

Practicum verslag voor Matlab Practicum 2003

Sebastiaan
Willem van Hillegaersbergstraat 100c
3051 RP Rotterdam
0104613233
sebastia@ch.twi.tudelft.nl
versie 2, maart 2004

14 maart 2004

Blok 1

Opgave F-1

Er wordt gevraagd om een functie voor een gewogen gemiddelde te schrijven. Als eerste wordt gekeken naar het aantal opgegeven argumenten. Is deze gelijk aan 0, dan wordt een foutmelding gegeven. Als er 1 invoerargument is gegeven, wordt het gemiddelde gelijk berekend (geoptimaliseerd omdat dan de andere controles kunnen worden overgeslagen).

De eerste controle gaat over de lengten. Die moeten gelijk zijn. Indien er een rij- en kolomvector zijn opgegeven worden deze rechtgetrokken. Daarna wordt gekeken of er een negatieve waarde in zit en of de som kleiner of gelijk is aan 0. Als alles goed gekeurd werd, wordt het gewogen gemiddelde bepaald.

```
1 % wmean(x,w)
2 %
3 % de functie wmean geeft het gewogen gemiddelde terug van twee vectoren
4 % x en w. Indien w niet gegeven wordt zal voor deze de eenheidsvector
5 % genomen worden.
6 %
7 % Eisen:
8 % - x en w moeten even lang zijn
9 % - alle elementen van w moeten positief zijn
10 % - de som van de elementen van x mag niet 0 zijn
```

```

11 %
12
13 function m = wmean(x,w)
14
15 if (nargin < 2)
16     if (nargin == 0)
17         fprintf('Ongeldige invoer. Geef ten minste een vector x op.\n')
18         return
19     end
20     m = sum(x)/length(x);
21     return
22 end
23
24 if (size(x) ~= size(w))
25     if (length(x) ~= length(w'))
26         fprintf('Ongeldige invoer. x en w hebben verschillende lengten.\n')
27         return
28     else
29         w=w';
30     end
31 end
32 if (sum((w-abs(w))<0))
33     fprintf('Ongeldige invoer. w mag geen negatieve elementen hebben.\n')
34     return
35 end
36 if (sum(w) <= 0)
37     fprintf('Ongeldige invoer. Som van de elementen van w zijn niet groter
38         dan 0.\n')
39     return
40 end
41 m = sum(x.*w)/sum(w);

```

Opgave F-2

Als eerste wordt de hoogte h uitgedrukt in r om aan de inhoudseis te voldoen. Deze functie h wordt in de kosten/oppervlakte functie gebruikt. Door een eenvoudig plaatje te tekenen is te achterhalen dat het minimum tussen de 2 en de 7 ligt (het is een parabolische functie van graad 2, dus er is maar 1 minimum). Daarna `fminbnd(@kostoppervlak, 2, 7)` geeft ons als als straal $r = 4,9237$ met

bijbehorende hoogte $h = 3,2826$ (beide in meters). De kosten zijn dan 91394 dollars.

```

1 function [kosten] = kostoppervlak(r)
2
3 % kosten = 300*oppervalk cilinder + 400*oppervlak topje
4
5 h=inline('(500-2/3*pi*r^3)/(pi*r^2)');
6
7 kosten = 300*2*pi*r*h(r) + 400*2*pi*r^2;

```

Opgave F-3

De eerste stap is om de functie $V(t)$ te implementeren om als function handle gebruikt te worden. De tweede functie die we maken maakt gebruik van de functie $V(t)$ om deze geschikt te maken om de afname te berekenen. Op deze functie kun je dan **fzero** toepassen om tot de conclusie te komen dat voor $r = 10^7$ en $x = 50\%$ de benodigde tijd 54,1832 dagen is.

```

1 function [funval] = functie_v(t,r)
2 funval = 1e9 + (1e8)*(1-exp(-t/100))-r*t;

1 function [funval] = functie_min(t,r,x)
2 funval = x/100*1e9 - functie_v(t,r);

1 % afnamemoment(x,r)
2 %
3 % deze functie berekent het aantal dagen dat het duurt totdat
4 % de watervoorraad met x procent verminderd is bij een uitstroom
5 % van r liter per dag.
6 %
7 function t = afnamemoment(x,r)
8 t=fzero(@functie_min, 300, [], r, x);

```

Opgave F-4

Volgens mij mis ik bij deze opgave iets. Ik voer beide invoerargumenten direct door naar de standaardfunctie **fplot** en ik voldoe aan alle eisen.

```

% funplot(functie, interval)
%
% plot de functie 'functie' op het interval 'interval'
%

```

```
function res=funplot(funcctie , interval)
fplot(funcctie , interval);
```

Opgave F-5

Als eerste wordt het aantal argumenten gecontroleerd. Als dit minder is dan 4 ontbreekt dus de laatste, en wordt stilzwijgend aangenomen dat $N = 100$. Om consequent te blijven met de Matlab 6 functies ten opzichte van de function handles gebruik ik `f=fcnchk(f)` om van een willekeurige functie f een standaard functie te maken die met `feval` uitgerekend kan worden. Nu worden er functies als `@sin`, `'sin'` en `'sin(x)'` geaccepteerd.

```
function I = trap(f , a , b , N)
% default is 100
if ( nargin < 4)
    N=100;
end

if a > b, error('a must be less or equal to b'), end
if N < 0, error('N must be positive'), end
if N-round(N)~=0, error('N must be integer'), end
h = (b-a)/N;
sum = 0;

% zorg dat zowel strings als function handles werken
f=fcnchk(f);

for i=1:N
    sum=sum + h*feval(f , a-0.5*h+i*h);
end
I=sum;
```

Opgave M-1

Een klein eenvoudig stelsel, niet moeilijk doen dus en direct oplossen. Als uitkomst voor x krijgen we dan $(-1, 1946 \quad -1, 0447 \quad -4, 4183)^T$.

De matrix is echter wel slecht geconditioneerd: `cond(A)=Inf`.

- 1 *% niet moeilijk doen, direct oplossen:*
- 2
- 3 $A = [7 \ 9 \ -9; 3 \ 2 \ -4; 1 \ 5 \ -1]$

```

4
5 b = [22; 12; -2]
6
7 x=A\b

```

Opgave M-2

Ook deze opgave gaat wat Matlab implementatie betreft rechtoe-rechtaan. Voor elk knooppunt schrijf je de vergelijking op voor de ingaande (bekende) data en de uitgaande. Voor knooppunt 1 krijg je dus $f_1 + f_3 = 100 + 200$. Aangezien we 6 knopen hebben en 7 onbekende stromen drukken we de 7e stroom uit in bekende stromen: $f_5 + f_7 = 200 + 400$. Zouden we de 7e stroom meenemen in de matrix-vector vergelijken dan krijgen we een overbepaald systeem. Dus vullen we f_7 weer in in de eerdere vergelijkingen en komen tot een stelsel van 6 vergelijkingen met 6 onbekenden.

De verkeersdoorstroming komt op $f = (-200 \ -200 \ 500 \ 300 \ 800 \ 100 \ -200)^T$.

```

1 % matrix maken van de vergelijkingen
2 A=[1 0 1 0 0 0
3   -1 1 0 1 0 0
4     0 1 0 0 1 0
5     0 0 1 0 0 1
6     0 0 0 1 0 -1
7     0 0 0 1 -1 1];
8 b=[300 300 600 600 200 -400]';
9 f=A\b;
10 f=[f
11     600-f(5)]

```

Opgave M-3

Wanneer we met `spy(A)` naar de matrix A kijken zien we een vol diagonaal gebied met daarbuiten weinig. Het aantal elementen ongelijk aan 0 is 28831 wat neerkomt op 0,16 procent. Het plaatje bij `gplot(A, [x y])` lijkt op een ondersteboven staand vliegtuig.

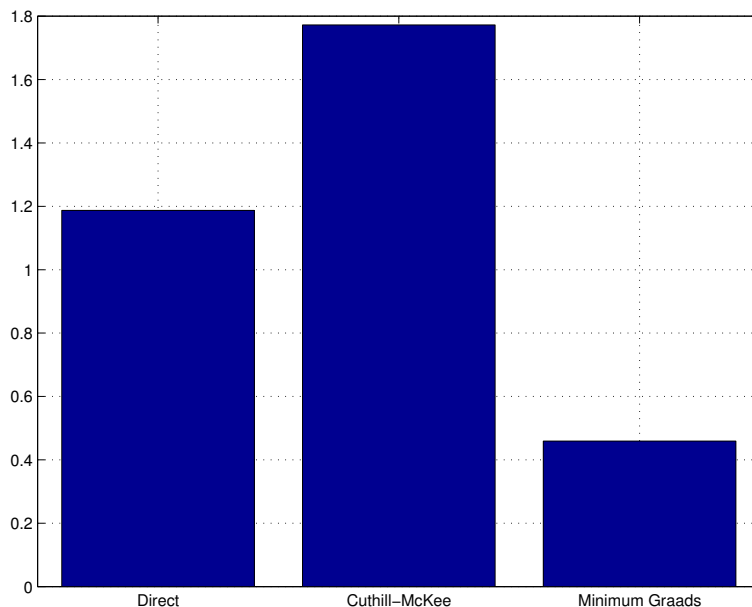
De LU decompositie geeft een onderdriehoeksmatrix L die 214755 elementen ongelijk heeft aan 0. De `spy(L)` plot geeft lange horizontale strepen vanaf de diagonaal naar links. De plot van U^T geeft hetzelfde beeld als van L met dezelfde hoeveelheid elementen ongelijk aan 0. Dat dit dezelfde elementen zijn zien we aan de plot van `spy(U'-L)` die wederom hetzelfde verloop geeft, én met dezelfde

hoeveelheid elementen ongelijk aan 0. Dus hebben U^T en L hetzelfde ijle patroon. Beter is om `find` te gebruiken en de uitvoer hiervan te controleren.

Wanneer we de Cuthill-McKee permutatie op A uitvoeren zien we dat de matrix nu een strakke diagonaal heeft, in tegenstelling tot de originele matrix die zo her en der wat diffuus was. Het patroon van L na de LU decompositie is nu een strakke onderdiagonaal met 320589 elementen ongelijk aan 0.

Als we nu hetzelfde doen met de Minimum Graads Herordening `symmmd` is de nieuwe matrix niet zo'n mooie diagonaal meer. Of juist wel. Het doet mij denken aan een kerstboom. Na de LU decompositie krijgen we een matrix L die er ook niet heel diagonaal uitziet, maar wel met weinig elementen ongelijk aan 0, namelijk 83107.

Samengevat over de drie L matrices krijgen we het overzicht gegeven in figuur 1 met betrekking tot de ijtheid van de matrix.



Figuur 1: IJtheid L matrix met verschillende methoden

```

1 clear; close all;           % clear
2
3 load airfoil;              % load data from .mat; data contains i, j, x,
   y;
4                             % i and j define connectivity
5                             % x and y define coordinates
6
7 n = max(max(i),max(j));    % search for dimensions

```

```

8  A = sparse(i,j,1,n,n);           % set square adjacency matrix
9
10 gplot(A, [x y]);                % plot finite element mesh, using connectivity
11                                 % information
12
13 % Next, our task is to create a matrix associated with the physical model.
14 % Unfortunately, the authors have not presented the true finite element
15 % matrix (as Kees said: factory secret); We have chosen to create a
16 % 'Poisson-like' positive definite matrix. For this we rely upon
17 % Gershgorin's circle theorem and the fact that we need negative
18 % off-diagonal elements for obtaining positive eigenvalues.
19
20 A = -A;                           % take care of negative off-diagonal elements
21 A = A + A';                       % connectivity information has been stored in
22                                 % in A in an one-sided manner only, using
23                                 % the fact that if i is connected to j this
24                                 % implies that j is connected to i. Here,
25                                 % we fill up the full A (apart from the
26                                 % main diagonal)
27 d = sum(abs(A))';                % determine row sums of off-diagonal
28                                 % elements of the final matrix
29 d = d + 1;                         % enhance row sum in order to enter into
30                                 % a save region
31 A = A + diag(sparse(d));        % set final matrix by plugging in a save
32                                 % contribution on the main diagonal
33
34 [L U]=lu(A);                     % maak een LU decompositie
35 spy(L);
36 spy(U');                          % geef de ijlheid van de matrices weer
37 spy(L-U');
38
39 [H I]=find(L);                    % onderzoek de gelijkheid
40 [J K]=find(U');
41 find(H-J)                          % deze moeten dan lege matrices opleveren.
42 find(I-K)
43
44 p=symrcm(A);                     % maak de Cuthill-McKee hernummering
45 B=A(p,p);
46 spy(B);                          % en bekijk de ijlheid
47 [L U]=lu(B);                    % bereken wederom de LU decompositie
48 spy(L);
49
50 p=symmmd(A);                     % maak de minimumgraad herordening

```

```

51 B=A(p,p);
52 spy(B);
53 [L U]=lu(B);
54 spy(L);
55
56 elementen=length(A)^2;      % aantal elementen in A
57 data=1/elementen*100*[214755 320589 83107];
58 bar(data);                  % maak een barplot
59 set(gca,'XTickLabel','Direct|Cuthill–McKee|Minimum_⊥Graads');
60 grid on;

```

Opgave M-4

De matrix A construeren we met behulp van Kronecker functies. Mijn opbouw van de matrix is vooral gebaseerd op veel combinaties te proberen van de Kronecker functies van verschillende blokmatrices.

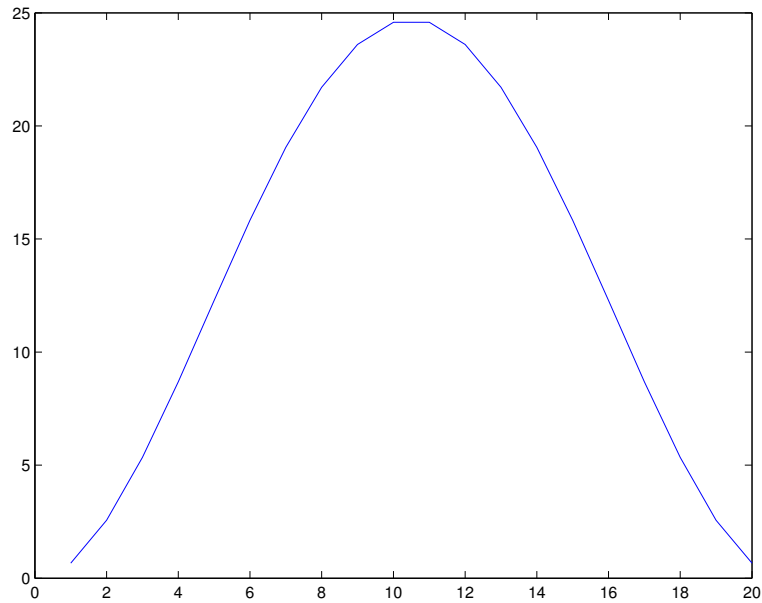
De losse opbouw van de matrix is handig om te zien wat er gebeurt, maar is alles behalve zuinig met geheugen. Derhalve is dezelfde procedure herhaald, maar nu samengeperst tot 4 regels. Nu kan op een computer met 256MB geheugen de matrix voor $n = 20$ bepaald worden.

Voor $n = 20$ is de grootte van de matrix gelijk aan 8000 bij 8000 elementen. Gevoelsmatig denk ik dat dit wel een minuutje of 10 duurt voordat deze decompositie voltooid is. Gelukkig zit ik niet op m'n oude 200Mhz computer maar op mijn nieuwe 750Mhz AMD computer, en duurt de decompositie 103,61 seconden. De matrix L is een strakke bandmatrix met 3055619 elementen ongelijk aan 0 (is 4,77) procent.

Het oplossen van het stelsel $Ax = b$ met b een vector van enen duurt na de LU decompositie 1,83 seconden. De oplossing lijkt op een klokkromme, zie figuur 2.

Wanneer we de voorgeconditioneerde Conjugate Gradients Methode (**pcg**) gebruiken duurt het oplossen 3,68 seconden na 41 iteraties. Dit is dus beduidend sneller dan eerst een LU decompositie maken en daarna oplossen. We kunnen ook proberen eerst nog de Incomplete Cholesky voorconditioner **cholinc** op A los te laten voordat we de **pcg** iteratie doen. Dit hele proces kost 10,96 seconden en stopt na 100 iteraties zonder dat de gewenste nauwkeurigheid van 10^{-6} bereikt is (relatieve rest is 0,011). Daar komt nog eens bij dat de oplossing ook nergens op lijkt, zie figuur 3

We zien dus dat het voorconditioneren met **cholinc** geen goed idee is. Wanneer we met een LU decompositie aan het werk gaan duurt het oplossen 105,44 seconden. Wanneer we echter de **pcg** iteratie gebruiken zijn we al binnen 3,68



Figuur 2: Oplossing met behulp van LU decompositie

seconden bij de oplossing, met een tolerantie van 10^{-6} , hetgeen mij redelijk lijkt voor dit soort problemen. De voorkeur ligt dus bij de `pcg` iteratie.

Nu gaan we de advection-diffusie vergelijking oplossen.

$$\epsilon u_x + \Delta u = f \tag{1}$$

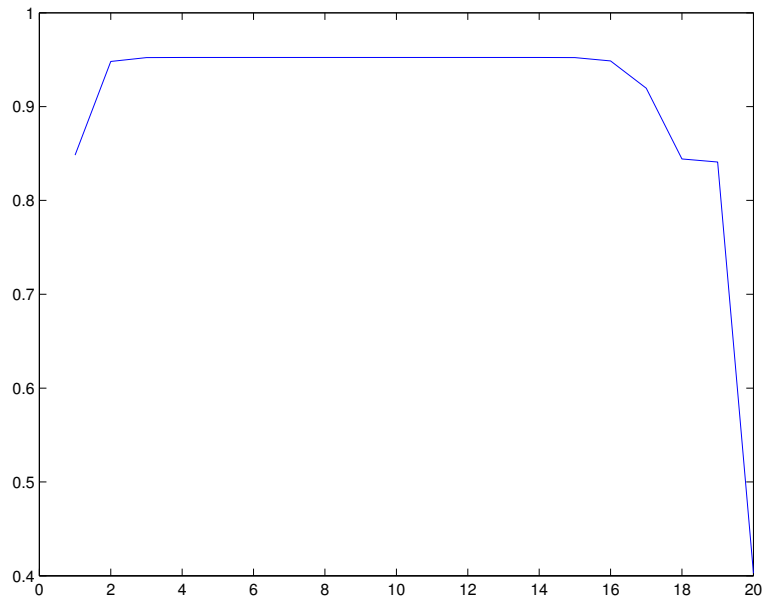
We construeren de matrix C vanzelfsprekend weer met de Kronecker functie.

We gaan $Bx = b$ weer oplossen met de LU decompositie. Dit duurt in totaal 109,48 seconden. De gegeneraliseerde minimum rest methode `gmres` convergeert in 9,97 seconden, na 6 iteraties tot de oplossing.

Het ligt dus voor de hand om voor de `gmres` methode te gaan.

```

1 n=20
2 epsilon=1/10;
3
4 %% de eerste blokmatrix, met 2 op de diagonaal
5 %B = diag(2*ones(n,1)) + diag(-ones((n-1),1),1) + diag(-ones((n-1),1),-1)
6
7 %% de tweede blokmatrix
8 %C = kron(B , eye(n));
9 %D = kron(eye(n) , B);
10 %K=C+D
```



Figuur 3: Oplossing met pcg iteratiena cholinc

```

11
12 %% de uiteindelijke vorm
13 %E=kron(C,eye(n));
14 %F=kron(eye(n),K);
15 %A=E+F
16
17 % Zuinig zijn met geheugen, wel minder overzichtelijk
18 % maar nu kunnen we A voor n=20 wel construeren
19 B = sparse(diag(2*ones(n,1)) + diag(-ones((n-1),1),1) + diag(-ones((n-1)
    ,1),-1));
20 C = sparse(kron(B, speye(n)));
21 A = sparse(kron(speye(n),C+kron(speye(n),B))+kron(C,speye(n)));
22 clear B,C;
23
24 % LU decompositie van A
25 tic;
26 [L U]=lu(A);
27 toc
28 spy(A);
29
30 % stelsel oplossen
31 b=ones(n*n*n,1);
32 tic

```

```

33 x=U\(L\b);
34 toc
35
36 % oplossing tekenen
37 plot(x([1:(n*n+n+1):(n*n*n)]))
38 pause
39
40 % gebruik nu de pcg iteratie om het stelsel op te lossen
41 tic , y=pcg(A,b,1e-6,100); toc
42 plot(y([1:(n*n+n+1):(n*n*n)]))
43 pause
44
45 % weer pcg iteratie , maar na de Choleski decompositie
46 tic , z=pcg(cholinc(A,'0'),b,1e-6,100); toc
47 plot(z([1:(n*n+n+1):(n*n*n)]))
48 pause
49
50
51 % advection-diffusion vergelijking
52 B=A+epsilon*kron(speye(n*n),sparse(1/2*(diag(ones(n-1,1),1)-diag(ones(n
    -1,1),-1)))));
53
54 % LU decompositie en oplossen
55 tic , [L U]=lu(B); u=U\(L\b); toc
56 plot(u([1:(n*n+n+1):(n*n*n)]))
57
58 % gmres iteratie
59 tic , v=gmres(B,b,20,1e-6,100); toc
60 plot(v([1:(n*n+n+1):(n*n*n)]))

```


Blok 2

Opgave G-1

Deze opgave is een inkomertje. De listing spreekt voor zich, ik denk niet dat het nodig is om de plaatjes in het verslag op te nemen.

```
1 jaar=[1990:1994];
2 temperatuur=[18 19 21 17 20];
3
4 stem(jaar,temperatuur)
5 pause
6 bar(jaar,temperatuur)
7 pause
8 stairs(jaar,temperatuur)
```

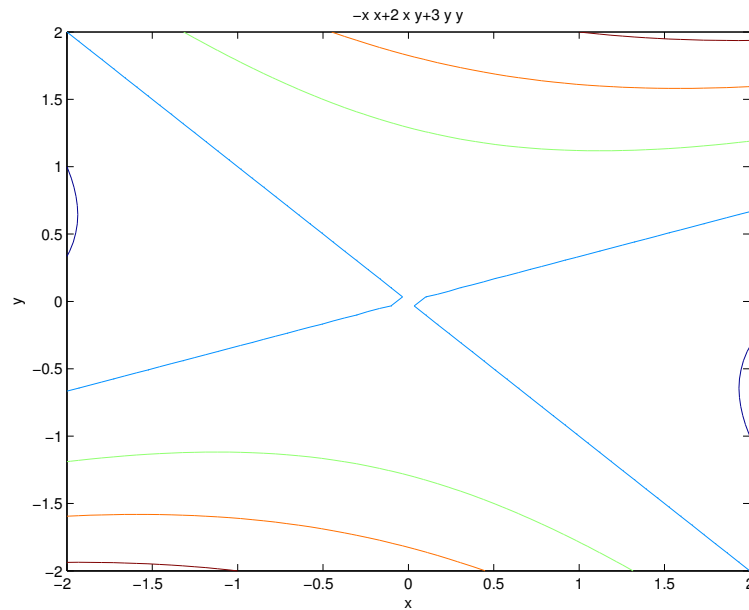
Opgave G-2

Een oppervalkteplot van een functie maken we door een meshgrid te maken en aan de hand daarvan de functie te plotten. We zien in het figuur inderdaad een zadelpunt terug. De contourplot van een zadelpunt herken je aan het feit dat je over de contourlijnen richting het punt loopt, erlangs gaat en weer wegloupt. De contourplot gegeven in figuur 4 is inderdaad wat ik verwachtte.

```
1 % eerst een grid maken
2 [X,Y]=meshgrid(-2:.1:2,-2:.1:2);
3 Z = -X.^2+2*X.*Y+3*Y.^2;
4
5 % dan een surface plot
6 surf(Z)
7 shg
8 pause
9
10 % en een eenvoudige contour
11 ezcontour('-x*x+2*x*y+3*y*y', [-2 2 -2 2]);
12 shg
```

Opgave G-3

Werkwijze is exact hetzelfde als in opgave G-2, alleen is er meer moeite gedaan om de assen netjes in beeld te krijgen, zie figuren 5 en 6.

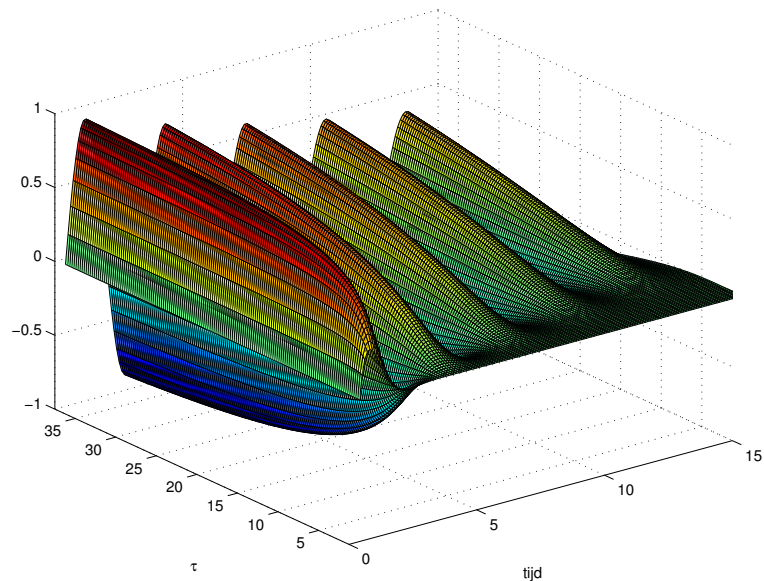


Figuur 4: Contourplot van de functie $z = -x^2 + 2xy + 3y^2$

```

1
2 % eerst een grid maken
3 [t,tau]=meshgrid(0:1:15,0.5:1:15);
4 Z = exp(-t./tau).*sin(2*t);
5
6 % dan een surface plot
7 surf(Z)
8 set(gca,'XLim',[0 150])
9 set(gca,'YLim',[5 150])
10 set(gca,'XTickLabel',[0:5:150])
11 set(gca,'YTickLabel',[5:5:150])
12 xlabel('tijd')
13 ylabel('\tau')
14 shg
15 pause
16
17 % en een eenvoudige contour
18 ezcontour('exp(-t/tau)*sin(2*t)', [0 15 0.5 15]) ;
19 set(gca,'XTick',[0:1:15])
20 set(gca,'YTick',[0:2:15])
21 shg

```



Figuur 5: Plot van de functie $z = e^{\frac{t}{\tau}} \sin(2t)$

Opgave G-4

De positie van het tweede venster wordt met `set(gcf, 'units', 'norm', 'pos', [.5 0 .5 .5])` naast de eerste gezet. Het eerste venster wordt namelijk in de hoek linksonder gezet en neemt zowel horizontaal als verticaal de helft van het scherm in beslag. Wil je het tweede venster ernaast zetten, moet je dus alleen de helft van het scherm opschuiven.

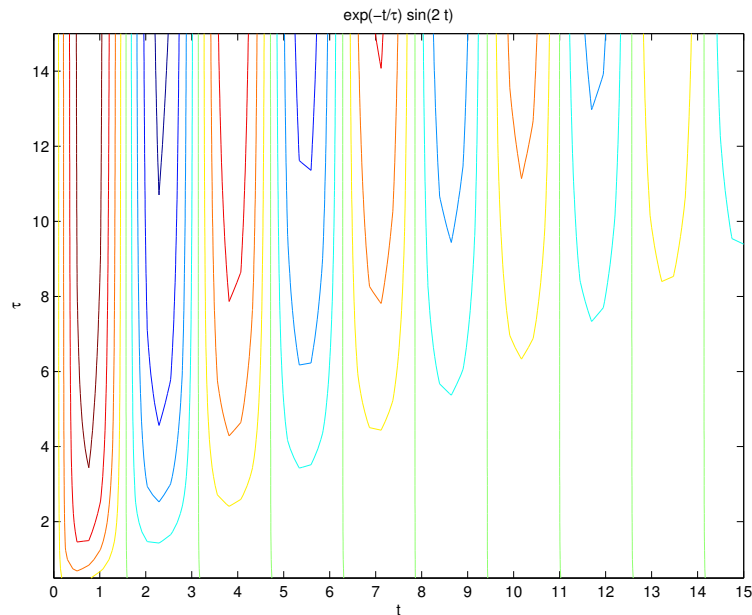
Opgave G-5

Ik maak de achtergrond standaard wit. Daarna ga ik met een loop van buiten naar binnen het gebied bepalen $< radius$ met $0 < radius < R$. Hij loopt van rood naar zwart, zie figuur 7.

```

1  % Program circimage1: circle with single color against white background
2
3  clear all; %close all;           % clear
4
5  R = 1;           % radius of circle
6  N = 200;        % for mesh on region
7
8  [x,y] = meshgrid(linspace(-2,2,N)); % create a mesh
9
10 %r = zeros(size(x));  g = r; b = r; % initialize to zero

```



Figuur 6: Contourplot van de functie $z = e^{\frac{t}{7}} \sin(2t)$

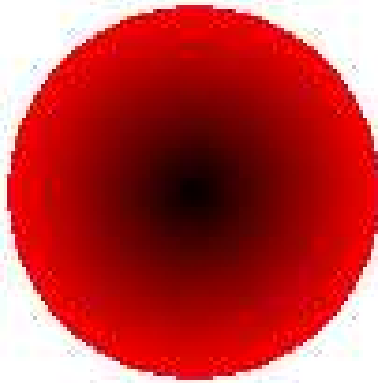
```

11                                     % r, g, b = red, green, blue
12  r = ones(size(x)); g=r; b=r;
13
14  for radius=R:-1/N:1
15      ind = find(x.^2 + y.^2 < radius^2);           % find outer region
16      r(ind) = 1-radius/R; g(ind) = 0; b(ind) = 0;   % set background to white
17  end
18
19  rgb = cat(3,r,g,b);                       % concatenate the different color
20                                             % arrays in a single 200 x 200 x 3
21                                             array
22
21  imagesc(rgb);                             % create the image
23  axis equal off;

```

Opgave G-6

Eerst maken we een nieuwe colourmap. Ons probleem heeft veel lage waarden, dus het is handig om de nieuwe kleuren zo te kiezen dat er veel verschil is in de lage waarden. Dit probeer ik te bereiken door de wortel te nemen en een beetje te spelen totdat je een combinatie krijgt die je (ik dus) mooi vindt, zie figuur 8.



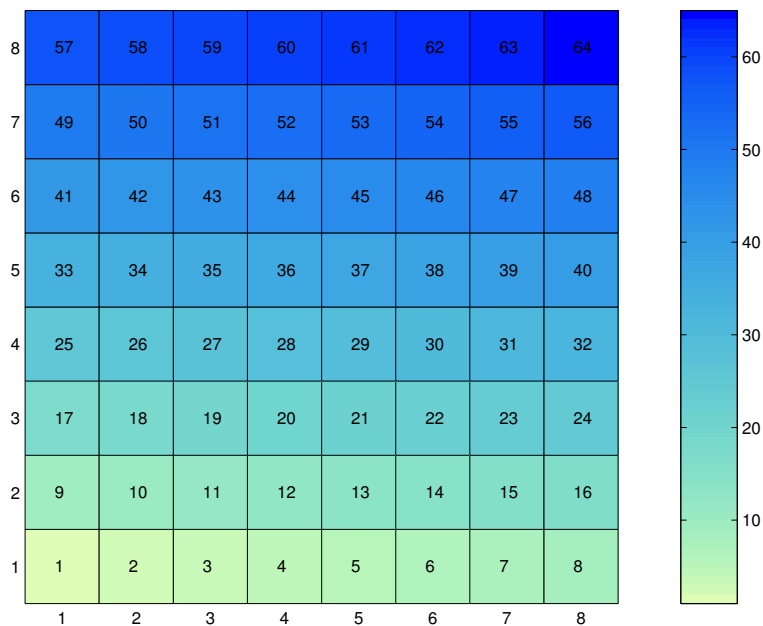
Figuur 7: Uitfadende cirkel

We maken hier een for-next loopje en tekenen voor elk tijdstip de surfaceplot. Elk frame wordt in een movie matrix opgeslagen. Als laatste wordt deze naar een AVI file geconverteerd.

De `movie2avi` conversie is niet zo triviaal als dat we graag zouden willen. Onder windoze wordt compressie gebruikt en hier komt het AVI formaat tot z'n recht (Intel Indeo 3.1/3.2 codec, grootte 624kB). Met de Linux versie van Matlab loopt het allemaal wat anders. Er wordt geen compressie gebruikt en zo wordt er een AVI animatie gemaakt die eigenlijk een ouderwetse serie bitmaps is. Deze kun je onder Linux probleemloos afspelen met `xanim`.

Echter, ongecomprimeerde filmpjes worden nogal groot (bij deze opdracht rond de 68MB). De wens bestaat dus om hier iets nuttigs mee te doen. Er doen zich 2 problemen voor: de Linux versie van Matlab produceert een AVI header die niet geheel aan de standaard voldoet en de huidige spelers (in relatie met de multimedia uitbreidingen in de computer hardware) verwacht frames die een afmeting hebben die een veelvoud is van 32 pixels (8 wil ook wel werken, maar 32 is meer standaard).

Het tweede probleem resulteert in het feit dat de frames 'uit fase' lopen. Dit is natuurlijk niet gewenst (ik zal er niet dieper op in gaan) maar is makkelijk te verhelpen. Met het commando `getframe(gcf,[25, 10, 512, 384])` pakken



Figuur 8: Kleurenmap

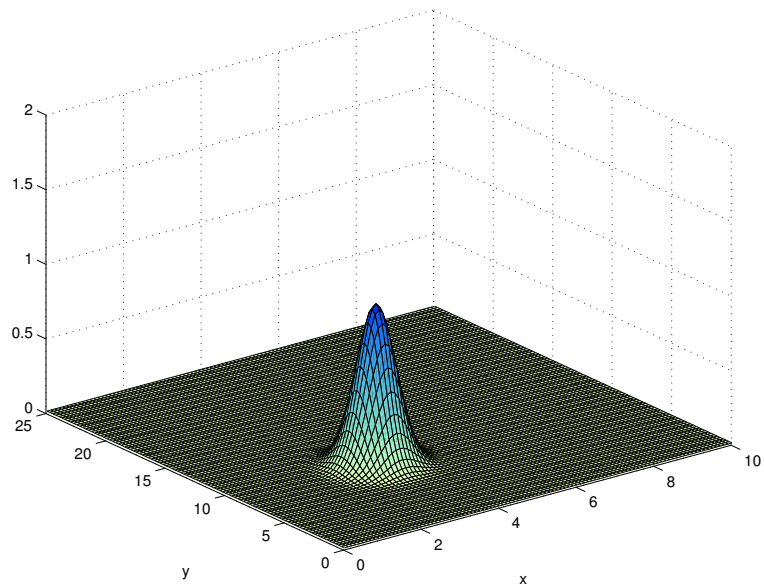
we een frame van 512×384 pixels die 25 pixels van onder en 10 pixels van links van het uitvoervenster staat. 512 en 384 zijn beide veelvouden van 32 die een verhouding hebben van 4 : 3, ofwel NTSC formaat (monitoren worden meestal in NTSC formaat gemaakt, dus een cirkel op een plaatje die verhouding 4 : 3 heeft blijft ook in full-screen mode rond, aangenomen dat de monitor goed ingesteld staat). Dit verlost ons van het tweede probleem.

Voor het eerste probleem is echter heel wat meer werk nodig. Als eerste moet de header gerepareerd worden. Dit kan onder Linux met `transcode -i movie.avi -o movie_fixed.avi -P1 -x avi,null`. Je hebt dan nu een film die met de normale afspeelprogramma's keurig loopt (zoals mplayer/avifile-player). Het filmpje is echter nog steeds niet gecomprimeerd. Dit kun je direct doen met `transcode -i movie.avi -o movie.mpg -x avi,null -y mpeg,null`. Je krijgt dan een MPEG filmpje van nog geen 5 MB. Je kunt ook andere codecs gebruiken, zoals bijvoorbeeld xvid, divx, mjpeg, etc.

Een alternatief is `avidemux movie_fixed.avi` (X applicatie). Hiermee bouw je hem makkelijk om naar een xvid gecodeerd filmpje (xvid, open source :)). We krijgen dan een filmpje van amper 500kB in grootte, ruim 130 keer zo klein.

Bovenstaand verhaal valt wellicht een beetje buiten het bestek van dit practicum, maar je kunt het als wiskundige nou eenmaal niet maken om met brakke filmpjes op te komen dagen. Met dank aan Kees Lemmens.

1 `clear;`



Figuur 9: Enkele frame uit de film

```

2 T=12;
3
4 for i=1:64
5     color(i,1)=1-sqrt(i/64);
6     color(i,2)=sqrt(1-i/64);
7     color(i,3)=sqrt(0.5+i/128);
8 end
9 colormap(color);
10
11 h=1/10;
12
13 [x,y]=meshgrid(0:1:10,0:1:10);
14 i=0;
15 for t=.1:h:T
16     i=i+1;
17     Z = 2/sqrt(t).*exp(-((x-t).^2+(y-t).^2)./(0.1*t));
18     surf(Z)
19     set(gca,'XLim',[0 100])
20     set(gca,'YLim',[0 100])
21     set(gca,'ZLim',[0 2])
22     set(gca,'XTickLabel',[0:2:100])
23     set(gca,'YTickLabel',[0:5:100])
24     xlabel('x')

```

```

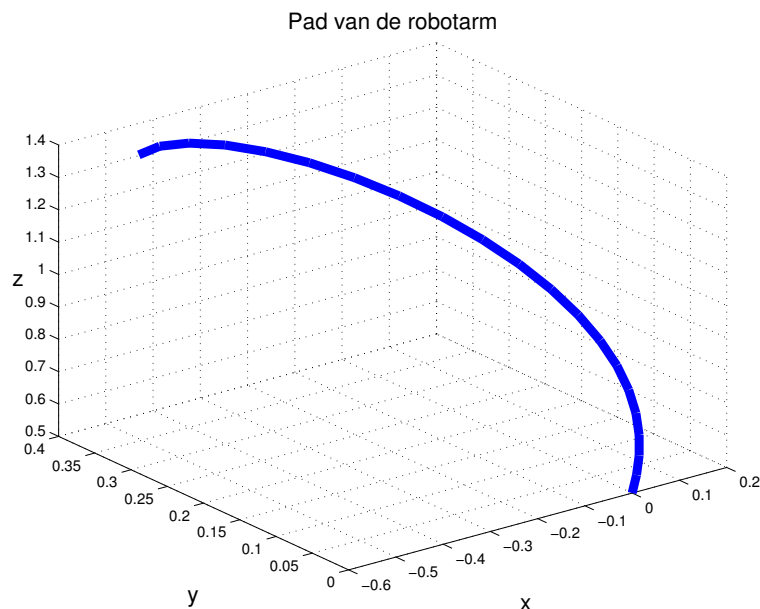
25     ylabel('y')
26     shg
27     M(i)=getframe(gcf,[25, 10, 512, 384]);
28 end
29 movie(M)
30 movie2avi(M,'/tmp/movie_uncompressed.avi','compression','None')

```

Opgave G-7

Een plaatje tekenen van een parametervoorstelling is zo gebeurd. De manier waarop het op het gepresenteerd wordt is echter niet optimaal. Ik heb het grid wat verfijnd, een titel erboven geplaatst en de lijn wat dikker getekend (daar draait het tenslotte om).

Matlab print het bestand standaard als 8" bij 6". Aangezien ik dit bestand als EPS exporteer en daarna weer in L^AT_EX gebruik speelt vergroting/verkleining geen rol: op postscript gebaseerde talen zijn oneindig scherp en het enige waar dan op gelet moet worden is de afdrukkwaliteit van het printapparaat en of de tekst makkelijk met het blote oog te lezen is (onscherpheid speelt dus geen rol meer). Alle plaatjes in dit verslag zijn op een breedte van 10cm ingesteld uitgaande van a4 papier.



Figuur 10: Parameterplot van de robotarm

```

1  % eerst een aantal punten berekenen om te plotten
2  t=0:0.01:0.2;
3  x = (0.5+5*t).*sin(2*pi/3*t).*cos(4*pi*t);
4  y = (0.5+5*t).*sin(2*pi/3*t).*sin(4*pi*t);
5  z = (0.5+5*t).*cos(2*pi/3*t);
6
7  % maak de plot
8  lijn =plot3(x,y,z);
9
10 % pas een aantal parameters aan voor de overzichtelijkheid
11 title('Pad van de robotarm','FontSize',15)
12 xlabel('x','FontSize',15)
13 ylabel('y','FontSize',15)
14 labelz=zlabel('z','FontSize',15);
15 set(labelz,'Rotation',0)
16 set(gca,'XTick',[-.6:.1:.2])
17 set(gca,'YTick',[0:.05:.4])
18 set(gca,'ZTick',[0.4:.1:1.4])
19 set(gca,'XLim',[-.6 .2])
20 set(lijn,'LineWidth',6)
21
22 % moet wat mee gespeeld worden, anders is 0 geen 0 maar .4534e-3843 (of
23 % iets dergelijks )
24 xtick =[-.6:.1:0.2]; xtick(7)=0;
25 set(gca,'XTickLabel',xtick);
26
27 grid on
28 shg

```

Opgave G-8

De functie doet er zonder veranderingen bijna 11 minuten over om tot een antwoord te komen.

Wanneer we alle storage statements uitzetten is dit gereduceerd naar 7,3 seconden. Hierbij krijg je geen plot (kan wel voor gezorgd worden, maar het doel van deze opgave ligt mijns inziens elders).

Als we de matrices van te voren vastleggen met `tstore=zeros(1,nstep)` dan duurt de berekening veel te lang bijna, 30 minuten. Dit komt omdat de manier van toewijzing verre van correct is (tenminste, ik geloof dat de bedoeling is dat je dit niet direct ziet). Nu worden er namelijk $4 \cdot 25000$ elementen gereserveerd én de nieuwe erachter gezet (in plaats van in plaats van).

Nadat we de manier van opslaan hebben aangepast en niet aan preallocating doen komen we tot een tijdsuitvoer van 10 minuten.

Als we dan ook nog het geheugen reserveren (preallocating) dan duurt het geheel nog slechts 8,52 seconden.

Wanneer we `ustore=cat(2,ustore,unew)` gebruiken met reservatie dan duurt het alsnog lang: 27 minuten. Het reserveren slaat ook nergens op aangezien `cat` de nieuwe waarde er achter plaatst (zie onderdeel c).

Misschien doe ik de preallocatie fout, maar ik zou niet goed weten hoe het anders moet. Zelf kies ik voor methode e.

```

1 function f=fde(t,u);
2
3     f=[u(2) ; u(3) ; -u(1)*u(3)];

1 % Program de: computes the solution of a system of three ordinary
2 % differential equations, using Runge–Kutta’s fourth
3 % order method and stores all results .
4
5 % uitvoering van de vraag is het gevolg van het aan en
6 % uitbecommentatieren van regels
7
8 clear all ; close all ;           % clear
9
10 nstap = 25000;                   % set number of time steps (25000)
11 h = 10/nstap;                   % set time step
12
13 t = 0;                           % initial data
14 u = [0 ; 0 ; 0.4696];
15
16 tstore = [t]; ustore = [u];      % start storage
17
18 tstore=zeros(1,nstap); ustore=zeros(3,nstap); % preallocating, vraag c,e,f
19 ustore(:,1)=u; tstore(1,1)=t;
20
21 for i = 1:nstap,                % go into time step loop
22
23     u1 = h*fde(t,u);             % do one RK4 step
24     u2 = h*fde(t+h/2,u+u1/2);   % rhs of diff. eq. is in function fde
25     u3 = h*fde(t+h/2,u+u2/2);
26     u4 = h*fde(t+h,u+u3);
27     unew = u+(u1+2*u2+2*u3+u4)/6;
28
29     t = t+h;                     % set new time

```

```

30
31 %   tstore = [tstore t];           % continue storage
32 %   ustore = [ustore unew];       %vraag a
33
34 %   tstore (1, i)=t;              % vraag d,e
35 %   ustore (:, i)=unew;
36   tstore = cat(2, tstore, t);     % vraag f
37   ustore = cat(2, ustore, unew);
38
39   u = unew;                       % new level becomes old level
40
41 end
42
43 plot(tstore, ustore (2,:))       % second component of u contains
44                                       % velocity profile in boundary
45                                       % layer

```

Opgave G-9

Dit gaat over optimaliseren. De voorgestelde procedure is een **for/next** lus met een **if** statement erin. Dit is van oudsher afgeraden. Gevraagd wordt om deze stapsgewijs te optimaliseren.

Om met deze eenvoudige functie φ een lus te gebruiken is een beetje teveel van het goede. De gegeven lus heeft 395,8734 seconden nodig om φ te maken met $J = 100000$. Met Matlab kun je echter een veel handigere manier gebruiken om zo'n vector op te zetten. Deze doet er slechts 0,0122 seconden over. Ik kan geen snellere manier bedenken.

```

1 clear;
2 J=100000
3 x=[0:1/(J-1):1];
4
5 % dit is de foute loop
6 tic
7 for j=1:J
8     if ((x(j)>0.4) & (x(j)<0.6))
9         phi(j) = 1;
10    else
11        phi(j) = 0;
12    end
13 end
14 toc

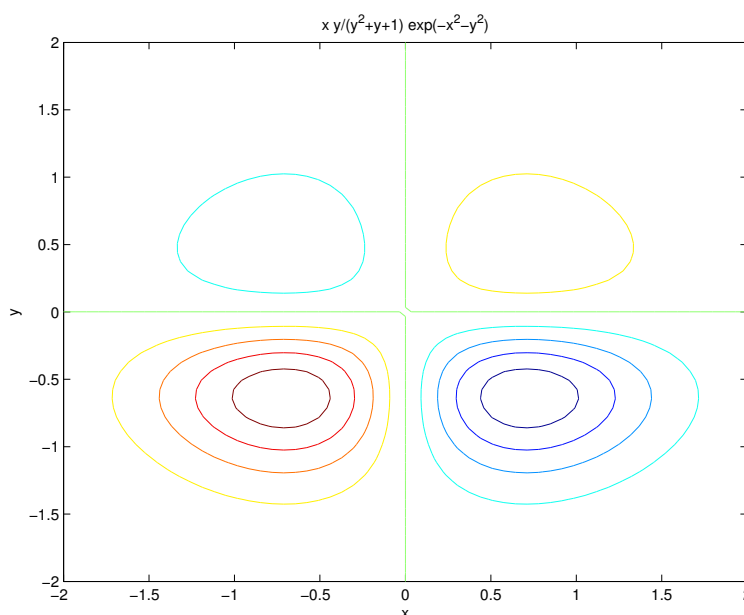
```

```
15
16 % ik weet niet hoe je dit stapsgewijs zou moeten optimaliseren, maar
17 % ik zou het volgende doen
18 tic
19 psi=[zeros(1,ceil(0.4*J)) ones(1,floor(J-0.8*J)) zeros(1,ceil(0.4*J))];
20 toc
21
22 % phi en psi zijn aan elkaar gelijk
23 sum(phi-psi)
```

Blok 3

Opgave S-1

Er wordt gevraagd alle lokale minima en maxima van de gegeven functie $f(x, y)$ te berekenen. Ik ken hier geen apart commando voor, dus ik heb eerst een contourplot gemaakt, zie figuur 11. Aan de functie is te zien dat f voor grote x en/of y naar 0 toe gaat.



Figuur 11: Contourplot van de functie $f(x, y) = \frac{xy}{y^2+y+1} e^{-x^2-y^2}$

In de contourplot zijn 4 extremen te zien. Het berekenen van de minima beginnende vanuit deze extremen doe ik door `fminsearch` te gebruiken. Ik krijg dan de volgende waarden:

x	y	$f(x, y)$
-6,1278	-29,0512	0
-0,7071	0,4764	-0,0956
0,7071	-0,6284	-0,2369
6,1278	29,0512	0

Tabel 1: Plaatsen en waarden van de minima

De reden dat Matlab minima vind voor grote waarden van y komt vermoedelijk voor afrondfouten.

Het globale minimum is het minimum des minimums, ofwel te vinden op $x_{min} = 0,70714634$, $y_{min} = -0,62835182$ en de functiewaarde $f(x_{min}, y_{min}) = -0,23690397$.

De maxima vinden we met dezelfde procedure, namelijk $\max_{x,y} f = -\min_{x,y} -f$. Aangezien een function handle geen $-$ teken voor zich accepteerd (*ezplot*($-\text{@}f$) is illegaal) heb ik aan de functie een extra parameter meegegeven die het teken bepaald. We vinden dan het globale maximum in $x_{max} = -0,70714634$, $y_{max} = -0,62835182$ en de functiewaarde $f(x_{max}, y_{max}) = 0,23690397$.

```

1 function [funval] = functie_s1(x,s)
2 % s is het teken, 1 of -1 (voor zoeken naar minima, respectievelijk maxima)
3 funval = s*(x(1)*x(2)/(x(2)^2+x(2)+1)*exp(-x(1)^2-x(2)^2));

1 % eerst een contour tekenen, dan weet je waar je moet zoeken
2 ezcontour('x*y/(y^2+y+1)*exp(-x^2-y^2)', [-2 2, -2 2])
3
4 % eerst de minima
5 [x1,fval1]=fminsearch(@functie_s1,[-1 -1],[],1 );
6 [x2,fval2]=fminsearch(@functie_s1,[-1 1],[],1 );
7 [x3,fval3]=fminsearch(@functie_s1,[1 -1],[],1 );
8 [x4,fval4]=fminsearch(@functie_s1,[1 1],[],1 );
9
10 A=[x1 fval1
11     x2 fval2
12     x3 fval3
13     x4 fval4]
14
15 [a,b]=min(A(:,3));
16 fprintf('Minimum is rond x=%1.8f, y=%1.8f en is %1.8f\n',A(b,1),A(b,2),A(b
    ,3))
17
18 % nu de maxima=-min(-f)
19 [x1,fval1]=fminsearch(@functie_s1,[-1 -1],[],-1 );
20 [x2,fval2]=fminsearch(@functie_s1,[-1 1],[],-1 );
21 [x3,fval3]=fminsearch(@functie_s1,[1 -1],[],-1 );
22 [x4,fval4]=fminsearch(@functie_s1,[1 1],[],-1 );
23
24 A=[x1 -fval1
25     x2 -fval2
26     x3 -fval3
27     x4 -fval4]
28
29 [a,b]=max(A(:,3));

```

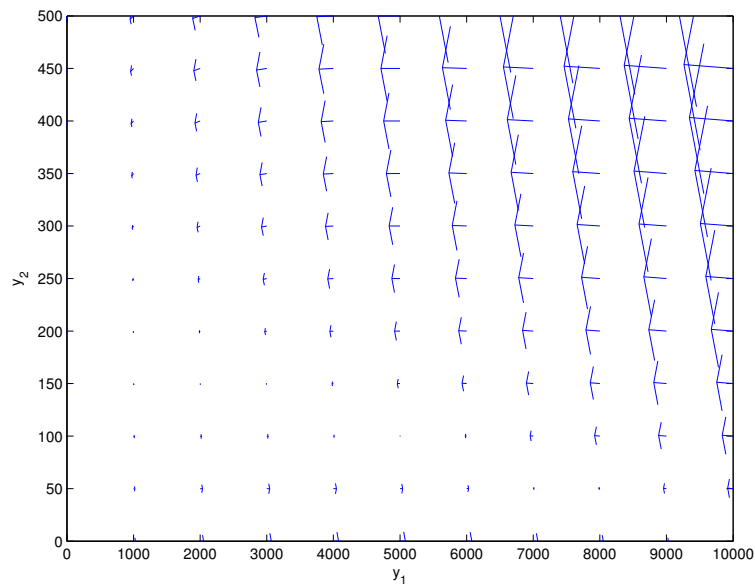
```

30 fprintf('Maximum is rond x=%1.8f, y=%1.8f en is %1.8f\n',A(b,1),A(b,2),A(b
    ,3))
31
32 pause
33 [X,Y]=meshgrid(-2:.1:2,-2:.1:2);
34 Z=X.*Y./(Y.^2+Y+1).*exp(-X.^2-Y.^2);
35 mesh(Z)

```

Opgave S-2

Het vectorveld plot ik door een grid te maken en `quiver` te gebruiken.



Figuur 12: Fasevlak van de differentiaalvergelijking

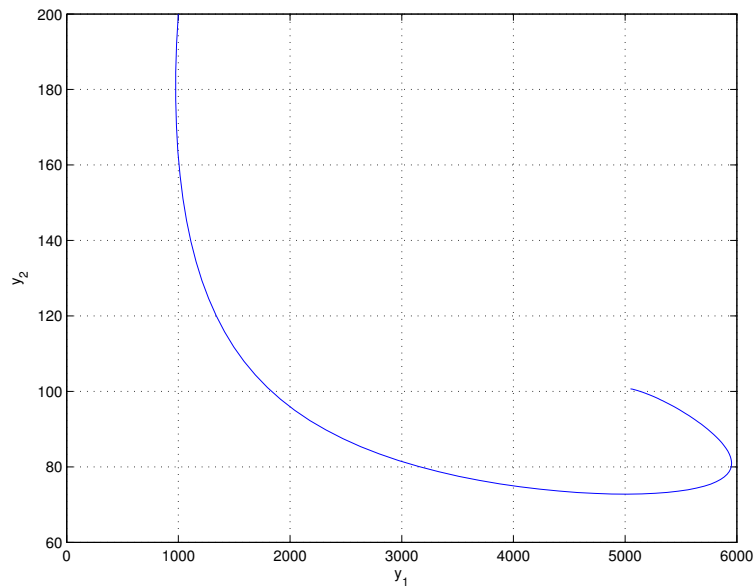
Het oplossen van het stelsel doe ik met `ode45`. De oplossing is een faseplot van $y_1(t)$ tegen $y_2(t)$.

Als laatste wordt gevraagd de eerste 4 tijdstippen te geven waar $y_1(t) = 1000$. Dit doe ik met behulp van de `Events` optie van `odeset`. Helaas krijg ik maar 2 tijdstippen waarop dit gebeurt.

```

1 % meshgrid maken voor de quiver plot
2 [y1,y2]=meshgrid(0:1000:10000,0:50:500);
3 Dy1Dt=2*y1-0.0002*y1.^2-0.01*y1.*y2;
4 Dy2Dt=-0.5*y2+0.0001*y1.*y2;
5

```



Figuur 13: Oplossing jager-prooi vergelijking

t	y_1	y_2
0	1000	200
0,524726	1000	161,994654

Tabel 2: Tijdstippen en aantallen waar $y_1(t) = 1000$

```

6 quiver(y1,y2,Dy1Dt,Dy2Dt)
7 set(gca,'XLim',[0,10000])
8 set(gca,'YLim',[0,500])
9 xlabel('y_{1}')
10 ylabel('y_{2}')
11
12 % oplossen met ode45
13 tspan=[0:.1:10];
14 [tijd ,prooien]=ode45(@prooi_accent,tspan,[1000,200]);
15
16 % en y(1) tegen y(2) plotten
17 pause
18 plot(prooien(:,1),prooien(:,2))
19 xlabel('y_{1}')
20 ylabel('y_{2}')
21
22 % vinden waar y1(t)=1000

```

```

23 options=odeset('Events',@prooi_events);
24 [ tijd ,prooien,  tijde ,prooiene,ie]=ode45(@prooi_accent,tspan,[1000,200],
      options);
25 grid on
26
27 A=[tijde prooiene]

1 function [yaccent] = prooi_accent(t,y)
2 yaccent = [ 2*y(1)-0.0002*y(1)^2-0.01*y(1)*y(2)
3           -0.5*y(2)+0.0001*y(1)*y(2)];

1 function [value,isterminal , direction ] = prooi_events(t,y)
2
3 value=y(1)-1000;
4 isterminal=0;
5 direction=0;

```

Opgave S-3

Als ik het bestand `exchem.m` direct uitvoer krijg ik de oplossing met behulp van `ode45`, hetgeen er nogal ruisig uitziet. We zetten `odeset('Stats','on')` om meer informatie te krijgen over het aantal stappen. Het kost 3,24 seconden om deze op te lossen en de waarde op $t = 3$ is gelijk aan $2,436120 \cdot 10^{-5}$. Dit na 14953 functie evaluaties.

Oplossen met behulp van de procedure voor een stijf stelsel `ode15s` kost slechts 0,11 seconden en de eindwaarde is gelijk aan $2,438383 \cdot 10^{-5}$ met 73 functie evaluaties.

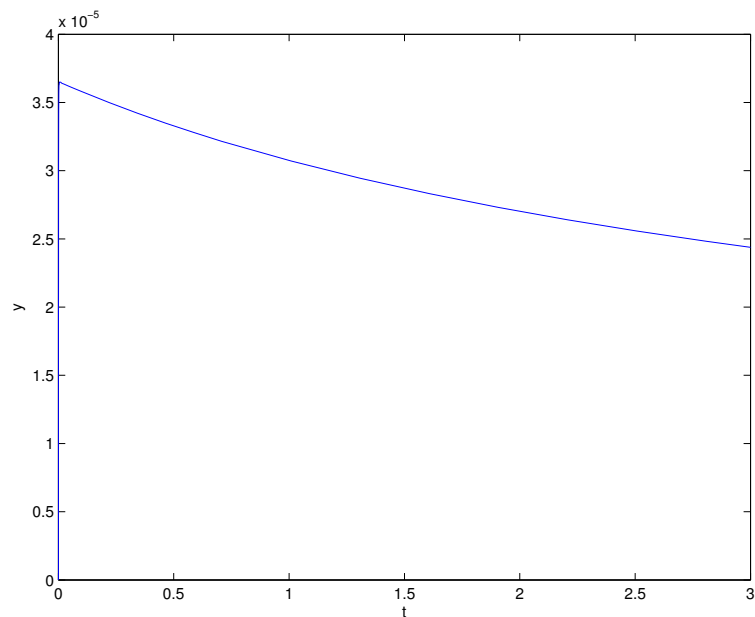
Nu zetten we de relatieve en absolute tolerantie op 10^{-8} . Met `ode45` duurt dit 3,19 seconden en de eindwaarde is $2,438026 \cdot 10^{-5}$ na 12967. De eerder geziende storing is nu weg. Voor `ode15s` neemt dit 0,33 seconden in beslag om na 127 functie evaluaties tot de waarde $2,438334 \cdot 10^{-5}$ te komen op het eindpunt.

Ik snap dit eigenlijk niet. Je legt hogere eisen op en toch kost het minder stappen om het stelsel op te lossen. Of heeft dit invloed op het aantal foutieve stappen dat minder wordt?

```

1 % program for chemical reaction
2 function exchem
3
4 %options=odeset('RelTol',1e-3,'AbsTol',1e-6,'Stats','on');
5 options=odeset('RelTol',1e-8,'AbsTol',1e-8,'Stats','on');
6

```



Figuur 14: Chemische reactie, opgelost met ode15s

```

7
8 tspan = [0 3];
9 yzero = [1;0;0];
10 tic
11 [ta,ya] = ode45(@chem, tspan,yzero,options);
12 toc
13 plot(ta,ya(:,2))
14 xlabel('t')
15 ylabel('y')
16 fprintf('Waarde op t=3, ya=%1.6e',ya(length(ya),2))
17
18 pause
19 tic
20 [ta2,ya2] = ode15s(@chem, tspan,yzero,options);
21 toc
22 plot(ta2,ya2(:,2));
23 xlabel('t')
24 ylabel('y')
25 fprintf('Waarde op t=3, ya2=%1.6e',ya2(length(ya2),2))
26
27 %-----
28

```

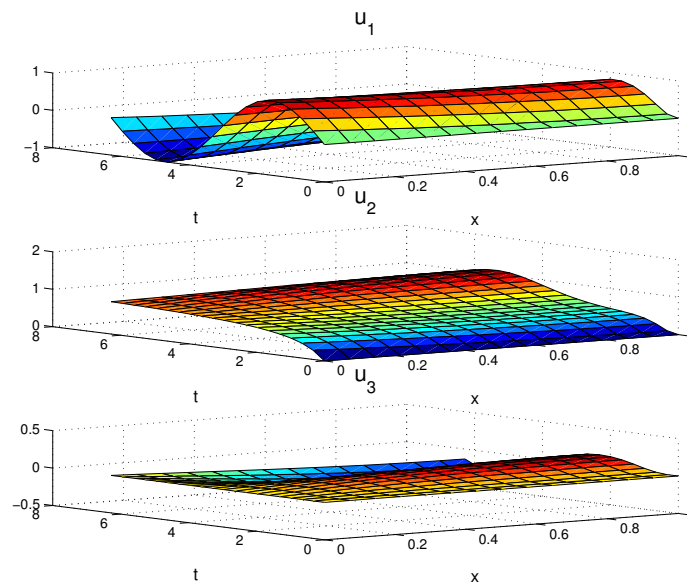
```

29 function yprime = chem(t,y)
30 % CHEM Robertson's chemical reaction model, Higham p. 157
31 %      YPRIME = CHEM(t,y)
32
33 yprime = [-0.04*y(1) + 1e4*y(2)*y(3);
34           0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)^2;
35           3e7*y(2)^2];

```

Opgave S-4

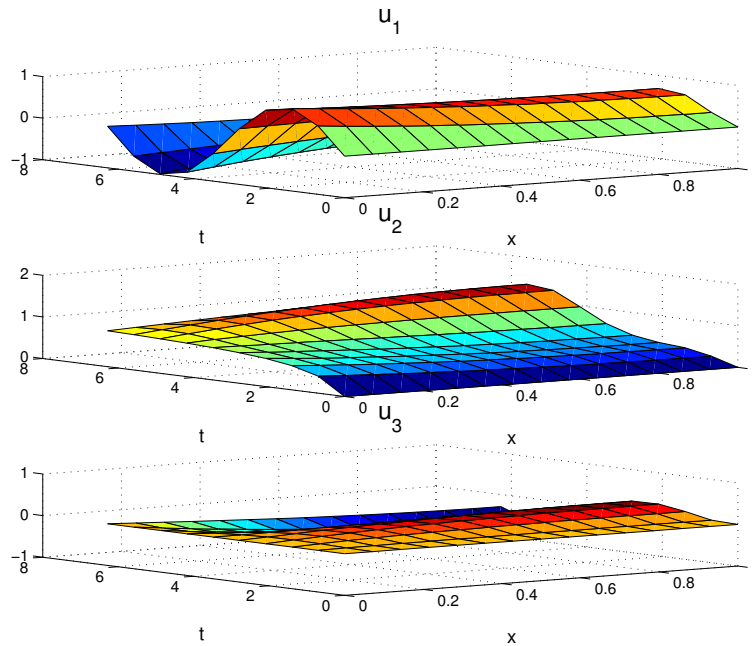
We voegen eerst a ($= c$ in de opgave) en k toe als extra argumenten aan alle functies en voegen a en k toe aan `pdepe` als extra parameters. Daarna pas ik de `tspan` aan naar het interval $[0, 2\pi]$ en pa in de randcondities. Voor $a = k = 1$ krijgen we de plots in figuur 15. De plots voor $a = 0, 2$ en $k = 2$ zien we in figuur 16.



Figuur 15: Oplossing voor $a = k = 1$

Voor de derde opdracht heb ik de `linspace` voor de `xmesh` uitgebreid naar 100 elementen en een nieuwe plotprocedure gemaakt. Het resultaat is te zien in figuur 17.

Eerst was ik vergeten om de randcondities aan te passen. Toen gaf Matlab problemen: *Warning: Failure at t=4.954947e+000. Unable to meet integration tolerances without reducing the step size below the smallest value allowed (1.760351e-*



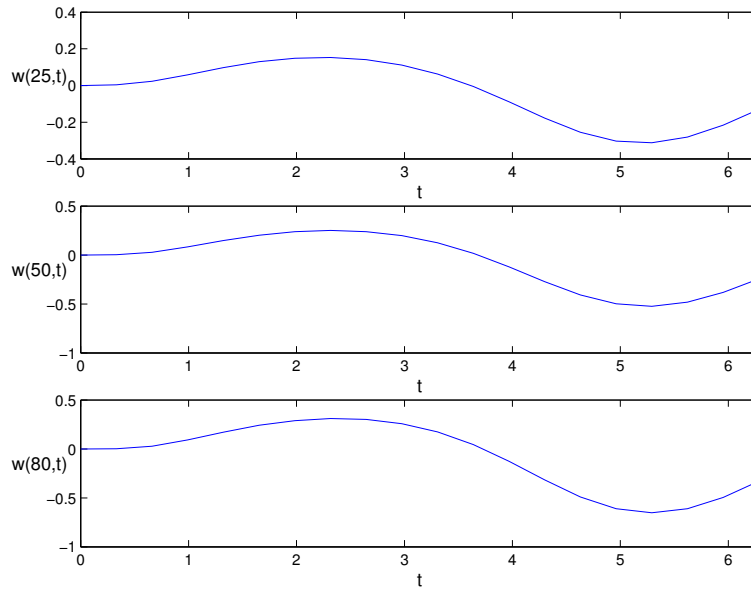
Figuur 16: Oplossing voor $a = 0, 2$ en $k = 2$

014) at time t . waarna Matlab afbreekt met de fout *Subscripted assignment dimension mismatch..* Nu lag het aan mij, maar wat moet je doen als je het probleem wel goed hebt opgegeven en je krijgt deze melding?

```

1 % Advection diffusion equation
2 %  $u_t = u_{xx} - a u_x - k*u*v$ 
3 %  $v_t = v_{xx} - a v_x - k*u*v$ 
4 %  $w_t = w_{xx} - a w_x + k*u*v$ 
5 function exadvec(a,k)
6
7 m=0;
8 xmesh = linspace(0,1,15);
9 %xmesh = linspace(0,1,101); %vraag c
10
11 tspan = linspace(0,2*pi,20);
12 sol = pdepe(m,@advecpde,@advecic,@advecbc,xmesh,tspan,[],a,k);
13 u1 = sol (:,:,1) ;
14 u2 = sol (:,:,2) ;
15 u3 = sol (:,:,3) ;
16
17 for k=1:3
18 subplot(3,1,k)

```



Figuur 17: Oplossing voor $a = 0, 1$ en $k = 3$ voor $w(x, t)$

```

19 eval(['surf(xmesh,tspan,u',int2str(k),')'])
20 xlabel('x', 'FontSize', 12);
21 ylabel('t', 'FontSize', 12)
22 title(['u_',int2str(k)], 'FontSize', 16);
23 end
24
25 % vraag c
26 %ix=[1/4;1/2;4/5]*100;
27 %for k=1:3
28 % subplot(3,1,k)
29 % eval(['plot(tspan,u3(:,',int2str(ix(k)),')'])
30 % xlabel('t', 'FontSize', 12);
31 % ylabel(['w(',int2str(ix(k)),', t)'], 'FontSize', 12, 'Rotation', 0);
32 % set(gca, 'XLim', [0 2*pi]);
33 %end
34
35 %-----
36 %%
37 function [c,f,s] = advecpde(x,t,u,dudx,a,k);
38 % a = 1;
39 % k=0.1;
40 c = [1;1;1];
41 f = dudx-a*u;

```

```

42     s = k* [-1;-1;1]*u(1)*u(2);
43
44     %%-----
45
46     function u0 = advecic(x,a,k);
47     u0 = zeros(3,length(x));
48
49     %%-----
50
51     function [pa,qa,pb,qb] = advecbc(xa,ua,xb,ub,t,a,k);
52     % a = 1;
53     % pa = ua-[sin(t);sin(t); 0];
54     pa=ua-[sin(t);t/(1+t);0];
55     qa=[0;0;0];
56     pb=[a*ub];
57     qb=[1;1;1];

```

Opgave AD-1

Hier moeten een aantal sinussen geplot worden. Als eerste heb ik een aparte functie gemaakt voor onze sinusfunctie met parameter n . Deze functie wordt voor elke n geplot en de uitvoer wordt wat aangepast om het leesbaarder te maken.

Daarna moeten de plaatjes een voor een geexporteerd worden. Dit gebeurt met het `print` commando die alles naar een bestand toe schrijft. Ik ben zo vrij geweest om encapsulated level 2 kleuren Postscript (eps2) te gebruiken in plaats van standaard Postscript omdat encapsulated Postscript zich beter leent voor verdere verwerking (zoals in dit verslag) en de plaatjes in kleur zijn.

```

1 function [y]=sinus_functie(x,n)
2 y=sin(pi*x*sqrt(n));

1 function res=sinusplots(N)
2
3 for n=1:N
4     fplot(@sinus_functie ,[0 2],[],[],[], n)
5     set(gca, 'XTick', [0:.25:2])
6     set(gca, 'XTickLabel', '0|1/4pi|1/2pi|3/4pi|pi|5/4pi|3/2pi|7/4pi|2pi')
7     set(gca, 'FontSize', 12)
8     xl=xlabel('x');
9     yl=ylabel('f_{n}');
10    tl=title(sprintf('sin(\pi*x\sqrt{%d}), n=%d', n, n));
11

```

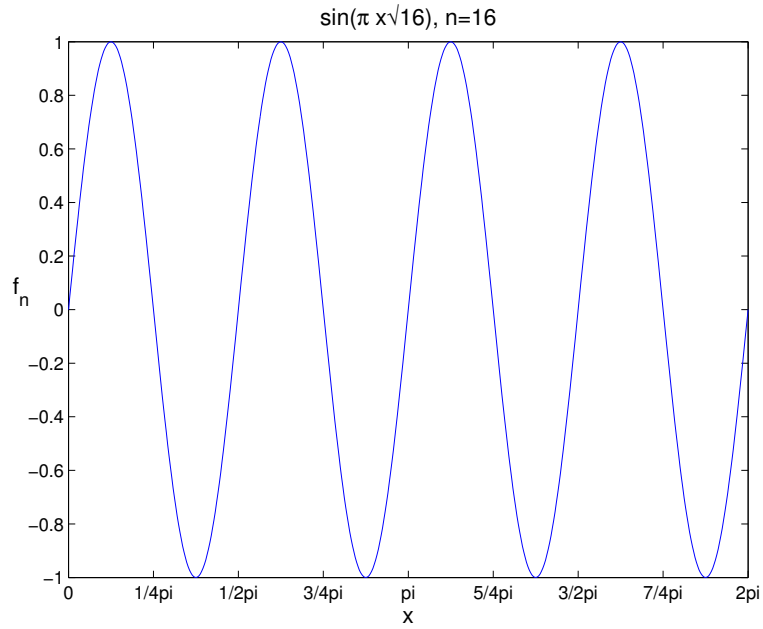
Bytes	naam	Bytes	naam
10324	sin01.eps	12312	sin16.eps
10609	sin02.eps	12308	sin17.eps
10807	sin03.eps	12365	sin18.eps
11007	sin04.eps	12406	sin19.eps
11216	sin05.eps	12449	sin20.eps
11261	sin06.eps	12442	sin21.eps
11398	sin07.eps	12485	sin22.eps
11485	sin08.eps	12510	sin23.eps
11748	sin09.eps	12548	sin24.eps
11862	sin10.eps	12598	sin25.eps
11995	sin11.eps	12570	sin26.eps
12096	sin12.eps	12604	sin27.eps
12131	sin13.eps	12644	sin28.eps
12163	sin14.eps	12648	sin29.eps
12230	sin15.eps	12659	sin30.eps

Tabel 3: Directory listing van de sinussen

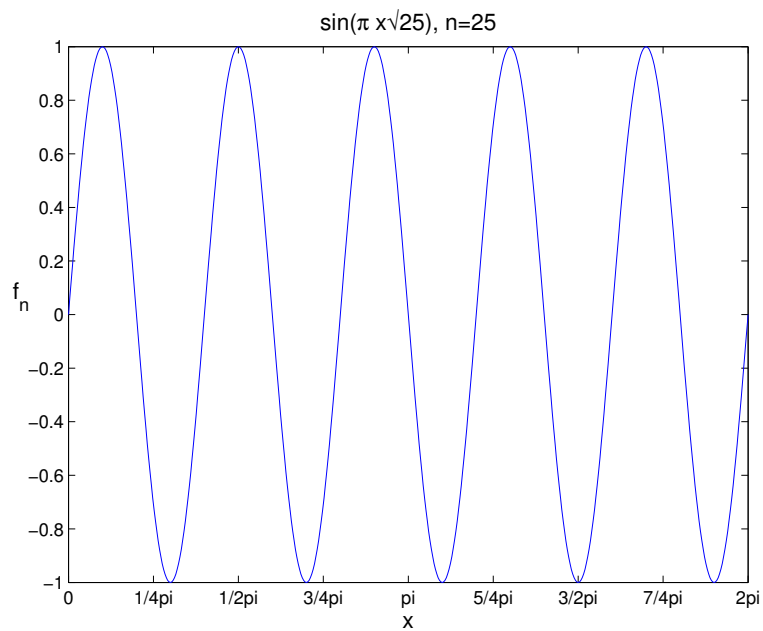
```

12     set(xl, 'FontSize',15)
13     set(yl, 'FontSize',15, 'Rotation',0)
14     set(tl, 'FontSize',15)
15
16     print(gcf, '-depsc2', sprintf('sin%.2d',n))
17 end

```



Figuur 18: plot voor $n = 16$



Figuur 19: plot voor $n = 25$