiteraard dezelfde, de keuze hangt enkel af van de omstandigheden.

3.4.1 Inline functies

De eerste methode is handig als men functies wil definiëren die men enkel in de huidige sessie zal gebruiken. Voor deze methode zijn de functie-aanroepen ook het meest natuurlijk. Eerst wordt de functie gedefinieerd:

```
>> g = inline('exp(-0.5*x.^2)/sqrt(2*pi)')
g =
     Inline function:
     g(x) = exp(-0.5*x*x)/sqrt(2*pi)
```

De functie kan geëvalueerd worden zoals elke gewone MATLAB functie:

```
>> g(0.0)
ans =
    0.3989
>> g(1.0)
ans =
    0.2420
```

Merk op dat de componentsgewijze operator '.^' gebruikt werd in de definitie van 'gaussian'. Hoewel de functie de juiste resultaten zou opleveren met de operator '^' in situaties zoals hierboven — i.e. wanneer het argument een scalaire grootheid is — zou er een foutmelding optreden indien we de functie in vectorcontext zouden gebruiken:

```
>> g2 = inline('exp(-0.5*x^2)/sqrt(2*pi)');
>> g2(0.0)
ans =
    0.3989
>> g2([0.0 1.0 2.0])
??? Error using ==> inlineeval
Error in inline expression ==> exp(-0.5*x^2)/sqrt(2*pi)
??? Error using ==> ^
Matrix must be square.

Error in ==> C:\MATLAB6p5\toolbox\matlab\funfun\@inline\subsref.m
On line 25  ==>      INLINE_OUT_ = inlineeval(INLINE_INPUTS_,
                     INLINE_OBJ_.inputExpr, INLINE_OBJ_.expr);
```

Daarentegen is dit voor de functie 'g' gedefinieerd met de operator '.^' geen probleem:

```
>> g([0.0 1.0 2.0])
ans =
    0.3989    0.2420    0.0540
```

Met behulp van deze methode kan men ook functies met meerdere argumenten definiëren. Men moet eenvoudig alle namen van argumenten opsommen in de definitie:

```
>> d = inline('norm(x2 - x1)', 'x1', 'x2')
d =
```

```
    Inline function:
    d(x1,x2) = norm(x2 - x1)
```

De aanroep is weer eenvoudig:

```
>> d([0 0], [3 4])
ans =
     5
>> d([1 2 3], [3 4 5])
ans =
    3.4641
```

3.4.2 M-files

De tweede manier bestaat erin een bestand aan te maken — een zogenaamde M-file — met daarin de definitie van de functie. Deze methode is vooral handig wanneer men functies wil definiëren die men vaak nodig heeft. Hoe een M-file te creëren werd besproken in sectie 2.4. Hieronder staat de inhoud van de M-file '`gaussian.m`' die de functie 'gaussian' definieert:

```
function y = gaussian(x)
  y = exp(-0.5*x.^2)/sqrt(2*pi);
```

De eerste regel is de functie declaratie, wat aangegeven wordt door '**function**'. De naam van die functie is in dit geval 'gaussian'. De functie heeft één argument met als naam 'x'. Het resultaat van de berekening zal toegekend worden aan de variabele 'y'. De definitie van de functie — i.e. hetgeen berekend wordt — staat op de tweede regel. De waarde wordt berekend met de variabele 'x' en toegekend aan de variabele 'y' overeenkomstig de functiedeclaratie.

Deze functie kan in MATLAB gebruikt worden zoals een functie die inline gedefinieerd was:

```
> gaussian(0.0)
ans =
    0.3989
>> guassian(1.0)
ans =
    0.2420
```

De volgende M-file '`distance.m`' definieert een functie met twee vectoren als argumenten heeft en de afstand tussen de punten berekent voorgesteld door deze vectoren:

```
function d = distance(x1, x2)
  d = norm(x2 - x1);
```

De declaratie vermeldt dat de functie met naam 'distance' twee argumenten heeft, namelijk 'x1' en 'x2' en dat het resultaat toegekend zal worden aan de variabele 'd'. De aanroep van de functie wordt geïllustreerd in het volgende stukje code waarbij de eerste aanroep vectoren van lengte 2, de tweede die van lengte 3 gebruikt:

```
>> distance([0 0], [3 4])
ans =
     5
>> distance([1 2 3], [3 4 5])
ans =
    3.4641
```

Merk op dat je slechts één functie per M-file kan definiëren en dat deze dezelfde naam heeft als het bestand. Je kan een functie op deze manier ook *enkel* definiëren in een bestand, *niet* in het '`Command Window`'.

Het is belangrijk in te zien dat functie zichzelf kunnen oproepen, met andere woorden, MATLAB laat recursieve functiedefinities toe. Een voorbeeld hiervan is de functie 'ggd' die de grootste gemene deler van twee natuurlijke getallen berekent:

```
function value = ggd(m, n)
    if m == n
        value = m;
    elseif m > n
        value = ggd(m - n, n);
    else
        value = ggd(m, n - m);
    end
```

De constructie '**if** ... **elseif** ... **else** ... **end**' wordt toegelicht in sectie 4.1.

3.5 2D plots

MATLAB heeft erg veel mogelijkheden voor data visualisatie. Diverse types van plots zijn beschikbaar, hier bespreken we er slechts enkele van.

3.5.1 Functie plots

Het is erg eenvoudig plots van functies te maken met behulp van MATLAB. Als voorbeeld wordt de M-file functie 'gaussian' uit sectie 3.4.2 gebruikt.

```
>> fplot(@gaussian, [-5.0, 5.0])
```

We gebruiken hier een zogenaamde function handle '@gaussian' opdat MATLAB de juiste argumenten kan bepalen.

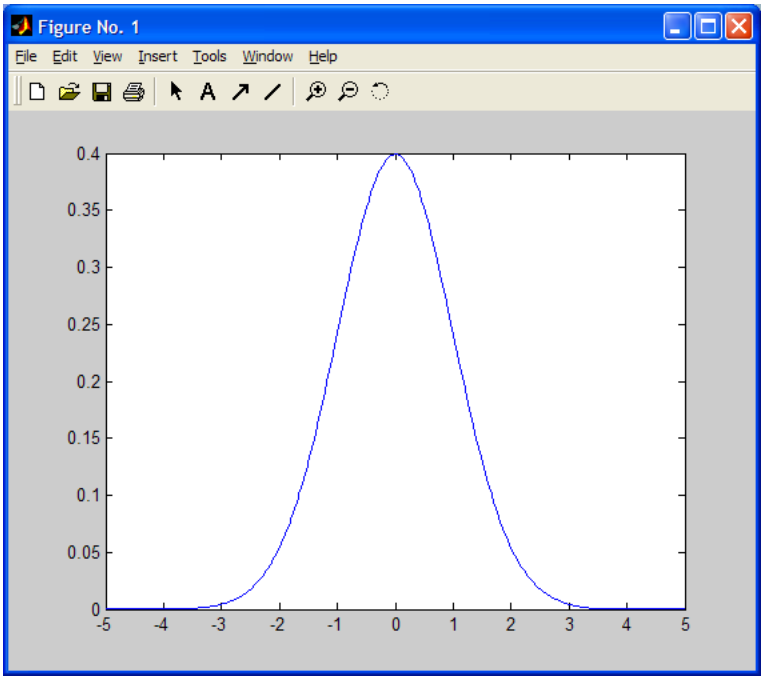
De plot verschijnt in het venster getoond in figuur 3.1. Er zijn een groot aantal mogelijkheden om het uitzicht van de figuur aan te passen. De stijl (kleur, iijntype,...) van de plot lijnen kan aangepast worden, er kunnen labels op de assen toegevoegd worden, een deel van de figuur kan vergroot worden om meer details te zien, de figuur kan geannoteerd worden met tekst, lijnen en pijlen. Kortom, de mogelijkheden zijn bijna eindeloos, men kan best wat experimenteren om de nuttigste te leren kennen. De figuur kan met copy/paste in andere software zoals Microsoft Word ingevoerd worden, maar men kan ze ook in diverse formaten opslaan.

Het is eenvoudig om meerdere functies op één figuur te zetten met behulp van de parameter '**hold**'. Als deze de waarde 'off' heeft (default) dan overschrijft elk plot commando de vorige uitvoer in het venster. De waarde op 'on' zetten daarentegen laat toe in hetzelfde venster te blijven werken.

```
>> fplot(@sin, [-3*pi, 3*pi])
>> hold on
>> fplot(@cos, [-3*pi, 3*pi], 'r--')
>> hold off
```

Hierbij geeft de string ''r--'' aan dat de plot van de cosinus in het rood 'r' en met een gebroken lijn '--' getoond moet worden. Het resultaat is te zien in figuur 3.2. Vergeet niet telkens '**hold**' weer op 'off' te zetten zodra je niet meer in hetzelfde venster wil werken.

De '**hold**' waarde werkt voor elk soort plot, ook in drie dimensies; een voorbeeld hiervan is te zien in figuur 3.8.



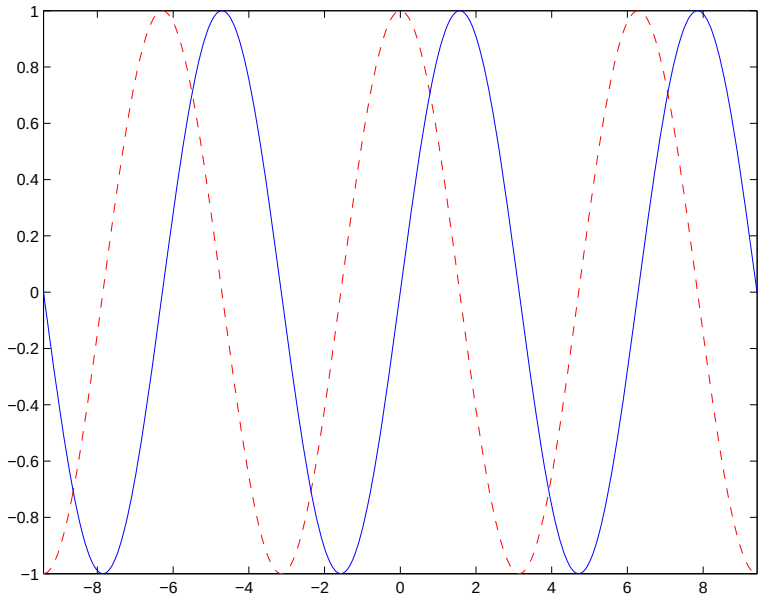
Figuur 3.1: MATLAB Figure venster

3.5.2 Data plots

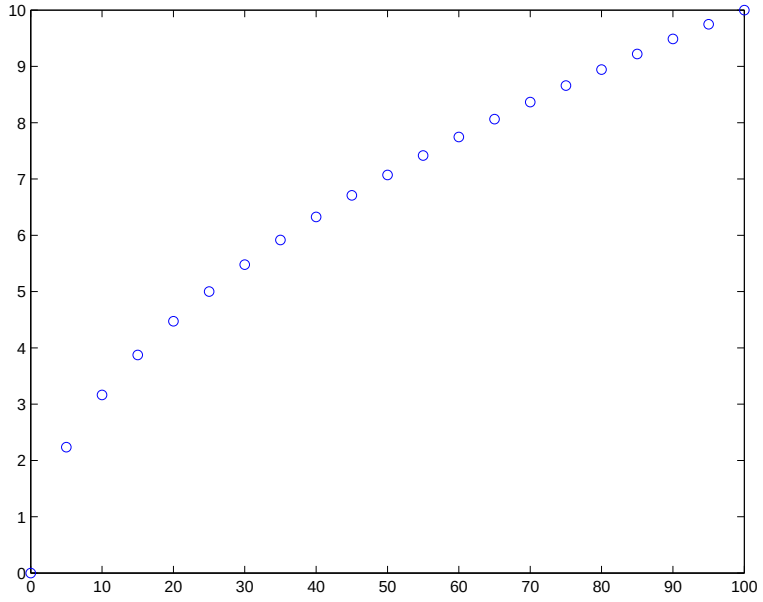
Vaak wordt de data die gevisualiseerd moet worden niet beschreven door een functie, maar is ze het resultaat van een berekening of een experimentele meting. Ook hiervoor biedt MATLAB erg veel mogelijkheden. Het volgende voorbeeld illustreert hoe x-waarden tegen y-waarden uitgezet kunnen worden.

```
>> x = 0.0 : 5.0 : 100.0;
>> y = sqrt(x);
>> plot(x, y, 'o')
```

De parameter ''o'' zorgt ervoor dat elk datapunt door een cirkeltje voorgesteld wordt en er geen lijn door de punten getekend wordt.



Figuur 3.2: meerdere functies op dezelfde plot: $\sin x$ (volle lijn), $\cos x$ (gebroken lijn)



Figuur 3.3: Data plot

3.6 3D plots

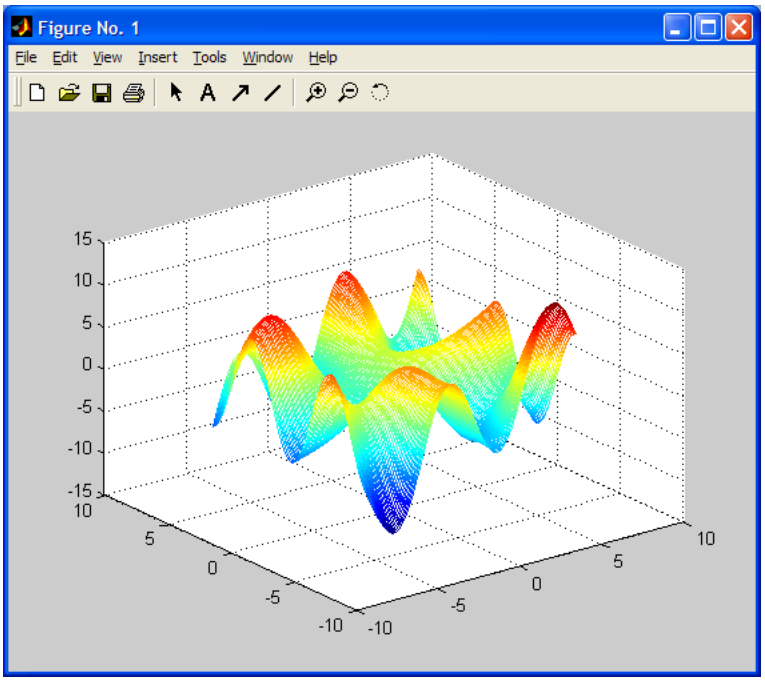
Net zoals voor 2D plots zijn er ook erg veel verschillende types 3D plots. Het zou ons te ver voeren ze allemaal te bespreken, dus ook hier beperken we ons tot een aantal vaak gebruikte soorten.

3.6.1 Oppervlakteplots

Beschouw de functie $f(x, y) = y \sin x + x \cos 2y$, hiervan kunnen we een figuur maken als volgt:

```
>> V1 = -2*pi : 0.1 : 2*pi;
>> V2 = -3*pi : 0.1 : 3*pi;
>> [X, Y] = meshgrid(V1, V2);
>> Z = sin(X).*Y + X.*cos(Y);
>> mesh(X, Y, Z)
```

Het resultaat wordt getoond in figuur 3.4. De functie '**meshgrid**' berekent aan de hand van de vectoren 'V1' en 'V2' twee vierkante matrices 'X' en 'Y' die de componenten van 'V1', respectievelijk 'V2' als rijen dan wel als kolommen bevatten. Met behulp van 'X' en 'Y' kunnen we de vierkante matrix 'Z' berekenen met behulp van componentsgewijs werkende operatoren. Deze matrix bevat nu de waarden $f(x, y)$ en kan getoond worden met behulp van de functie '**mesh**'. In de MATLAB GUI is het erg eenvoudig aspecten van de plot aan te passen of deze te roteren in drie dimensies.



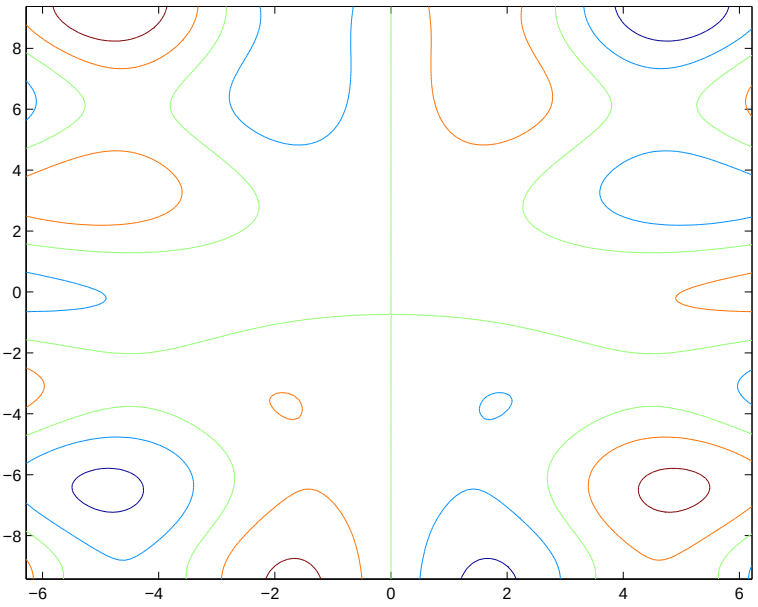
Figuur 3.4: 3D plot van de functie $f(x, y) = y \sin x + x \cos 2y$

3.6.2 Contourplots

Een ander type plot dat vaak van pas komt voor drie dimensionale data — en soms gemakkelijker te analyseren is dan een 3D plot — is een contourplot. De onderstaande code genereert het resultaat getoond in figuur 3.5.

```
>> contour(X, Y, Z)
```

De matrices ‘X’, ‘Y’ en ‘Z’ zijn dezelfde die gebruikt werden voor ‘**mesh**’. Het aantal contourlijnen kan meegegeven worden als optionele parameter (e.g. **contour**(X, Y, Z, 20)’ zal 20 contourlijnen gebruiken).



Figuur 3.5: contourplot van de functie $f(x, y) = y \sin x + x \cos y$

Het is ook mogelijk beide types plot te combineren, een voorbeeld hiervan is te zien in figuur 3.11 waarbij opnieuw gebruik gemaakt werd van de ‘**hold**’ variabele.

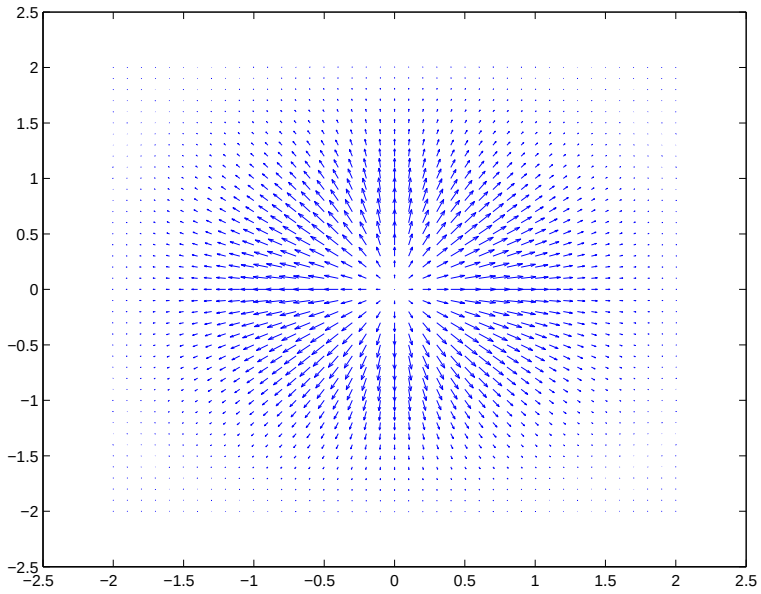
3.6.3 Vectorveldplot

MATLAB beschikt over een derde type plot, ‘**quiver**’ dat nuttig is voor het visualiseren van vectorvelden. Dit kan geïllustreerd worden aan de hand van het vectorveld:

$$\vec{F}(x, y) = (\vec{e}_x \sin x + \vec{e}_y \sin y)e^{-(x^2+y^2)}$$

Dit veld wordt getoond in figuur 3.6.

```
>> [X, Y] = meshgrid(-2.0 : 0.1 : 2.0);
>> U = sin(X).*exp(-(X.^2 + Y.^2));
>> V = sin(Y).*exp(-(X.^2 + Y.^2));
>> quiver(X, Y, U, V)
```



Figuur 3.6: vectorvelfd plot van $\vec{F}(x, y) = (\vec{e}_x \sin x + \vec{e}_y \sin y) \exp(-(x^2 + y^2))$

3.7 Differentiatie

MATLAB heeft geen functie die de afgeleide van een functie in een punt berekent — dit is trouwens niet triviaal — maar het is vrij eenvoudig om een figuur van de afgeleide functie te plotten, ook al is hier de nodige voorzichtigheid geboden. Wanneer men de afgeleide wil tekenen van de functie '**sin**' in het interval $[0, \pi]$ dan kan men als volgt te werk gaan. Definieer een (groot) aantal x-waarden en bijbehorende functiewaarden:

```
>> x = 0.0 : 0.01 : pi;
>> y = sin(x);
```

Nu kan men gebruik maken van de MATLAB functie '**diff**' die het verschil tussen opeenvolgende elementen van een vector berekent:

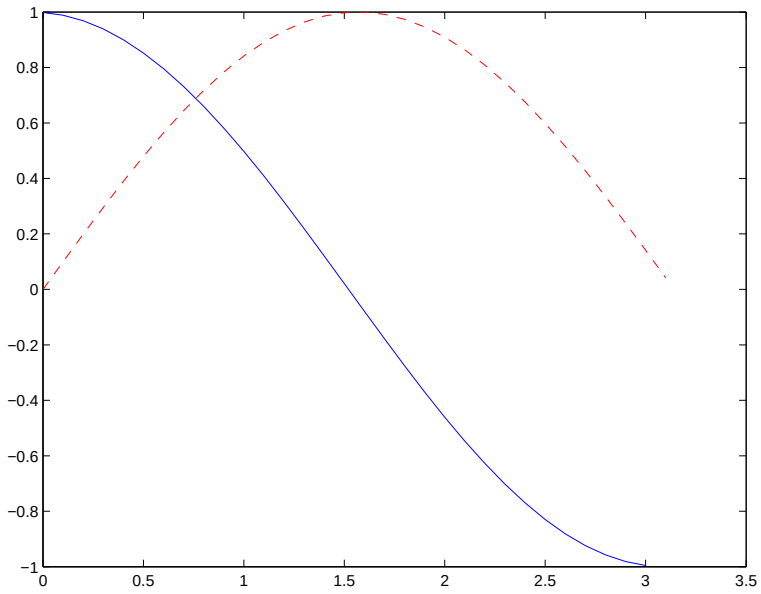
```
>> dy = diff(y)./diff(x);
```

Deze vector 'dy' heeft één component minder dan de vector 'y' zodat we voor een plot ook enkel de eerste `length(x)` − 1 componenten van de vector 'x' nodig hebben:

```
>> dx = x(1:length(x) - 1);
```

Figuur 3.7 toont de plot gemaakt met behulp van '**plot**(dx, dy)'.

Uiteraard zal de figuur nauwkeuriger worden naarmate er meer punten gebruikt worden voor de berekening ervan.



Figuur 3.7: afgeleide functie (volle lijn) van de functie '**sin**' (gebroken lijn) in het interval $[0, \pi]$

3.8 Gradiënt, divergentie en rotor

Voor calculus in meerdere veranderlijken ondersteunt MATLAB de functies voor het berekenen van de gradiënt van een functie ('**gradient**'), de divergentie ('divergence') en de rotor ('curl') van een vectorveld. De nabla-operator is gedefinieerd als

$$\vec{\nabla} = \frac{\partial}{\partial x}\vec{e}_x + \frac{\partial}{\partial y}\vec{e}_y + \frac{\partial}{\partial z}\vec{e}_z$$

3.8.1 Gradiënt

De gradiënt van een functie $f(x, y, z)$ is gedefinieerd als volgt:

$$\vec{\nabla} f = \frac{\partial f}{\partial x}\vec{e}_x + \frac{\partial f}{\partial y}\vec{e}_y + \frac{\partial f}{\partial z}\vec{e}_z$$

Voor het berekenen van de gradiënt van een functie in twee veranderlijken worden eerst de matrices 'X' en 'Y' berekend die de x- en y-coördinaten bevatten waarin de functie geëvalueerd zal worden.

```
>> [X, Y] = meshgrid(0.0 : 0.1 : pi);
```

Vervolgens kan de matrix 'Z' berekend worden met de functiewaarden:

```
>> Z = sin(X).*Y + X.*cos(Y);
```

Tenslotte bekomen we de gradiënt gebruik makend van de matrix 'Z' waarbij de afstand tussen de punten in elke richting specificeren:

```
>> [DX, DY] = gradient(Z, 0.1, 0.1);
```

Het resulterende vectorveld wordt gevisualiseerd in figuur 3.6.

De gradiënt van functies in meer dan twee veranderlijken wordt analoog berekend.

3.8.2 Divergentie

De divergentie van een vectorveld $\vec{F} = F_x\vec{e}_x + F_y\vec{e}_y + F_z\vec{e}_z$ wordt gegeven door:

$$\vec{\nabla} \cdot \vec{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$$

en is eenvoudig te berekenen in MATLAB. Beschouw als voorbeeld het veld gegeven door $\vec{F}(x, y) = \vec{e}_x \cos x + \vec{e}_y \sin y$. Eerst creëren we de matrices 'X' en 'Y' waarin we het vectorveld beschouwen en berekenen we de matrices 'U' en 'V' waarin de x- en de y-component van het vectorveld $\vec{F}$ opslaan:

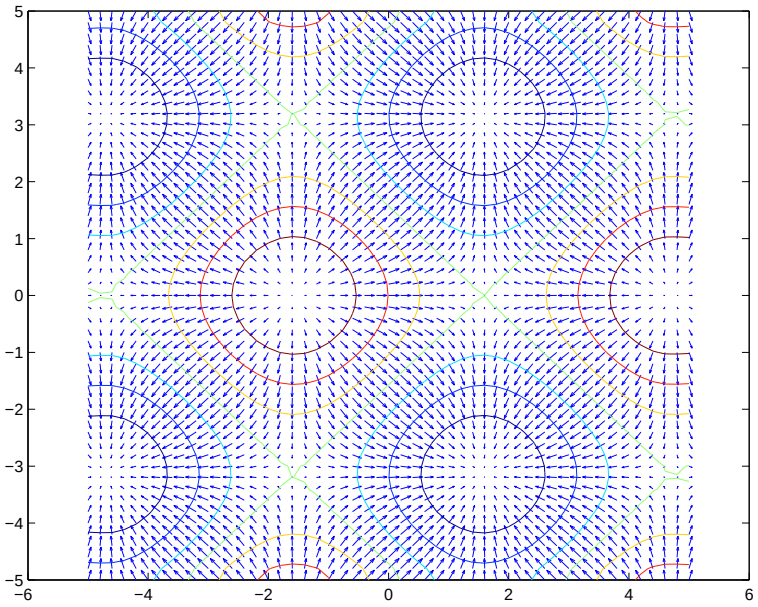
```
>> [X, Y] = meshgrid(-5.0 : 0.2 : 5.0);
>> U = cos(X);
>> V = sin(Y);
```

Dit vectorveld wordt getoond in figuur 3.8 (vectorpijlen). De divergentie $\vec{\nabla} \cdot \vec{F}$ wordt nu gegeven door:

```
>> div = divergence(X, Y, U, V);
```

Deze functie wordt gevisualiseerd in figuur 3.8 (contourlijnen). Deze figuur werd gecreëerd met behulp van:

```
>> quiver(X, Y, U, V)
>> hold on
>> contour(X, Y, div)
>> hold off
```



Figuur 3.8: vectorveld $\vec{e}_x \cos x + \vec{e}_y \sin y$ en de divergentie ervan (contourlijnen)

3.8.3 Rotor

De rotor van een vectorveld $\vec{F} = F_x\vec{e}_x + F_y\vec{e}_y + F_z\vec{e}_z$ is op de volgende manier gedefinieerd:

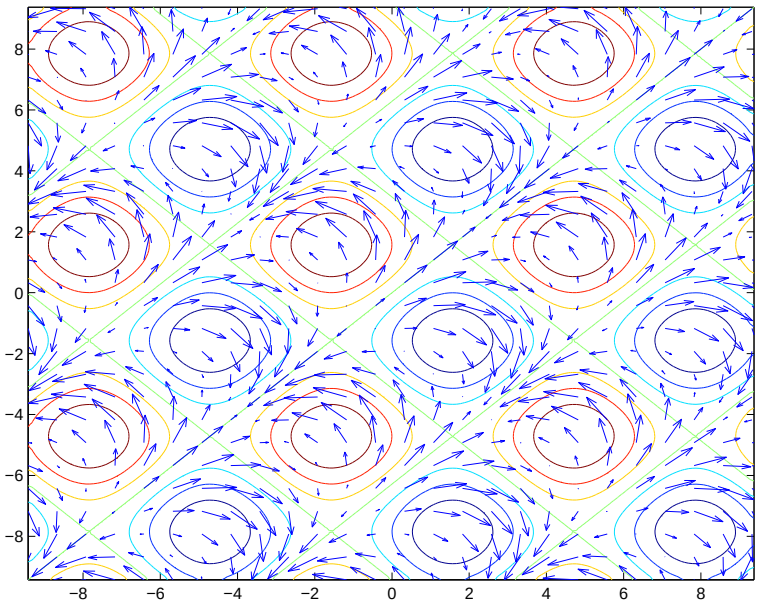
$$\vec{\nabla} \times \vec{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right)\vec{e}_x - \left(\frac{\partial F_z}{\partial x} - \frac{\partial F_x}{\partial z}\right)\vec{e}_y + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\vec{e}_z$$

De rotor van een vectorveld kan berekend worden met behulp van de functie 'curl'. Beschouw bij wijze van voorbeeld het vectorveld gegeven door $\vec{F}(x,y,z) = (\sin x + \cos y)\vec{e}_x + (\cos x + siny)\vec{e}_y$. Zoals bij de divergent moeten we ook hier weer twee matrices 'X' en 'Y' construeren die de x- en y-coördinaten bevatten van de punten waarin we het vectorveld definiëren en de rotor berekenen. De matrices 'U' en 'V' bevatten de x- en y-componenten van het vectorveld, dus respectievelijk $\sin x + \cos y$ en $\cos x + \sin y$. Aangezien het vectorveld dat we in dit voorbeeld beschouwen twee-dimensionaal is, is de rotor een vectorveld met uitsluitend een z-component: $\sin y - \sin x$ aangezien $\vec{\nabla} \times \vec{F} = (\sin y - \sin x)\vec{e}_z$. Deze wordt opgeslagen in de matrix 'cZ'. De onderstaande code produceert figuur 3.9

```
>> [X, Y] = meshgrid(-3*pi : 0.1 : 3*pi);
>> U = sin(X) + cos(Y);
>> V = cos(X) + sin(Y);
>> [cZ, cav] = curl(U, V);
>> contour(X, Y, cZ)
>> hold on
>> [X, Y] = meshgrid(-3*pi : 0.75 : 3*pi);
>> U = sin(X) + cos(Y);
>> V = cos(X) + sin(Y);
>> quiver(X, Y, U, V)
>> hold off
```

Voor een algemeen drie-dimensionaal vectorveld gebruikt men

```
>> [cX, cY, cZ] = curl(X, Y, Z, U, V, W);
```



Figuur 3.9: de z-component van de rotor (contourlijnen) van het vectorveld $(\sin x + \cos y)\vec{e}_x + (\cos x + siny)\vec{e}_y$

3.9 Integratie

Net zoals bij het differentiëren van functies kan de integratie ook enkel numeriek uitgevoerd worden. Er kunnen dus enkel bepaalde integralen berekend worden.

3.9.1 Enkelvoudige integralen

Enkel bepaalde integralen kunnen berekend worden; in de literatuur wordt dit ook vaak quadratuur genoemd, en dat is ook de terminologie in MATLAB (quadrature). MATLAB heeft een functie — '**quad**' — om een bepaalde integraal van een functie te berekenen met behulp van het algoritme van Simpson. Het gebruik ervan wordt geïllustreerd aan de hand van het volgende voorbeeld:

```
>> quad(@sin, 0.0, pi)
ans =
    1.99999999639843
```

Aangezien de functie numeriek berekend wordt is de bekomen waarde uiteraard slechts een benadering. De quadratuur functies hebben dan ook een optionele parameter die deze nauwkeurigheid bepaald. De standaardwaarde is 10^{-6}, hetgeen betekent dat het resultaat nauwkeurig is tot op vijf beduidende cijfers. In de praktijk ligt de precisie vaak hoger zoals te zien is in het bovenstaande voorbeeld. Is een grotere nauwkeurigheid gewenst, dan kan men deze waarde als volgt aanpassen:

```
>> quad(@sin, 0.0, pi, 1.0e-10)
ans =
    1.99999999999987
```

Merk op dat MATLAB nog een tweede quadratuurfunctie heeft, 'quadl' die het adaptive Lobatto algoritme gebruikt.

3.9.2 Meervoudige integralen

Er zijn ook functies voorzien voor het berekenen van dubbele en drievoudige integralen, 'dblquad' en 'triplequad'. 'dblquad' wordt als volgt gebruikt:

```
>> f = inline('sin(x)*y + x*cos(y)', 'x', 'y')
f =
     Inline function:
     f(x,y) = sin(x)*y + x*cos(y)
>> dblquad(f, 0.0, 2*pi, 0.0, 2*pi)
ans =
 -4.7607e-010
```

3.10 Minimalisatie

Een vaak voorkomend probleem is het zoeken van het minimum van een functie. MATLAB maakt dit relatief eenvoudig met behulp van de functies 'fminbnd' voor één en 'fminsearch' voor meerdere veranderlijken.

3.10.1 Functies in één veranderlijke

De naam geeft aan dat het minimum van de functie berekend zal worden in een door de gebruiker gespecificeerd interval, met andere woorden, er moet van te voren iets bekend zijn over de aard van de functie als men betrouwbare resultaten wil bekomen. Neem als voorbeeld de onderstaande functie waarvoor we het minimum zoeken in het interval $[0.0, 5.0]$:

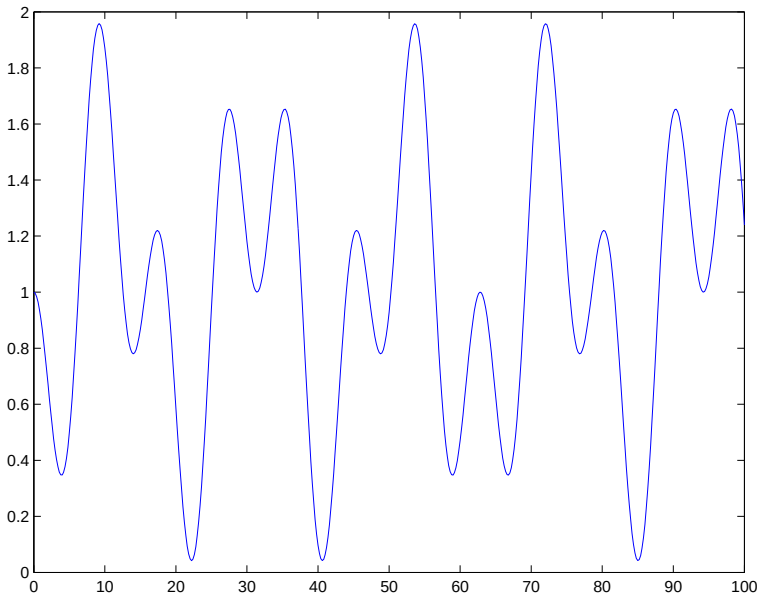
```
>> f = inline('1 - sin(0.2*x).*sin(0.5*x)', 'x');
```

De positie van het minimum wordt gegeven door:

```
>> [x0, val] = fminbnd(f, 0.0, 5.0)
x0 =
    3.9079
val =
    0.3467
```

Hierbij geeft 'x0' de positie en 'val' de waarde van het minimum van de functie 'f'. Figuur 3.10 toont echter dat dit geen globaal minimum is van de functie 'f'. Een globaal minimum — dat overigens niet uniek is — bevindt zich bij 22.2, hetgeen benadrukt dat functie-onderzoek nuttig is.

```
>> [x0, val] = fminbnd(f, 0.0, 50.0)
x0 =
    22.2117
val =
    0.0421
```



Figuur 3.10: De functie $1 - \sin(0.2x)\sin(0.5x)$ heeft een zowel locale als globale minima.

3.10.2 Functies in meerdere veranderlijken

Optimalisatie van functies in meerdere veranderlijken verloopt zeer analoog, hiervoor is de functie 'fminsearch' voorzien. Beschouw bij wijze van voorbeeld de functie:

$$f(x, y) = (x - 1)^2 + (y + 1)^2 + 2$$

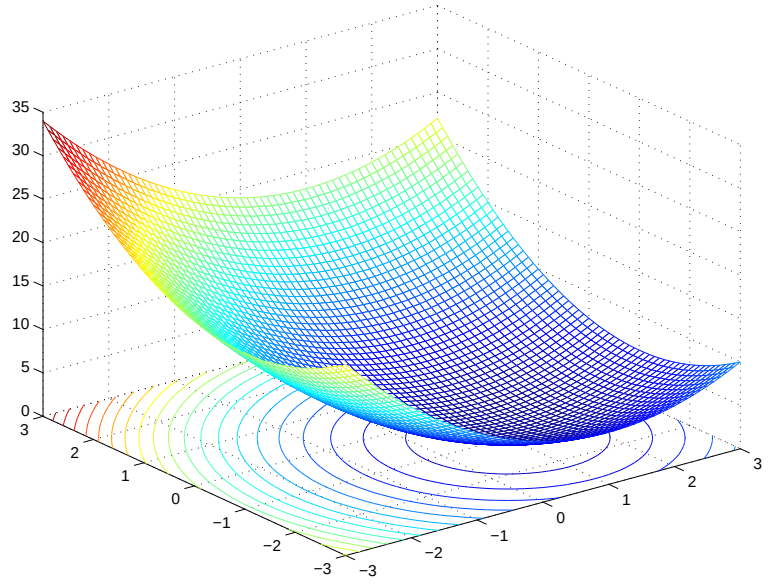
of in MATLAB:

```
>> f = inline('(x(1) - 1)^2 + (x(2) + 1)^2 + 2', 'x');
```

De functie wordt getoond in figuur 3.11 met behulp van de volgende commando's:

```
>> [X, Y] = meshgrid(-3.0 : 0.1 : 3.0);
>> Z = (X - 1).^2 + (Y + 1).^2 + 2;
>> mesh(X, Y, Z)
```

```
>> hold on
>> contour(X, Y, Z, 20)
>> hold off
```



Figuur 3.11: de functie $f(x, y) = (x - 1)^2 + (y + 1)^2 + 2$

Om het minimum van deze functie te zoeken moeten we een startpunt specificeren. De keuze hiervan is kritiek indien de functie meerdere lokale minima heeft. Voor dit zeer eenvoudige voorbeeld kunnen we bijvoorbeeld $(0, 0)$ kiezen.

```
>> [x0, val] = fminsearch(f, [0.0, 0.0])
x0 =
    1.0000   -1.0000
val =
    2.0000
```

Het minimum bevindt zich inderdaad in het punt $(1, -1)$ en heeft de waarde $f(1, -1) = 2$.

3.11 Vergelijkingen

Een ander veel voorkomend probleem is het oplossen van vergelijkingen. Ook hiervoor levert MATLAB de nodige ondersteuning. Twee soorten problemen worden hier behandeld: (1) het oplossen van een vergelijking in één onbekende en (2) het oplossen van stelsels lineaire vergelijkingen.

3.11.1 Vergelijkingen in één onbekende

MATLAB heeft geen functies om een vergelijking rechtstreeks op te lossen, maar wel om nulpunten van functies te zoeken. Uiteraard komt dit op hetzelfde neer, immers:

$$f(x) = g(x) \Leftrightarrow F(x) = 0$$

waar $F(x) = f(x) - g(x)$. Beschouw bij wijze van voorbeeld de volgende niet-lineaire vergelijking:

$$\tanh(2x) = x$$

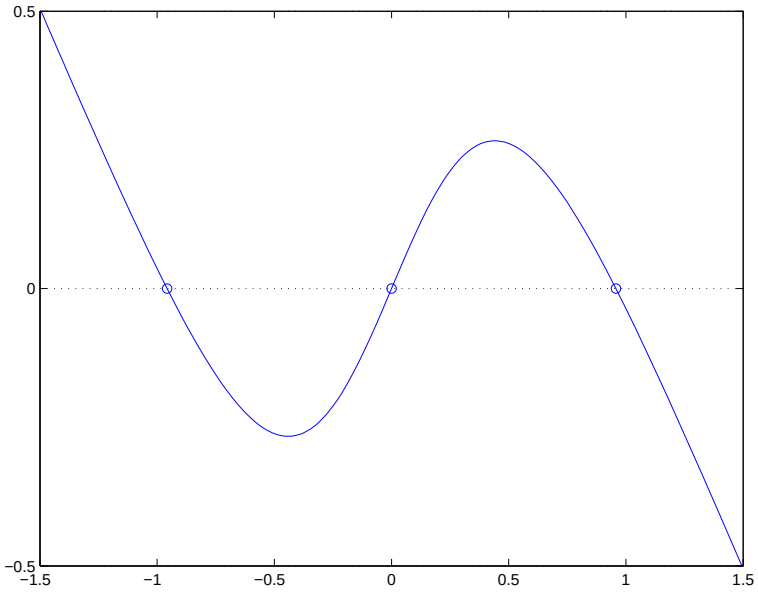
De strikt positieve oplossing kan gevonden worden met behulp van:

```
>> f = inline('tanh(2*x) - x');
>> fzero(f, 2.0)
ans =
    0.9575
```

Hierbij is de waarde 2.0 een startwaarde: de functie 'fsolve' zal bij het zoeken naar een nulpunt vertrekken van deze waarde.
Net zoals bij het zoeken van minima van functies is ook hier voorzichtigheid geboden. Immers, '**fzero**' levert niet steeds het gewenste — of zelfs verwachte — resultaat.

```
>> fzero(f, 0.5)
ans =
  2.3942e-021
```

Hoewel 0 een oplossing is van de vergelijking is het niet degene die gevraagd wordt. Figuur 3.12 toont de drie oplossingen. Ook hier moet men dus het nodige wiskundig inzicht aan de dag leggen.



Figuur 3.12: De functie $\tanh(2x) - x$ heeft drie wortels bij $x \approx -0.96$, $x = 0$ en $x \approx 0.96$.

3.11.2 Stelsels van lineaire vergelijkingen

Veelal wordt de oplossing gezocht van een stelsel van n lineaire vergelijkingen met n onbekenden. Dit is het meest eenvoudige geval, het wordt hier geïllustreerd aan de hand van een voorbeeld. Beschouw het volgende stelsel van vergelijkingen:

$$\begin{aligned} 2x_1 + 3x_2 &= 5 \\ x_1 - x_2 &= 0 \end{aligned}$$

Dit stelsel kan herschreven worden in termen van matrices:

$$A \cdot x = b$$

waarbij de matrix A en de vectoren x en b gegeven worden door:

$$A = \begin{pmatrix} 2 & 3 \\ 1 & -1 \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad b = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$$

De oplossing van dit stelsel wordt in MATLAB als volgt berekend: de matrix 'A' en de vector 'b' worden gedefinieerd,

```
> A = [2  3; 1  -1]
A =
     2      3
     1     -1
>> b = [5; 0]
b =
     5
     0
```

en vervolgens wordt de oplossing berekend.

```
>> x = A\b
x =
     1
     1
```

Het is natuurlijk mogelijk dat de vergelijkingen niet onafhankelijk zijn zoals bijvoorbeeld het volgende stelsel:

$$\begin{aligned} 2x + 3y &= 5 \\ \frac{2}{3}x + y &= 9 \end{aligned}$$

MATLAB detecteert dat de determinant van de matrix A zeer dicht bij nul ligt en deze A dus singulier is.

```
>> A = [2  3; 2/3  1]
A =
    2.0000    3.0000
    0.6667    1.0000
>> b = [5; 9]
b =
     5
     9
>> A\b
Warning: Matrix is singular to working precision.
(Type "warning off MATLAB:singularMatrix" to suppress
      this warning.)
ans =
   Inf
   Inf
```

3.11.3 Stelsels niet-lineaire vergelijkingen

Naast stelsels van lineaire vergelijkingen zoals behandeld in Sectie 3.11.2 zijn er ook veel problemen die aanleiding geven tot een stelsel van vergelijkingen die niet lineair zijn. Beschouw bijvoorbeeld het zoeken van de snijpunten van een cirkel en een rechte: de vergelijking van de cirkel is een tweede graadsvergelijking, dus niet lineair:

$$\begin{aligned} x^2 + (y-1)^2 &= 2 \\ 3x - y &= 1 \end{aligned}$$

De oplossing van dit stelsel kunnen we bekomen met behulp van de functie 'fsolve', die net zoals '**fzero**' de nulpunten zoekt van functies. We definiëren dus eerst een functie die als resultaat een kolomvector heeft met de waarden van de functies:

$$\begin{aligned} f_1(\vec{x}) &= x_1^2 + (x_2 - 1)^2 - 2 \\ f_2(\vec{x}) &= 3x_1 - x_2 - 1 \end{aligned}$$

of in MATLAB:

```
>> F = inline('[x(1).^2+(x(2)-1).^2-2; 3*x(1)-x(2)-1]');
```

Het argument van deze functie moet een vector zijn met twee componenten, bijvoorbeeld:

```
>> F([1; 1])
ans =
    -1
     1
```

Analoog aan '**fzero**' wordt aan de functie 'fsolve' een kolomvector met startwaarden als parameter gegeven worden, bovendien verwacht 'fsolve' opties:

```
>> options = optimset('Display', 'notify');
```

Voor een discussie van de mogelijkheden van 'optimset' wordt verwezen naar de MATLAB documentatie, dit zou hier te ver voeren. Het stelsel kan nu opgelost worden met:

```
>> fsolve(F, [2; 2], options)
Optimization terminated successfully:
 First-order optimality is less than options.TolFun.
ans =
    1.0000
    2.0000
```

Het punt $(1, 2)$ is dus een snijpunt van de cirkel en de rechte.

Zoals Figuur 3.13 aangeeft hebben de cirkel en de rechte twee snijpunten, het tweede kunnen we bekomen door andere startwaarden te gebruiken:

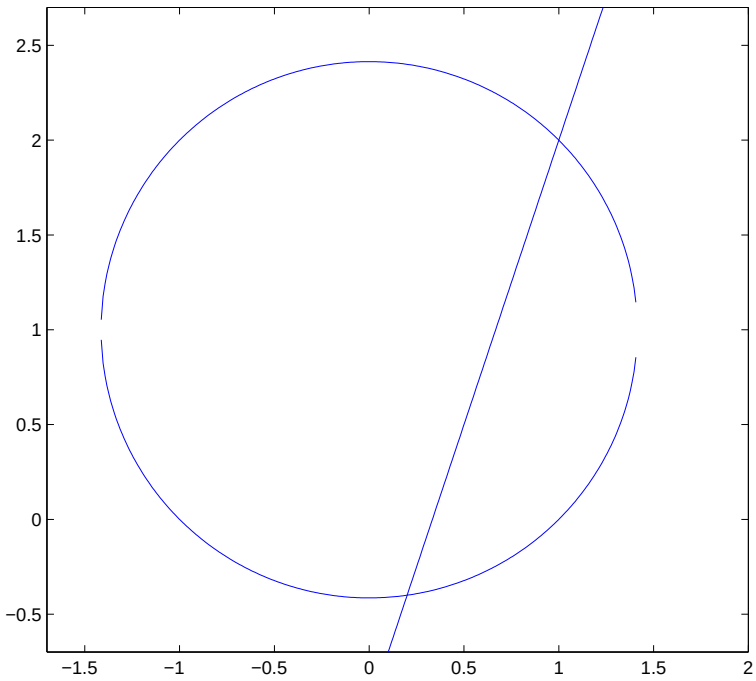
```
>> fsolve(F, [-1; -1], options)
Optimization terminated successfully:
 First-order optimality is less than options.TolFun.
ans =
    0.2000
   -0.4000
```

Het tweede snijpunt is dus $(0.2, -0.4)$.

Merk op dat voorzichtigheid geboden is, beschouw de volgende vergelijking die opnieuw een cirkel en een rechte voorstellen:

$$\begin{aligned} x^2 + y^2 &= 1 \\ x - y &= 3 \end{aligned}$$

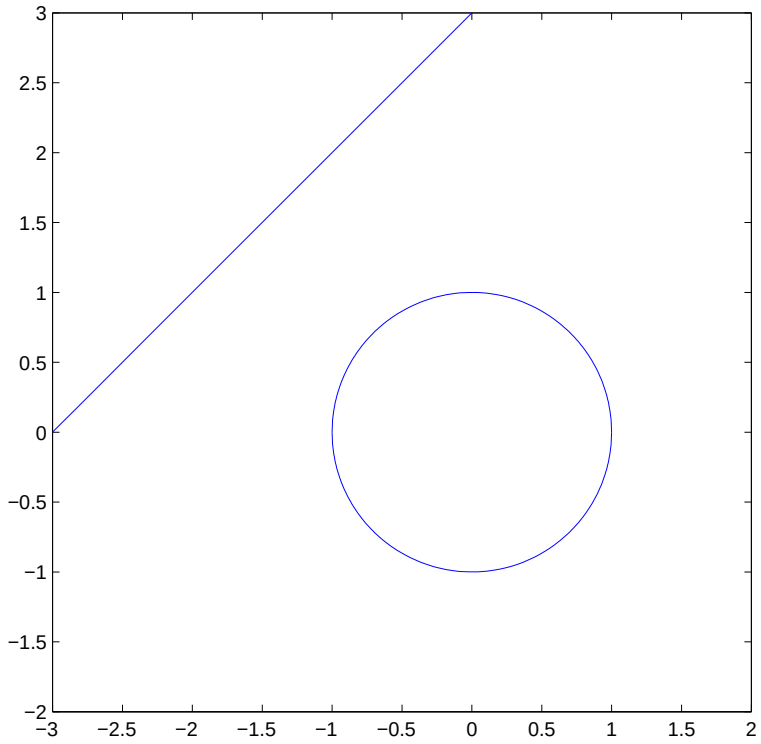
Het is duidelijk uit Figuur 3.14 dat deze cirkel en rechte geen snijpunt hebben. De 'fsolve' geeft nu de volgende waarschuwing:



Figuur 3.13: De cirkel en de rechte met respectievelijk vergelijking $x^2+(y-1)^2 = 2$ en $y = 3x-1$ hebben twee snijpunten.

```
>> F = inline('[(x(1).^2 + x(2).^2 - 1); (x(1) - x(2) + 3)]');
>> options = optimset('Display', 'notify');
>> fsolve(F, [1; 1], options)
Optimizer appears to be converging to a minimum that is not a
root:
Sum of squares of the function values is > sqrt(options.TolFun).
Try again with a new starting point.
ans =
   -0.9086
    0.9086
```

Het resultaat is dus niet betrouwbaar.



Figuur 3.14: De cirkel en de rechte met respectievelijk vergelijking $x^2 + y^2 = 1$ en $y = x + 3$ hebben geen snijpunten.

3.12 Differentiaalvergelijkingen

3.12.1 Eerste orde differentiaalvergelijkingen

Beschouw als voorbeeld de volgende differentiaalvergelijking:

$$\frac{dy(t)}{dt} = -\frac{1}{2}y(t)$$

met $y(0) = 1$ als beginvoorwaarde. Het rechterlid van deze vergelijking wordt als functie 'exponential' gedefinieerd in een M-file:

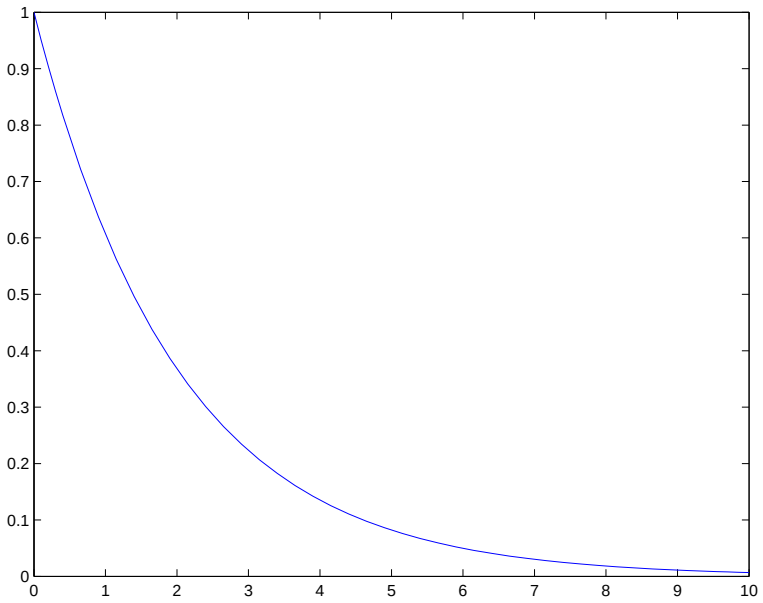
```
function dydt = exponential(t, y)
  dydt = -0.5*y;
end
```

Nu kan de vergelijking opgelost worden met behulp van de functie '**ode45**'.

```
>> [t, y] = ode45(@exponential, [0.0 10.0], [1.0]);
```

Het resultaat van deze functie bestaat uit twee delen, 't' en 'y', het eerste bevat een lijst van parameter waarden, het tweede een lijst met de bijbehorende functiewaarden. De oplossing bestaat dus uit een lijst van functiewaarden. Het is nu eenvoudig een plot te maken van de oplossing (zie figuur 3.15).

```
>> plot(t, y)
```



Figuur 3.15: Oplossing van de differentiaalvergelijking $dy(t)/dt = -y(t)/2$ met beginvoorwaarde $y(0) = 1$

3.12.2 Stelsel van eerste orde differentiaalvergelijkingen

Het oplossen van stelsel eerste orde differentiaalvergelijkingen is een uitbreiding van de methode besproken in de vorige sectie. Beschouw het volgende stelsel vergelijkingen:

$$\begin{aligned} y_1'(t) &= y_1(t)y_2(t) \\ y_2'(t) &= -2y_1(t)y_2(t) \end{aligned}$$

We definiëren de functie 'diffeqs' zodat deze als eerste component het rechterlid van de eerste differentiaalvergelijking en als tweede die van het tweede teruggeeft in de M-file '`diffeqs.m`':

```
function dydt = diffeqs(t, y)
  dydt = [y(1)*y(2); -2*y(1)*y(2)]
end
```

Vervolgens lossen we het stelsel op met beginvoorwaarden $y_1(0) = y_2(0) = 1$ in het interval $[0, 3]$:

```
[t, y] = ode45(@diffeqs, [0.0 3.0], [1.0 1.0]);
```

Het resultaat wordt getoond in figuur 3.16 met behulp van:

```
plot(t, y(:,1), '-', t, y(:,2), 'r--')
```

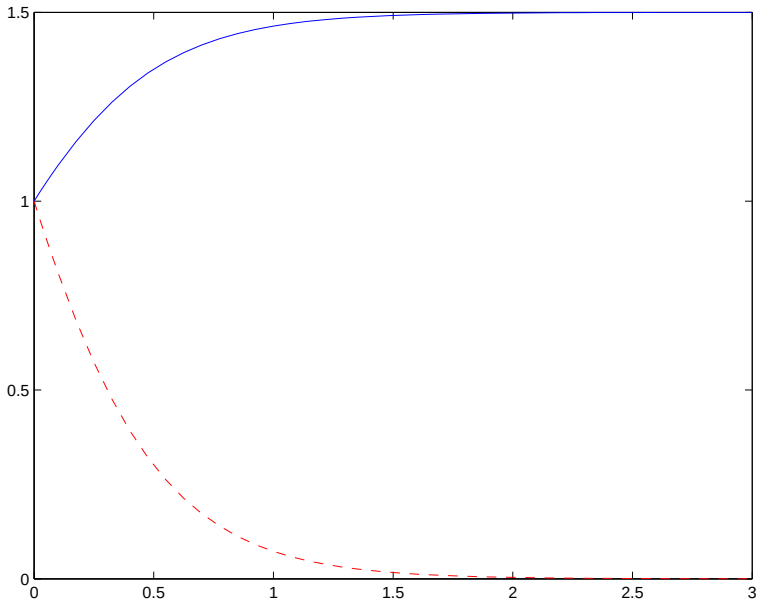
3.12.3 Hogere orde differentiaalvergelijkingen

Oplossen van hogere orde differentiaalvergelijkingen kan jammer genoeg niet rechtstreeks, een dergelijke vergelijking moet eerst omgezet worden in een stelsel eerste orde differentiaalvergelijkingen. Beschouw de volgende differentiaalvergelijking:

$$y_1''(t) + \frac{1}{2}y_1'(t) + y_1(t) = 5$$

met als beginvoorwaarden $y_1'(0) = 0$ en $y_1(0) = 1$. Deze vergelijking kan omgezet worden naar het volgende stelsel:

$$y_2'(t) = -\frac{1}{2}y_2(t) - y_1(t) + 5$$



Figuur 3.16: Oplossing van een stelsel differentiaalvergelijkingen: $y_1(t)$ (volle lijn en $y_2(t)$ (gebroken lijn)

$$y_1'(t) \quad = \quad y_2(t)$$

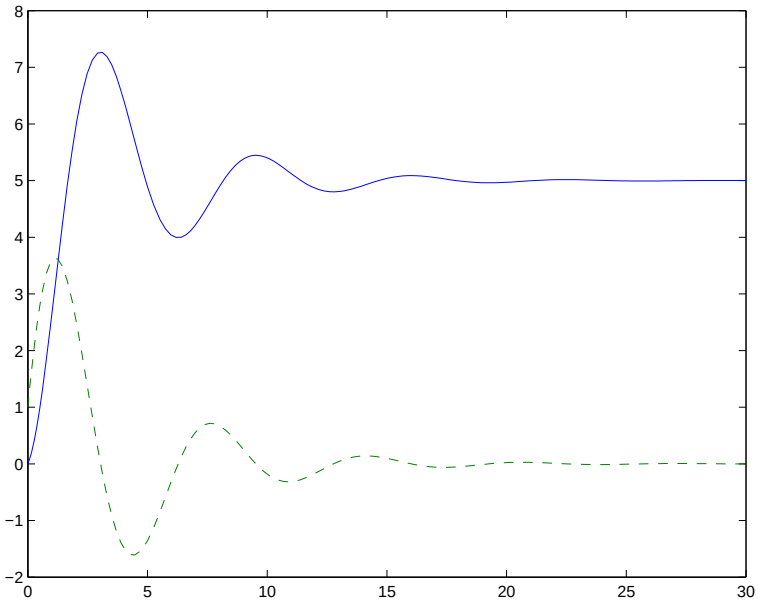
en de beginvoorwaarden $y_1(0) = 1$ en $y_2(0) = 0$. De tweede orde differentiaalvergelijking werd dus omgezet in een stelsel van twee eerste orde differentiaalvergelijkingen. De M-file die de functies definieert ziet eruit als volgt:

```
function dydt = harmonic(t, y)
  dydt = [y(2); -0.5*y(2) - y(1) + 5]
end
```

Het stelsel wordt opgelost en geplot met de volgende functie-aanroepen:

```
>> [t, y] = ode45(@harmonic, [0.0 30.0], [0.0; 1.0]);
>> plot(t, y(:,1), '-', t, y(:,2), 'r--')
```

Het resultaat is te zien in figuur 3.17.



Figuur 3.17: Oplossing $y_1(t)$ (volle lijn) van een differentiaalvergelijking van hogere graad en de eerste afgeleide $y_1'(t)$ (gebroken lijn).

Hoofdstuk 4

Programmeren in MATLAB

MATLAB beschikt voor een eigen programmeertaal die je kan gebruiken om repetitieve taken eenvoudiger te maken. Ze is echter compleet, met andere woorden, ze is even krachtig als e.g. Java of C. Uiteraard is ze vooral en bij uitstek geschikt om problemen op te lossen die te maken hebben met wiskundige modellen, data-analyse, -manipulatie en -visualisatie.

Dit gedeelte van de tekst heeft niet de ambitie alle mogelijkheden van MATLAB als programmeeromgeving te bespreken, integendeel, wat hier behandeld wordt is een minimum om van start te kunnen gaan. Vier concepten zijn belangrijk. Het eerste, het gebruik van variabelen, werd reeds uitgebreid behandeld in sectie 3.1.2. Het tweede en het derde zijn respectievelijk voorwaardelijke en herhalingsopdrachten, hierover vind je meer in secties 4.1 en 4.2. Tenslotte is het noodzakelijk dat deze voorwaardelijke en herhalingsopdrachten willekeurig genest kunnen worden, en dat is geen probleem in MATLAB.

4.1 Voorwaardelijke opdrachten

Beschouw de functie die als resultaat de waarde 1 geeft wanneer haar argument groter is dan 0, de waarde 0 in het andere geval. De M-file die deze functie implementeert ziet er als volgt uit:

```
function value = heaviside(x)
  if  x > 0
      value = 1;
  else
      value = 0
  end
end
```

Een meer alledaags voorbeeld is een functie die, gegeven een jaartal, de waar 1 geeft wanneer dit een schrikkeljaar is, 0 in het andere geval.

```
function value = leapyear(n)
    if rem(n, 100) == 0 & ~rem(n, 400) == 0
        value = 0;
    elseif rem(n, 4) == 0
        value = 1;
    else
        value = 0;
    end
end
```

In deze functie werden twee klassen van operatoren gebruikt die zeer nuttig bij zowel voorwaardelijke als bij bepaalde herhalingsopdrachten. Deze worden in de volgende subsecties besproken.

4.1.1 Relationele operatoren

Naast de evidente relationele operatoren '<', '<=', '>' en '>=' zijn er nog twee operatoren die respectievelijk gelijkheid en ongelijkheid testen: '==' en '~='.

De operanden van een relationele operator moeten met elkaar vergeleken kunnen worden, in MATLAB dus typisch twee uitdrukkingen die getallen als resultaat hebben. Als de vergelijking opgaat dan is het resultaat de logische *waar*, anders de logische *vals*. Uitdrukkingen met relationele operatoren zullen dus vaak een operand zijn van een logische operator.

Merk op dat de toekenningsoperator '=' en de gelijkheid '==' een totaal andere rol spelen. De eerste kent de waarde die rechts staat toe aan de variabele links, de tweede vergelijkt de linker- en de rechteroperand.

4.1.2 Logische operatoren

In de eerste test hebben we gebruik gemaakt van twee logische operatoren: '&' stelt de logische *en*, '~' de logische *niet*. Daarnaast is ook nog de logische *of* gedefinieerd als '|'.

In MATLAB stelt 0 de logische waarde *vals* voor, alles wat niet gelijk is aan 0 heeft de logische waarde *waar*.

Je kan dus eigenlijk willekeurige MATLAB uitdrukkingen die een getal als resultaat hebben gebruiken als operanden van logische operatoren, maar dit is onverstandig omdat het tot erg veel verwarring kan leiden. Het is een goede gewoonte als operanden van een logische operator enkel uitdrukkingen te gebruiken waarvan het resultaat een logische waarde voorstelt zoals bijvoorbeeld relationele uitdrukkingen of andere testen.
Het resultaat van een logische operator is opnieuw een logische waarde.

4.2 Herhalingsopdrachten

Herhalingsopdrachten zijn typisch onder te verdelen in twee categoriën: een (reeks van) opdracht(en) moet een welbepaald aantal keer herhaald worden, of ze moet herhaald worden zolang een bepaald voorwaarde voldaan is. MATLAB voorziet hiervoor twee opdrachten die hieronder besproken worden.

4.2.1 for-lussen

Stel bij wijze van voorbeeld dat we een 5×5-matrix A willen definiëren waarvan de componenten gegeven worden door $A_{ij} = 2^i 3^j$. Het volgende fragment MATLAB-code doet dit:

```
>> for i = 1 : 5
     for j = 1 : 5
       A(i,j) = 2^i * 3^j;
     end
   end
>> A
A =
           6          18          54         162         486
          12          36         108         324         972
          24          72         216         648        1944
          48         144         432        1296        3888
          96         288         864        2592        7776
```

Dit voorbeeld laat meteen ook zien dat for-lussen genest kunnen worden. De volgende code toont dat we de index 'i' van de for-lus met andere waarden dan 1 kunnen verhogen:

```
>> for i = 10 : -2 : 1
     sqrt(i)
   end
ans =
    3.1623
ans =
    2.8284
ans =
    2.4495
ans =
     2
ans =
    1.4142
```

Waarschijnlijk is het reeds duidelijk dat het bovenstaande gebruik van de for-lus een speciaal geval is van de vorm waarbij de index loopt over de componenten van een vector. Dit wordt geïllustreerd door het volgende voorbeeld:

```
>> v
v =
     1     3     7    15    31
>> for x = v
     sqrt(x + 1)
   end
ans =
    1.4142
ans =
     2
ans =
    2.8284
ans =
     4
ans =
    5.6569
```

Hoewel een for-lus een zeer belangrijke opdracht is moet toch opgemerkt worden dat het nut ervan in MATLAB beperkt is ten opzichte van *"klassieke"* programmeertalen. MATLAB is gebaseerd op berekeningen met vectoren en matrices zodat dit soort problemen vaak met de operatoren die daarop gedefinieerd zijn opgelost kunnen worden. Ter illustratie: de matrix A uit het eerste voorbeeld kan eenvoudiger berekend worden door gebruik te maken van het uit-product:

```
>> v1 = (3*ones(1,5)).^(1:5)
v1 =
     3     9    27    81   243
>> v2 = (2*ones(1,5)).^(1:5)
v2 =
     2     4     8    16    32
>> A = v2.' * v1
A =
           6          18          54         162         486
```

```
    12    36   108   324   972
    24    72   216   648  1944
    48   144   432  1296  3888
    96   288   864  2592  7776
```

Het is eenvoudig in te zien dat ook de andere voorbeelden in deze sectie kunnen opgelost worden met vectoroperaties.

Een voorbeeld dat dit nogmaals illustreert is het berekenen van een som van een reeks getallen:

```
>> s = 0;
>> for i = 1 : 10
       s = s + i;
   end
>> s
s =
    55
```

Dit kan natuurlijk veel eenvoudiger: '**sum**$(1:10)$'.

Het is een kwestie van stijl of men een probleem aanpakt met vector- en matrixoperatoren of met for-lussen, waarbij MATLAB-experts en puristen uiteraard voor de eerste benadering zullen opteren. In de praktijk is slechts één ding van belang: *"get the job done"!*

4.2.2 while-lussen

Het tweede type herhalingsopdracht wordt gestuurd door een voorwaarde. De volgende functie berekent de grootste gemene deler van twee natuurlijke getallen met behulp van de methode van Euclides:

```
function value = ggdi(m, n)
    while m ~= n
        if m > n
            m = m - n;
        else
            n = n - m;
        end
    end
    value = m;
end
```

Merk terloops op dat dit de iteratieve versie is van de recursief gedefinieerde functie 'ggd' uit sectie 3.4.2.

Conclusie

Hoewel deze tekst uiterst beperkt is werd toch geprobeerd een overzicht te geven over de verschillende aspecten en mogelijkheden van MATLAB. Het topje van de ijsberg dat hier behandeld werd zal je ervan overtuigd hebben dat MATLAB — mits goed gebruikt — een uiterst handig instrument is voor wiskundig modelleren, data-analyse en visualisatie. De enorme bibliotheek met wiskundige functies enerzijds en het feit dat je zelf functionaliteit kan toevoegen door het schrijven van eenvoudige functies maken MATLAB tot een onmisbaar werktuig in een groot aantal takken van het wetenschappelijk onderzoek en het bedrijfsleven.

Bibliografie

[Fau99] Laurene V. Fausett. *Applied numerical analysis using MATLAB*. Prentice Hall, Upper Saddle River, NJ, 1999.

[Hah02] Brian D. Hahn. *Essential MATLAB for scientists and engineers*. Butterworth, Oxford, UK, 2002.

[HL01] Duane Hanselman and Bruce Littlefield. *Mastering MATLAB 6: a comprehensive tutorial and reference*. Prentice Hall, Upper Saddle River, NJ, 2001.

[HLR02] Brian R. Hunt, Ronald L. Lipsman, and Jonathan M. Rosenberg. *A guide to MATLAB: for beginners and experienced users*. Cambridge University Press, Cambridge, UK, 2002.

[KG02] Abdelwahab Kharab and Ronald E. Guenther. *An introduction to numerical methods: a MATLAB approach*. Chapman & Hall/CRC, Boca Raton, FL, 2002.

[MM02] Wendy L. Martinez and Angel R. Martinez. *Computational statistics handbook with MATLAB*. Chapman & Hall/CRC, Boca Raton, FL, 2002.

[Pal98] William J. Palm, III. *Introduction to MATLAB for engineers*. Basic Engineering Series and Tools. McGraw-Hill, Boston, MA, 1998.

[Whi04] Robert E. White. *Computational mathematics: models, methods, and analysis with MATLAB and MPI*. Chapman & Hall/CRC, Boca Raton, FL, 2004.

[WTH03] Howard B. Wilson, Louis H. Turcotte, and David Halpern. *Advanced mathematics and mechanics applications using MATLAB*. CRC Press, Boca Raton, FL, 2003.