Prosjekt 1 - Introduksjon til Vitenskapelige Beregninger

Studentnr: 755110, 759144 og 753717

Februar 2016

1 Introduksjon

Vi studerer en væske Ω i en fast beholder $\Gamma_B \subset \partial\Omega$ og med en fri overflate $\Gamma_F \subset \partial\Omega$. I to dimensjoner må væsken tilfredsstille følgende betingelser for overflaten

$$\boldsymbol{n} \cdot \boldsymbol{\nabla}\psi(x,y) = \begin{cases} 0 & \text{for } (x,y) \in \Gamma_B \\ \lambda\psi(x,y) & \text{for } (x,y) \in \Gamma_F \end{cases} \tag{1}$$

og følgende betigelse for hele væsken

$$\nabla^2\psi(x,y) = 0 \text{ for } (x,y) \in \Omega. \tag{2}$$

En foreslått form på løsningen ψ er

$$\psi = \sum_{j=1}^{N} \alpha_j \Psi(x - x_j^s, y - y_j^s). \tag{3}$$

Hvor Ψ er definert som

$$\Psi(x,y) = \log\sqrt{x^2 + y^2}, \tag{4}$$

og α_j for $j = 1, ..., N$ er reelle konstanter.

2 Oppgave 1

Vi skal vise at

$$\nabla^2\psi(x,y) = 0 \tag{5}$$

for ψ gitt ved ligning (3). Deriverer først ψ med hensyn på x

$$\frac{\partial\psi}{\partial x} = \sum_{j=1}^{N} \alpha_j \frac{(x - x_j^s)}{(x - x_j^s)^2 + (y - y_j^s)^2}, \tag{6}$$

og deriverer så en gang til mhp. x, noe som gir

$$\frac{\partial^2 \psi}{\partial x^2} = \sum_{j=1}^{N} \alpha_j \frac{(y - y_j^s)^2 - (x - x_j^s)^2}{((x - x_j^s)^2 + (y - y_j^s)^2)^2}. \tag{7}$$

Ved symmetri får vi tilsvarende for derivasjon med hensyn på y

$$\frac{\partial^2 \psi}{\partial y^2} = \sum_{j=1}^{N} \alpha_j \frac{(x - x_j^s)^2 - (y - y_j^s)^2}{((x - x_j^s)^2 + (y - y_j^s)^2)^2}, \tag{8}$$

og dermed blir

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0. \quad \square \tag{9}$$

3 Oppgave 2

Med N kollokasjonspunkter, med N_B punkter langs beholderveggen og N_F punkter langs den frie overflaten, får vi N ukjente og N ligninger gitt ved overflatebetingelsene (1). Ved insettelse av ψ fra (3) inn i denne ligningen får vi

$$(\boldsymbol{n} \cdot \nabla) \cdot \sum_{j=1}^{N} \alpha_j \Psi = \begin{cases} 0 & \text{for } (x, y) \in \Gamma_B \\ \lambda \sum_{j=1}^{N} \alpha_j \Psi(x - x_j^s, y - y_j^s) & \text{for } (x, y) \in \Gamma_F \end{cases} \tag{10}$$

Dette ligningssystemet kan uttrykkes i matriseformen $A\boldsymbol{\alpha} = \lambda B\boldsymbol{\alpha}$, hvor $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_N]^T$. Matrisene A og B er begge $N \times N$ matriser, med én kolonne for hvert kildepunkt og én rad for hvert kollokasjonspunkt. A_{ij} gis så ved venstre side av (10), innsatt kollokasjonspunkt i og kildepunkt j

$$A_{ij} = \boldsymbol{n}(x_i^c, y_i^c) \cdot \nabla \Psi(x_i^c - x_j^s, y_i^c - y_j^s) \tag{11}$$

Likedan for høyre side av (10) gir

$$B_{ij} = \begin{cases} 0 & \text{for } i \in [1, N_B] \\ \Psi(x_i^c - x_j^s, y_i^c - y_j^s) & \text{for } i \in [N_B + 1, N_B + N_F] \end{cases} \tag{12}$$

4 Oppgave 3

Definisjonene av Γ_B og Γ_F er gitt fra oppgaveteksten [1] med henholdsvis $\boldsymbol{C}_B(\xi)$ og $\boldsymbol{C}_F(\xi)$, der de sistnevnte er stykkevis deriverbare funksjoner. En boksgeometri med lengde L og høyde H kan parametriseres med

$$\boldsymbol{C}_B(\xi) = \begin{cases} \boldsymbol{e}_y(H-\xi) & 0 \leq \xi < H \\ \boldsymbol{e}_x(\xi - H) & H \leq \xi < H+L \\ L\boldsymbol{e}_x + H\boldsymbol{e}_y(\xi - L - H) & H+L \leq \xi \leq 2H+L \end{cases} \tag{13}$$

$$\boldsymbol{C}_F(\xi) = \boldsymbol{e}_x(L-\xi) + H\boldsymbol{e}_y, \qquad 0 \leq \xi \leq L \tag{14}$$

Med en slik geometri, fås enhetsnormalvektorer gitt ved

$$\boldsymbol{n}_B(\xi) = \begin{cases} -\boldsymbol{e}_x & 0 < \xi < H \\ -\boldsymbol{e}_y & H < \xi < H+L \\ \boldsymbol{e}_x & H+L < \xi < 2H+L \end{cases} \tag{15}$$

$$\boldsymbol{n}_F(\xi) = \boldsymbol{e}_y, \qquad 0 < \xi < L \tag{16}$$

5 Oppgave 4

Med tilsvarende funksjoner for å beskrive Γ_B og Γ_F, kan en sirkelgeometri med sentrum i $(0,0)$, radius R, en åpning på α radianer og høyde fra sentrum til den frie overflaten $H = cos(\alpha)R$, parametriseres med

$$\boldsymbol{C}_B(\theta) = \boldsymbol{e}_\theta R = R(cos(\theta)\boldsymbol{e}_x + sin(\theta)\boldsymbol{e}_y) \qquad \frac{\pi+\alpha}{2} \leq \theta \leq \frac{5\pi-\alpha}{2} \tag{17}$$

$$\boldsymbol{C}_F(\theta) = \boldsymbol{e}_y H + \boldsymbol{e}_x sin(\theta)R \qquad -\frac{\alpha}{2} \leq \theta \leq \frac{\alpha}{2} \tag{18}$$

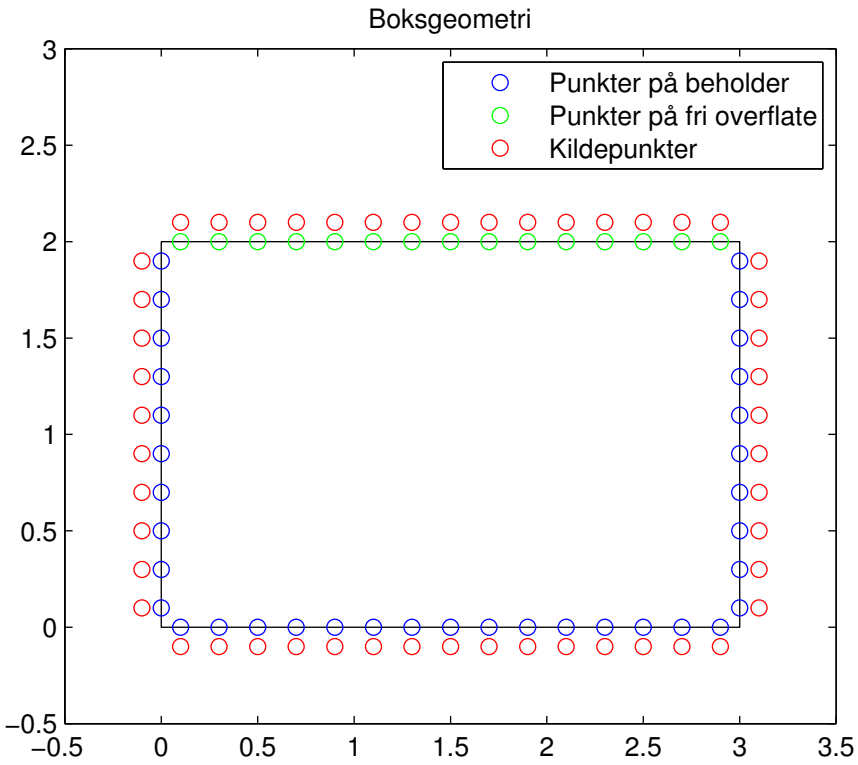
Da fås enhetsnormalvektorer gitt ved

$$\boldsymbol{n}_B(\theta) = cos(\theta)\boldsymbol{e}_x + sin(\theta)\boldsymbol{e}_y \qquad \frac{\pi+\alpha}{2} < \theta < \frac{5\pi-\alpha}{2} \tag{19}$$

$$\boldsymbol{n}_F(\theta) = \boldsymbol{e}_y \qquad -\frac{\alpha}{2} < \theta < \frac{\alpha}{2} \tag{20}$$

6 Oppgave 5

Funksjonen `getBoxPoints()` (se seksjon 12.1) tar inn lengden L og høyden H til en åpen boks, samt antall datapunkter N og en valgt avstand δ mellom kollokasjonspunktene og kildepunktene. Den returnerer koordinater til punktene, antall punkter på hhv. beholderen og den frie overflaten og randens normalvektor i punktene. Funksjonen sørger for at ingen punkter plasseres i hjørnene. Et plot av punktene returnert ved kjøring av `getBoxPoints(3,2,50,0.1)` er vist i figur 1.



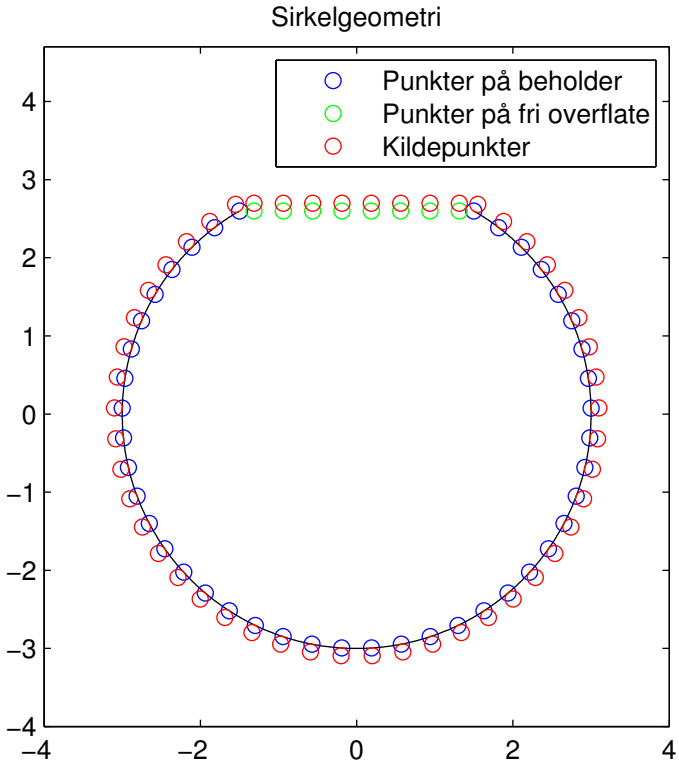
Figur 1: Plot av punktene returnert av kode gitt i seksjon 6.

`getCirclePoints()` (seksjon 12.2) er en tilsvarende funksjon for sirkelgeometri. Figur 2 viser plot av koordinatene returnert ved kjøring av `getCirclePoints(3,pi/3,99,0.1)`.

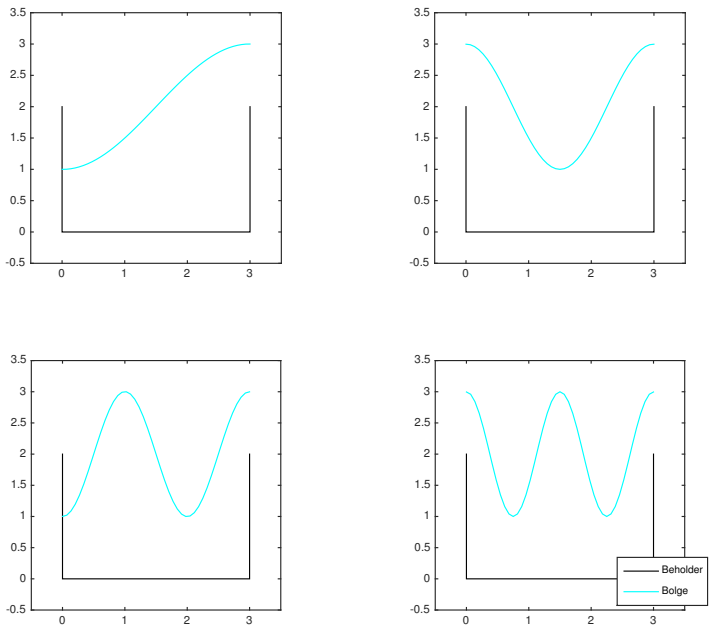
7 Oppgave 6

Det vil nå være mulig å løse ligningssystemet gitt i introduksjonen for boks- og sirkelgeometrien. Ved bruk av `getBoxPoints()` og `getCirclePoints()` kan vi nå fylle numeriske verdier inn i A og B matrisen gitt ved ligning (11) og (12). Dette gjøres med funksjonen `getEigenValues()` (se seksjon 12.3).

`getEigenValues()` tar inn kilde- og kollokasjonspunktene, samt normalvektorene som inputt. Funksjonen fyller inn A og B matrisene, og benytter seg deretter av matlab-funksjonen `eig(A,B)` til å beregne eigenverdiene og -vektorene. Til slutt fjernes negative og uendelige eigenverdier, og de resterende returneres deretter i sortert rekkefølge.



Figur 2: Plot av punktene returnert av kode gitt i seksjon 6.



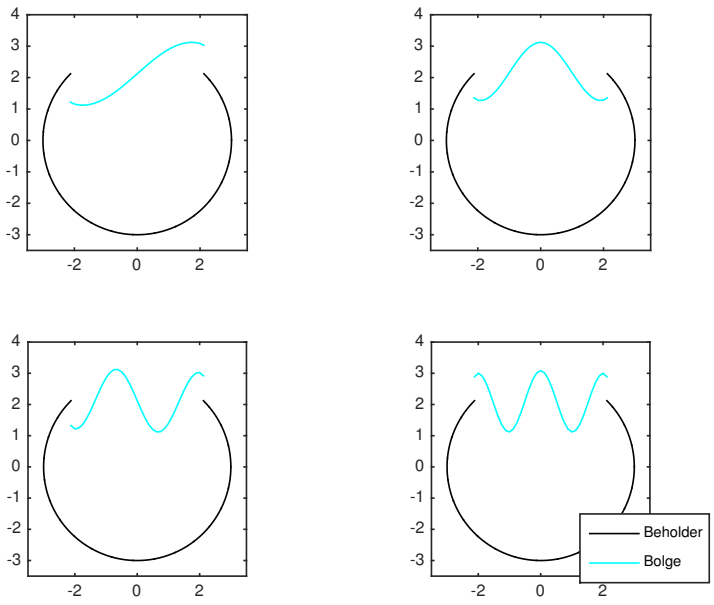
Figur 3: Boksgeometri med $L = 3$ og $H = 2$. Plott av bølgene knyttet til de fire minste, positive eigenverdiene. Med antall punkter $N = 150$ og kildepunktseparasjon $\delta = 0.2$.

Ved hjelp av eigenverdiene og de respektive eigenvektorene, vil det nå være mulig å finne formen på overflatebølgende gitt ved

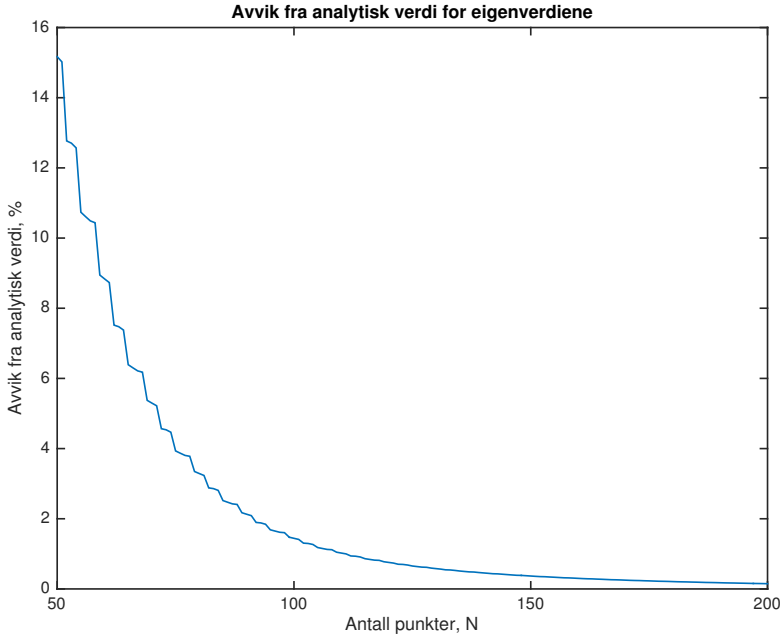
$$h(x,t) = g^{-1}\omega \sin(\omega t)\psi(x,H) \tag{21}$$

hvor $\omega = \sqrt{g\lambda}$. For en gitt eigenverdi, λ, og tilhørende eigenvektor, $\vec{\alpha}$, beregner `getWaveFunction()` (12.4) denne funkjsonen. Denne bølgefunksjonen kan så normeres (maks amplitude på 1) og plottes inn sammen med beholdergeometrien. Bølgen må også translateres til posisjonen til den frie overflaten. Bølgende knyttet til de fire minste, positive eigenverdiene er plottet i figur (3) og (4), for henholdsvis boks- og sirkelgeometrien. Bølgen til den minste eigenverdien er plottet oppe i venstre kvadrant, den nest minste oppe i høyre kvadrant, tredje minste nede til venstre og fjerde minste nede i høyre kvadrant. t i ligning (21) er satt lik 1 i disse plottene. Plot-skriptene for boks- og sirkelgeometrien er gitt i henholdsvis seksjon 12.5 og 12.6.

Men hvor nøyaktig er denne numeriske løsningen, og hvordan avhenger nøyaktigheten av antall punkter N og kildeseperasjonen δ? Siden vi har en analytisk løsning for boksgeometrien, så kan vi sammenligne den analytiske



Figur 4: Sirkelgeometri med $R = 3$ og $a = \frac{\pi}{2}$. Plott av bølgene knyttet til de fire minste, positive eigenverdiene. Med antall punkter $N = 150$ og kildepunktseparasjon $\delta = 0.2$.



Figur 5: Boksgeometri: Numerisk avvik fra analytisk verdi for eigenverdier som en funksjon av antall punkter N. Konstant parameter $\delta = 0.1$

.

løsningen med den numeriske for boksgeometrien. Ved ekstrapolering vil man kunne trekke noen slutninger om hvor nøyaktig løsningen er for de resterende geometriene.

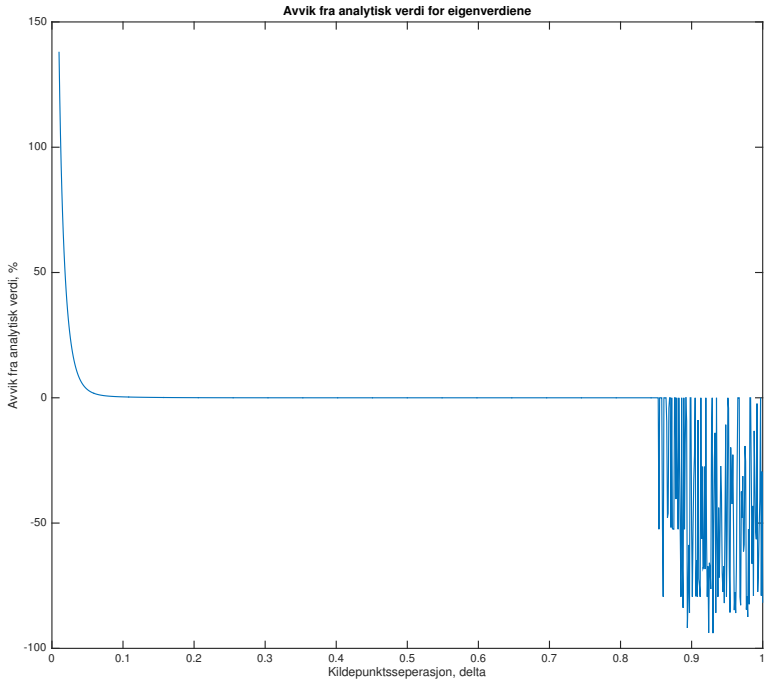
Den analytiske løsningen for eigenverdiene til boksgeometrien er

$$\lambda_n = \frac{n\pi}{L} \tanh \frac{n\pi H}{L} \tag{22}$$

Vi beregner så de numeriske eigenverdienes gjennomsnittlige avvik fra analytisk verdi for de første minste, positive eigenverdiene. Først som en funksjon av antall punkter N i figur (5), og så som en funksjon av kildepunktseperasjon δ i figur (6).

Figur (5) viser at avviket nærmer seg null assymptotisk når antall punkter N øker, som forventet. Ved $N = 150$ og $\delta = 0.1$ har vi et avvik på 0.38%, noe som er akseptabelt, og dette punktantallet er derfor i bruk i beregningene. Figur (6) viser at kildepunktseperasjon i intervallet $\delta \in [0.15, 0.8]$ gir lavest avvik. I beregningene bruker vi derfor $\delta = 0.2$. Dette gir et avvik på 0.05% når $N = 150$. Avviksanalysen er gitt i seksjon 12.5.

Avviket fra analytisk løsning er innenfor akseptable rammer for boksgeometrien. Vi velger derfor å benytte oss av de samme paramterverdiene, $N = 150$ og



Figur 6: Boksgeometri: Numerisk avvik fra analytisk verdi for eigenverdier som en funksjon av antall kildepunktsseperasjon δ. Konstant paramter $N = 150$.

$\delta = 0.2$, for de resterende geometriene som skal undersøkes. Vi kan ikke fastslå at usikkerheten for disse geometriske er av samme størrelsesorden. Men vi antar at disse parameterverdiene er gode tilnærmelser for å minimere feilen også i disse andre geometriene. Et øvre feilanslag på 2% vil trolig være tilstrekkelig, og vi kan derfor trekke slutninger med relativt god sikkerhet.

Plottet for den minste, positive eigenverdien til boksgeometrien i figur (3) kan sammenlignes med den minste, positive eigenverdien til sirkelgeometrien i figur (4) for å forutse hvilken av beholderene som er mest utsatt for søl. Forskjellen er at for boksgeometrien befinner bølgetoppen og dalen seg helt på kanten av beholderen, mens for sirkelgeometrien så er disse plassert litt vekk fra kanten. Vi vil derfor kunne annta at sirkelgeometrien er mindre utsatt for søl, fordi væskefronten mot beholderkanten er av mindre størrelse.

8 Oppgave 7

Datapunktene gitt i oppgaven [1] leses fra en .dat-fil, og returneres som vektorer, ved bruk av funksjonen `getPointVector()` (seksjon 12.7). `getInterpolant()` (seksjon 12.9) tar inn punktvektorer som argumenter og returner interpolanten $\boldsymbol{C}_B(\xi)$ som en matematisk funksjon. Dette gjøres ved bruk av Lagrange-polynomer og Chebyshev-noder. `getInterpolantDerivative` (seksjon 12.10) og `getUnitNormalFunction` (seksjon 12.11) gjør tilsvarende for hhv. den deriverte av interpolanten $\boldsymbol{C}'_B(\xi)$, og for enhetsnormalvektoren $\boldsymbol{n}_B(\xi)$.

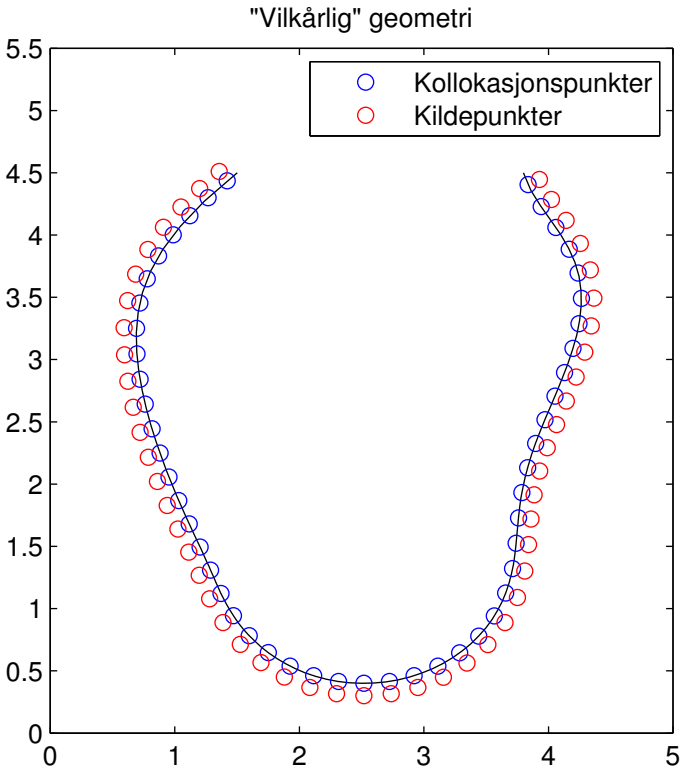
9 Oppgave 8

For å fordele punktene uniformt på en vilkårlig geometri, brukes integralene gitt i seksjon 7 i oppgaven [1] til å finne de aktuelle parameterverdiene $\hat{\xi}_j$. Dette er implementert i funksjonen `distributeInterpolationPoints()` (seksjon 12.12). Integralene beregnes numerisk ved hjelp av Matlabs innebygde funksjon `integral()`, med den øvre grensen som ukjent. Den resulterende ligningen løses ved bruk av `fzero()`, som forsøker å finne en rot av ligningen i nærheten av et gitt startpunkt.

Figur 7 viser et eksempelplot av kollokasjons- og kildepunkter returnert av `distributeInterpolationPoints()`. Den tar inn antall punkter på beholderen N_B, $\boldsymbol{C}_B(\xi)$, $\boldsymbol{C}'_B(\xi)$ og δ. Lengden av beholderen returneres også, til senere bruk.

10 Oppgave 9

`getEigenValues()` er en generell funksjon uavhengig av spesifikk geometri, og kan derfor brukes til å finne en løsning for den vilkårlige geometrien gitt i figur 7. Funksjonene utviklet i oppgave 7 og 8 genererer de nødvendige geometriske parameterne for å finne en løsningene slik som i oppgave 6. Løsningene er plottet

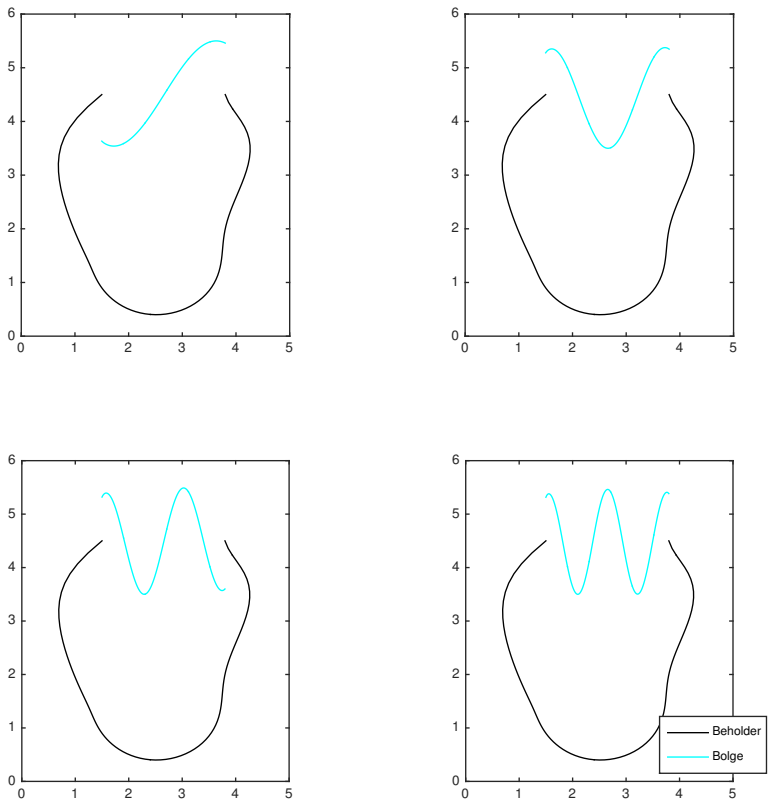


Figur 7: Plot av punktene returnert av funksjon beskrevet i seksjon 9.

i figur 8. Plot-skriptet er gitt i seksjon 12.13. Metoden er den samme som i oppgave 6, med $N = 150$ og $\delta = 0.2$.

11 Kilde

[1] Evgrafov, A. (2016) TMA4320 - Introduksjon til vitenskapelige beregninger, Vår 2016, Prosjekt 1. Tilgjengelig fra: `https://wiki.math.ntnu.no/_media/tma4320/2016v/project_sloshing.pdf` (Hentet: 15. februar 2016).



Figur 8: Generell geometri: Plott av bølgene knyttet til de fire minste, positive eigenverdiene. Med antall punkter $N = 150$ og kildepunktseperasjon $\delta = 0.2$.

12 Vedlegg

12.1 getBoxPoints.m

```
% Produces N points approximately uniformly distributed around the egde of
% a box
function [x_c y_c x_s y_s N_B N_F] = getBoxPoints(L, H, N, sourceSeparation)
% Distributing points to the edge and the free surface of the box
C = 2*H+2*L; % cirumference C
N_B = round( (2*H+L)/C*N );
N_F = round( L/C*N );
if N_B+N_F > N; N_B = N_B-1; end
E = 0; % initializing parameter epsilon

for i = 1:N_B
    E = (i-1)/(N_B-1) * (2*H+L);
    if E < H;
        x_c(i) = 0;
        y_c(i) = H-E;
        x_s(i) = -sourceSeparation;
        y_s(i) = y_c(i);
    elseif E < H+L
        x_c(i) = E-H;
        y_c(i) = 0;
        x_s(i) = x_c(i);
        y_s(i) = -sourceSeparation;
    else
        x_c(i) = L;
        y_c(i) = E-L-H;
        x_s(i) = x_c(i) + sourceSeparation;
        y_s(i) = y_c(i);
    end
end

for i = 1:N_F
    E = i/(N_F+1)*L;
    x_c(N_B+i) = L-E;
    y_c(N_B+i) = H;
    x_s(N_B+i) = x_c(N_B+i);
    y_s(N_B+i) = y_c(N_B+i) + sourceSeparation;
end

end
```

12.2 getCirclePoints.m

```
function [x_c y_c x_s y_s N_B N_F n_x n_y surfaceLength surfaceHeight] = ...
getCirclePoints(R, a, N, sourceSeparation)
% Produces N collocation points and corresponding source points, approximately uniformly
% distributed  around the egde of a "cut circle" (by an angle a) and its free horisontal
% surface

edgeLength = (2*pi-a)*R;
surfaceLength = 2*R*sin(a/2);
```

```
circumference = edgeLength+surfaceLength;
surfaceHeight = R*cos(a/2);

% Calculating distribution of the N points
N_B = round(edgeLength/circumference*N);
N_F = N-N_B;

% Forced distance from corners to avoid discontinuity
cornerSeparation = 1/2/N*circumference;

% Collocation points
T = linspace((pi+a)/2, (5*pi-a)/2, N_B);    % Initializing parameter theta
x_c = [ ...
      R*cos(T) ... % Edge
      linspace(surfaceLength/2 - cornerSeparation, cornerSeparation - surfaceLength/2, N_F) ...
      % Free surface
      ];

y_c = [ ...
      R*sin(T) ... % Edge
      surfaceHeight * ones(1,N_F) ... % Free surface
      ];

% Source points
x_s = [ ...
      (R + sourceSeparation) * cos(T) ... % Edge
      x_c(N_B+1:N) ... % Free surface
      ];

y_s = [ ...
      (R + sourceSeparation) * sin(T) ... % Edge
      (surfaceHeight + sourceSeparation) * ones(1,N_F) ... % Free surface
      ];

% Normal vectors
n_x = [ ...
      cos(T) ... % Edge
      zeros(1,N_F) ... % Free surface
      ];

n_y = [ ...
      sin(T) ... % Edge
      ones(1,N_F) ... % Free surface
      ];

end
```

12.3 getEigenValues.m

```
function [eigenVectors, eigenValues] = getEigenValues(x_c, y_c, x_s, y_s, N_B, N_F, n_x, n_y)
%Finds Steklov eigenvalues and eigenvectors of a box geometry of height H and length L

N = N_F + N_B;
A = zeros(N);
B = A;
```

```
%Checking if N_B + N_F = N
assert( (N_B + N_F) == N, 'Error: N_B + N_F != N' )

%Filling up the A matrix
for i = 1:N; %Colocation points
  for j = 1:N; %Source points
    normalVector = [n_x(i) n_y(i)];
    nablaPsiVector = nablaPsi(x_c(i) - x_s(j), y_c(i) - y_s(j));
    A(i,j) = dot(normalVector, nablaPsiVector);
  end
end

disp(A);

%Filling up the B matrix
for i = (N_B+1):N %Colocation points
  for j = 1:N %Source points
    B(i,j) = Psi(x_c(i) - x_s(j), y_c(i) - y_s(j));
  end
end

[eigenVectors, eigenValues] = eig(A,B,'vector');

eigenVectors(:,isinf(eigenValues)) = [];
eigenValues(isinf(eigenValues)) = []; %Deletes infinite eigenvalues

eigenVectors(:,eigenValues < 0) = [];
eigenValues(eigenValues < 0) = []; %Deletes negative

[eigenValues, sortIndex] = sort(eigenValues);
eigenVectors = eigenVectors(:,sortIndex);

end %function


function [nablaPsiVector] = nablaPsi(x,y);
%Returns a vector in the form of [x-component, y-component] of \nabla \cdot \Psi at (x,y)

xComponent = x ./ (x^2 + y^2);
yComponent = y ./ (x^2 + y^2);

nablaPsiVector = [xComponent, yComponent];

end %function


function z = Psi(x, y)
%Calculates the value of Psi at (x,y) and returns it
  z = (1/2) .* log(x.^2 + y.^2);
end %function
```

12.4 getWaveFunction.m

```
function h = getWaveFunction(eigenValue, eigenVector, x_s, y_s, H)
%Returns
```

```
g = 9.81;
omega = sqrt(g*eigenValue);

h = @(x,t) (omega/g) .* sin(omega .* t) * psiSum(eigenVector, x_s, y_s, x, H);

end


function z = psiSum(alphaVector, x_s, y_s, x, y);

z = 0;
N = size(alphaVector,1);

for i = 1:N
  z = z + alphaVector(i,1) * Psi(x-x_s(i),y-y_s(i));
end

end
```

12.5 plotBox.m

```
clear all
close all

%Geometry parameters
boxLength = 3;
height = 2;
N = 150;
sourceSeparation = 0.2;

%Calculate necessary geometric paramteres of box geometry
[x_c y_c x_s y_s N_B N_F n_x n_y] = getBoxPoints(boxLength, height, N, sourceSeparation);

%Plot the box
figure
plot([0 0 boxLength boxLength 0], [height 0 0 height height], 'k')
axis([-0.5 boxLength+0.5 -0.5 height+1]);
daspect([1 1 1]);
hold on

%Plot collocation and sourcepoints
h = plot(x_c(1:N_B),y_c(1:N_B),'ob',x_c(N_B+1:N_B+N_F),y_c(N_B+1:N_B+N_F),'og',x_s,y_s,'or')

%Set legend and title
title('Boksgeometri');
legend(h, {'Puntker  p    beholder', 'Punkter  p    fri overflate', 'Kildepunkter'});

%Calculate eigenvectors and eigenvalues
[eigenVectors, eigenValues] = getEigenValues(x_c, y_c, x_s, y_s, N_B, N_F, n_x, n_y);

%Plotting the waves for the four smallest, positive eigenvalues
figure
for eV = 1:4
  subplot(2,2,eV)
  plot([0 0 boxLength boxLength], [height 0 0 height], 'k');
```

```
  axis([-0.5 boxLength+0.5 -0.5 height+1.5]);
  daspect([1 1 1]);
  hold on
  h = getWaveFunction(eigenValues(eV), eigenVectors(:,eV), x_s, y_s, height);
  xWavePlot = linspace(0, boxLength, N_F);
  yWavePlot = h(xWavePlot, 1);
  yWavePlot = yWavePlot ./ max(abs(yWavePlot)) + height;
  plot(xWavePlot, yWavePlot, 'c')
end

hL = legend('Beholder', 'Bolge');
% Programatically move legend
newPosition = [0.8 0.1 0.1 0.1];
newUnits = 'normalized';
set(hL,'Position', newPosition, 'Units', newUnits);

%Analytical solutions for eigenvalues
n = 1:4;
lambda = n.*pi/boxLength.*tanh(n*pi*height/boxLength);

%Plot discrepancy from analytical solution as a function of N
i = 1;
for Ntest = 50:1:200
  [eigenVectors, eigenValues] = boxEigenValues(boxLength, height, Ntest, sourceSeparation);
  discrepancy(i) = mean(100 * (eigenValues(1:4)' - lambda) ./ lambda);
  i = i + 1;
end
figure
plot(50:1:200, discrepancy);
xlabel('Antall punkter, N')
ylabel('Avvik fra analytisk verdi, %')
title('Avvik fra analytisk verdi for eigenverdiene')
disp(discrepancy(100))

%Plot discrepancy from analytical solution as a function of sourceSeparation
j = 1;
for sPTest = 0.01:0.001:1
  [eigenVectors, eigenValues] = boxEigenValues(boxLength, height, 150, sPTest);
  sPDiscrepancy(j) = mean(100 * (eigenValues(1:4)' - lambda) ./ lambda);
  j = j + 1;
end
figure
plot(0.01:0.001:1, sPDiscrepancy);
xlabel('Kildepunktsseperasjon, delta')
ylabel('Avvik fra analytisk verdi, %')
title('Avvik fra analytisk verdi for eigenverdiene')
```

12.6 plotCircle.m

```
clear all
close all

%Geometric paramaters
R = 3;
a = pi/2;
N = 150;
```

```
sourceSeparation = 0.2;

%Creating linearly distributed plot parameter
T = linspace((pi+a)/2, (5*pi-a)/2, 1000);

%Calculating necessary geometric parameters of circle geometry
[x_c y_c x_s y_s N_B N_F n_x n_y surfaceLength surfaceHeight] = getCirclePoints(R, a, N, sourceSepar
figure
plot(R*cos(T), R*sin(T), 'k');
axis([-R-1 R+1 -R-1 R+1.7]);
daspect([1 1 1]);
hold on;
h = plot(x_c(1:N_B),y_c(1:N_B),'ob',x_c(N_B+1:N_B+N_F),y_c(N_B+1:N_B+N_F),'og',x_s,y_s,'or');
title('Sirkelgeometri');
legend(h, {'Puntker pa beholder', 'Punkter pa fri overflate', 'Kildepunkter'});

[eigenVectors, eigenValues] = getEigenValues(x_c, y_c, x_s, y_s, N_B, N_F, n_x, n_y);

figure
for eV = 1:4
  subplot(2,2,eV)

  plot(R*cos(T), R*sin(T), 'k');
  axis([-R-0.5 R+0.5 -R-0.5 R+1]);
  daspect([1 1 1]);
  hold on;

  h = getWaveFunction(eigenValues(eV), eigenVectors(:,eV), x_s, y_s, surfaceHeight);
  xWavePlot = linspace(-surfaceLength/2, surfaceLength/2, N_F);
  yWavePlot = h(xWavePlot, 1);
  yWavePlot = yWavePlot ./ max(abs(yWavePlot)) + surfaceHeight;
  plot(xWavePlot, yWavePlot, 'c')
end

hL = legend('Beholder', 'Bolge');
% Programatically move legend
newPosition = [0.8 0.1 0.1 0.1];
newUnits = 'normalized';
set(hL,'Position', newPosition, 'Units', newUnits);
```

12.7 getPointVector.m

```
function [P_x P_y] = getPointVector(fileName)
% Loads .dat-file with list of points, returns them in vectors

data = dlmread(fileName,' ',1,1);
P_x = data(:,1);
P_y = data(:,2);

end
```

12.8 getChebyshevNodes.m

```
function chebyshevNodes = getChebyshevNodes(a,b,n);
```

```
% Returns chebyshevnodes for a given interval (a,b) and n datapoints

chebyshevNodes = cos( (2*n-(2:2:2*n)+1) * pi/(2*n) )*(b-a)/2 + (a+b)/2;

end
```

12.9 getInterpolant.m

```
function [interpolant_x, interpolant_y] = getInterpolant(P_x,P_y)
% Returns parametrized interpolant of the points P by the use of chebyshev,
% with paramater E (0<=E<=1)

n = length(P_x);
a = ( cos(pi/(2*n))-1 ) / ( 2*cos(pi/(2*n)) );
b = ( cos(pi/(2*n))+1 ) / ( 2*cos(pi/(2*n)) );
chebyshevNodes = getChebyshevNodes(a,b,n);

% Initalizing interpolant
interpolant_x = 0;
interpolant_y = 0;

syms E;
for j = 1:n
    p = 1;  % Initializing variabel for Lagrange polygons

    for i = 1:n
        if i~=j
            p = p*(E - chebyshevNodes(i)) / (chebyshevNodes(j) - chebyshevNodes(i));
        end
    end

   interpolant_x = interpolant_x + P_x(j)*p;
   interpolant_y = interpolant_y + P_y(j)*p;
end

interpolant_x = matlabFunction(interpolant_x);
interpolant_y = matlabFunction(interpolant_y);

end
```

12.10 getInterpolantDerivative.m

```
function [interpolantDerivative_x, interpolantDerivative_y] = ...
    getInterpolantDerivative(P_x,P_y)
% Returns the derivative of a parametrized interpolant of the points P by the
% use of chebyshev, with parameter E (0<=E<=1)

n = length(P_x);
a = ( cos(pi/(2*n))-1 ) / ( 2*cos(pi/(2*n)) );
b = ( cos(pi/(2*n))+1 ) / ( 2*cos(pi/(2*n)) );
chebyshevNodes = getChebyshevNodes(a,b,n);

% Initalizing variabel for the derivate of the interpolant
interpolantDerivative_x = 0;
```

```
interpolantDerivative_y = 0;

syms E;
for j = 1:n
    p = 0;  % Initializing variabel for derivative of Lagrange polygons

    for i = 1:n
        if i~=j

            f = 1;  % Temporary variabel for storing factor
            for k = 1:n
                if k~=i && k~=j
                    f = f*(E - chebyshevNodes(k)) / (chebyshevNodes(j) - chebyshevNodes(k));
                end
            end

            p = p + f/(chebyshevNodes(j)-chebyshevNodes(i));
        end
    end

    interpolantDerivative_x = interpolantDerivative_x + P_x(j)*p;
    interpolantDerivative_y = interpolantDerivative_y + P_y(j)*p;
end

interpolantDerivative_x = matlabFunction(interpolantDerivative_x);
interpolantDerivative_y = matlabFunction(interpolantDerivative_y);

end
```

12.11 getUnitNormalFunction.m

```
function [unitNormalFunction_x, unitNormalFunction_y] = getUnitNormalFunction( ...
    interpolantDerivative_x, interpolantDerivative_y)
% Returns a unit normal function of a parametrized interpolant of for a
% given with paramerter E (0<=E<=1)

% Calculating unit normal
syms E;
unitNormalFunction_x = interpolantDerivative_y(E) / sqrt(interpolantDerivative_x(E)^2 + ...
interpolantDerivative_y(E)^2);
unitNormalFunction_y = -interpolantDerivative_x(E) / sqrt(interpolantDerivative_x(E)^2 + ...
interpolantDerivative_y(E)^2);

unitNormalFunction_x = matlabFunction(unitNormalFunction_x);
unitNormalFunction_y = matlabFunction(unitNormalFunction_y);

end
```

12.12 distributeInterpolationPoints.m

```
function [colP_x, colP_y, sourceP_x, sourceP_y, containerLength] = ...
    distributeInterpolationPoints(N_B, interpolant_x, interpolant_y, ...
    interpolantDerivative_x, interpolantDerivative_y, sourceSeparation)
% Produces N_B collocation points and corresponding source points,
```

```
% approximately uniformly distributed around the egde of a container

a = ( cos(pi/(2*N_B))-1 ) / ( 2*cos(pi/(2*N_B)) );
b = ( cos(pi/(2*N_B))+1 ) / ( 2*cos(pi/(2*N_B)) );
chebyshevNodes = getChebyshevNodes(a,b,N_B);

syms E;
interpolantDerivativeMagnitude = sqrt(interpolantDerivative_x(E)^2 + ...
    interpolantDerivative_y(E)^2);
interpolantDerivativeMagnitude = matlabFunction(interpolantDerivativeMagnitude);

% Defining integral function
I = @(E_lLimit, E_uLimit) integral(interpolantDerivativeMagnitude, E_lLimit, E_uLimit);

containerLength = I(chebyshevNodes(1), chebyshevNodes(N_B));
stepLength = containerLength/N_B;
E = zeros(1,N_B);

% Finding parameter value for the first point, and then the rest
E(1) = fzero(@(E_uLimit) I(chebyshevNodes(1), E_uLimit) - stepLength/2, 0);

for i = 1:N_B-1
    E(i+1) = fzero(@(E_uLimit) I(E(i), E_uLimit) - stepLength, E(i));
end

% Collocation points
colP_x = interpolant_x(E);
colP_y = interpolant_y(E);

% Source points
[unitNormalFunction_x, unitNormalFunction_y] = getUnitNormalFunction( ...
    interpolantDerivative_x, interpolantDerivative_y);
unitNormal_x = unitNormalFunction_x(E);
unitNormal_y = unitNormalFunction_y(E);

sourceP_x = colP_x + unitNormal_x*sourceSeparation;
sourceP_y = colP_y + unitNormal_y*sourceSeparation;

end
```

12.13 plotInterpolation.m

```
clear all;
close all;

%Plot paramaters
N_B = 125; %Results in N     150
sourceSeparation = 0.2;

%Import data points of geometry
[P_x P_y] = getPointVector('data.dat');

%Calculate necessary geometric parameters of the geometry
[C_Bx, C_By] = getInterpolant(P_x,P_y);
[dC_Bx, dC_By] = getInterpolantDerivative(P_x,P_y);
[colP_x, colP_y, sourceP_x, sourceP_y, containerLength] = distributeInterpolationPoints( ...
```

```
    N_B, C_Bx, C_By, dC_Bx, dC_By, sourceSeparation);

%Plotting the geometry
figure
plot(C_Bx(0:0.01:1), C_By(0:0.01:1), 'k');
hold on
h = plot(colP_x, colP_y, 'bo', sourceP_x, sourceP_y, 'ro');
axis([0 5 0 5+0.5]);
daspect([1 1 1]);
title('"Vilkarlig" geometri');
legend(h, {'Kollokasjonspunkter', 'Kildepunkter'});

%Assembling necessary vectors and variables for eigenvalue calculations with getEigenValues
surfaceLength = 3.8-1.5;
N_F = round(surfaceLength* (N_B/containerLength));
fprintf('N = %3.0f', N_B + N_F)
x_c = [colP_x linspace(3.8, 1.5, N_F)];
y_c = [colP_y 4.5*ones(1,N_F)];
x_s = [sourceP_x linspace(3.8, 1.5, N_F)];
y_s = [sourceP_y 5.5*ones(1,N_F)];
n_x = x_s - x_c;
n_y = y_s - y_c;

%Calculating the eigenvectors and eigenvalues
[eigenVectors, eigenValues] = getEigenValues(x_c, y_c, x_s, y_s, N_B, N_F, n_x, n_y);

%Plotting the four waves corresponding to the 4 smallest eigenvalues
figure
height = 4.5;
t = 1;
for eV = 1:4
  subplot(2,2,eV)
  plot(C_Bx(0:0.01:1), C_By(0:0.01:1), 'k');
  axis([0 5 0 6]);
  daspect([1 1 1]);
  hold on
  h = getWaveFunction(eigenValues(eV), eigenVectors(:,eV), x_s, y_s, height);
  xWavePlot = linspace(1.5, 3.8, 10*N_F);
  yWavePlot = h(xWavePlot, t);
  yWavePlot = yWavePlot ./ max(abs(yWavePlot)) + height;
  plot(xWavePlot, yWavePlot, 'c')
end

hL = legend('Beholder', 'Bolge');
% Programatically move legend
newPosition = [0.8 0.1 0.1 0.1];
newUnits = 'normalized';
set(hL,'Position', newPosition, 'Units', newUnits);
```