Miniprojekt 2
Beräkningsvetenskap 3
Fredrik Gustafsson
2018-09-22

In this projekt the finite element method and application of it in the form of the PDE toolbox of matlab was explored. The finite element method(FEM) can be used to approximate continous functions by using piecevise linear functions between nodes, whose values is the solution at these points. When doing this in 1 dimension it is similar to just calculate the value of the points analyticaly and interpolate between them. However in higher dimensions FEM starts to shine since it can be used to calculate the solution to equation on complex domains by fitting a triangular mesh on them. This makes it a useful tool when solving thing like heat equations or motion of fluids in 2 or 3 dimensions which might be close to impossible to solve purely analytically.

The first part consisted of solving the steady state heat equation $-k\frac{dT^2}{dx^2} = Q + h(T_{ext} - T)$ with k=h=1 and Q=0 and with boundary conditions T(0)=40 and T(10)=200 and $T_{ext} = 20$. By calling $\frac{dT^2}{dx^2} = T$" we arrive at the equation $-T" + T = T_{ext}$. In order to solve this with FEM we first need to compute the weak solution. This is done in a couple of steps. First we multiply with a test function v which fullfill the criteria of belonging to a space which is continous on the intervall in question, 0 to 10, whose derivative is picewise continous and limited on [0, 10], and $v(0) = \alpha, (10) = \beta$. Next we integrate it to get the equation $-\int_0^{10}(T"v - Tv)dx = \int_0^{10} T_{ext}v$.

Next we partial integrate the term -T"v arriving at $\int_0^{10} T'v'$ and due to $V(0) = \alpha v(10) = \beta$ the two other terms is equal to zero since we don't need to calculate it for these two points, the solution is known. We then want to find a solution for T belonging to the same space as v for the equation $\int_0^{10}(T'v' + Tv)dx = \int_0^{10} T_{ext}v$. A solution to this is given by $T = \sum_{j=1}^{n-1} c_j\phi_j + \alpha\phi_0 + \beta\phi_n$ so we want to find $c_j, j = 1, 2, ..., n-1$ such that

$$\sum_{j=1}^{n-1} c_j \int_0^{10} (\phi_j'\phi_i' + \phi_j\phi_i)dx + \alpha \int_0^{10} (\phi_j'\phi_i' + \phi_j\phi_i)dx + \beta \int_0^{10} (\phi_j'\phi_i' + \phi_j\phi_i)dx = T_{ext} \int_0^{10} \phi_i dx \tag{1}$$

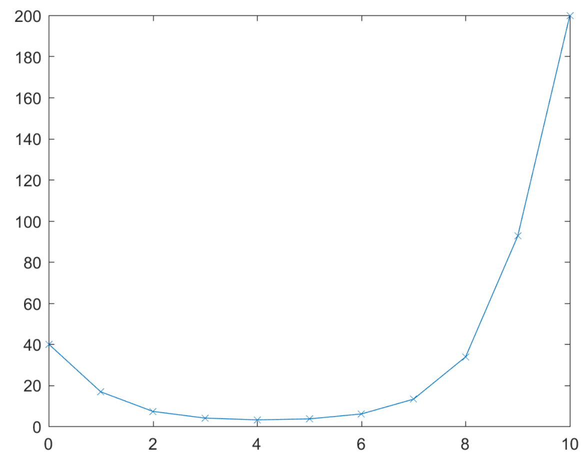
by computing the integrals and using a linear system of Kc=b we arrive at

$$K = \begin{bmatrix} 2/h + 2h/3 & -1/h + h/6 & 0 & 0 & ... & ... & 0 \\ -1/h + h/6 & 2/h + 2h/3 & -1/h + h/6 & 0 & ... & ... & 0 \\ 0 & -1/h + h/6 & 2/h + 2h/3 & -1/h + h/6 & 0 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & 0 & 0 & -1/h + h/6 & 2/h + 2h/3 \end{bmatrix}$$

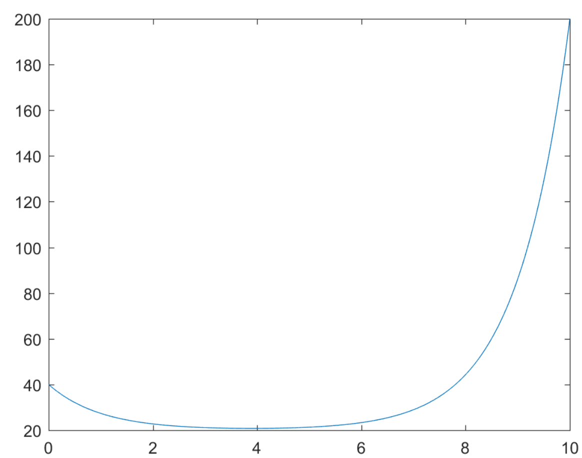
(2)

$c = [c1, ..., c_{n-1}]^T$ $b_1 = 20\int_0^{10} \phi_1 dx - 40\int_0^{10}(\phi_j'\phi_i' + \phi_j\phi_i)dx,$ $b_i = 20\int_0^{10} \phi_1 dx$
$i = 1, 2, ..., n-1$
$b_n = 20\int_0^{10} \phi_1 dx - 200\int_0^{10}(\phi_j'\phi_i' + \phi_j\phi_i)dx$ and $b = [b1, b2, ...b_n]^T$

By implementing this in matlab and for example using 10 nodes one get the plot

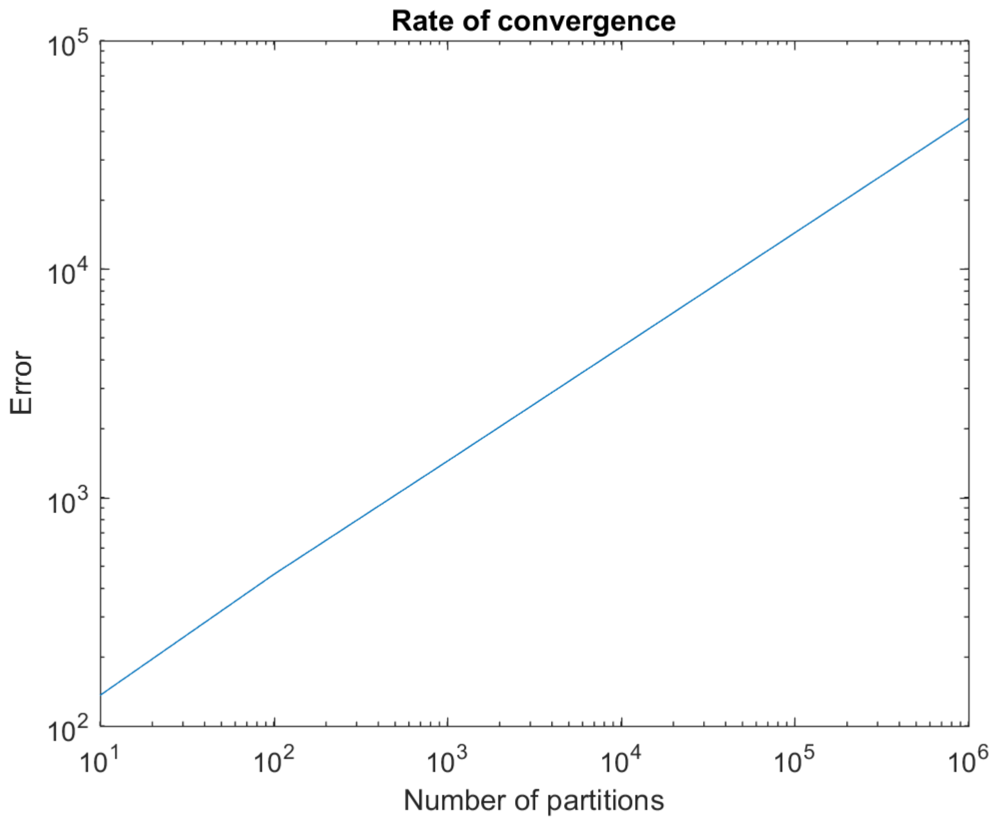


which can be compared to the analytical solution when using small steps.



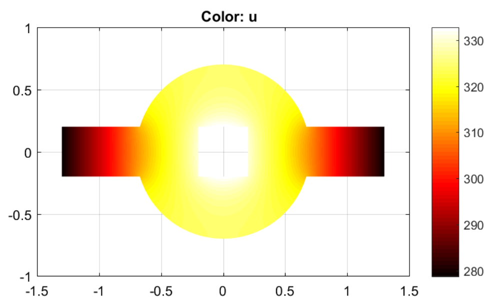
As one can see these two plots seems to correspond to each other fairly well, however how fast is the rate of convergence, roc? This was the second thing to be examined to do this one calculates the l_2 norm for different stepsizes and then use a logarithmic scaled plot. One can then compute the roc as

$roc = \frac{X_{i+1} - X_i}{h_{i+1} - h_i}$ so the change in the error divided by the change in stepsize between two meassurements, loglog plot gives us a linear plot as shown below. By comparing this with loglog plots for n^1, n^2, n^3,... one can see that its roc is 2.



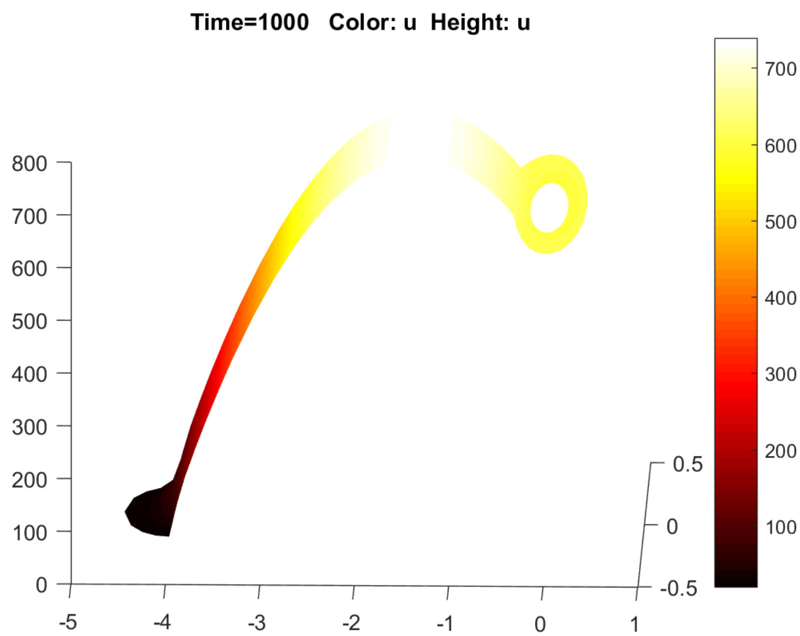
1 FEM in 2D

After having explored the FEM in 1D we'll now move into 2D with the help of matlabs pdetool which allows us to build complex geometries, defining boundary conditions and pdes. The first part was once again a steady state problem as shown below.



The boundary conditions was that we had a constant temperature flow of $200K * 2$ on the right and leftmost edges, the square in the middle had a constant flow of 800K and the outher edges were perfect insulated. As we can see however the highest temperature isn't 800K but rather a bit above 330 and nor were the lowest temperature 400 but instead 280K when at steady state. By changing it from steady state to an actual heat flow with initial temperature 290K the system found its steady state so fast that one couldn't see any change when trying different time steps. This shouldn't come as a surprise since 290K is close to both 290K and 330K which is the temperature max and min at steady state. However when using 0K as initial temperature one could see a difference between different time steps early but it once again found its steady state fairly quickly.

For the last part one was supposed to model a geometry of their own choice with criteriums that they weren't trivial and should be believable. The goemetry I modeled is shown below.



The leftmost semi-cirle has a boundary temperature which is constant at 20^oC while the iner-circle has a constant temperature of 600^oC with the rest being perfect isolated. As we can see the maximum temperature is not at the ring but rather a bit in on the bar, which is longer than it's wide.

2 Appendix

Matlab code: For heat equation in 1D:

```
n = 10000; % Amount of partitions
h = 10/n; %Length of each step
g1=40; %Boundary condition
g2=200 %Boundary condition
x = linspace(h, 10-h,n-1)'; % Vector for plotting
%Create b vector with boundary conditions
b1 = 20/n-g1*(-1/h+((h)/12));
bj = 20*ones(n-1,1)/n;
bn = 20/n+200*(1/h+((h)/12));
B1=[1,zeros(1,n-3),0]'; %For first element in b vector
B2=[0,ones(1,n-3),0]'; %For every element EXCEPT first and last in b vector
B3=[0,zeros(1,n-3),1]'; %For last element in b vector
B4 = B1.*b1;
B5 = B2.*bj;
B6 = B3.*bn;
B = [B4+B5+B6];
%Create K matrix
e = ones(n-1,1);
K = spdiags([(-e/h)+(h/6)((2*e)/h)+((2*h)/3)(-e/h)+(h/6)],-1:1, n-1, n-1);
%Solve linear system
ufem=K;
figure(1)
plot([0;x;10],[g1;ufem;g2], 'x-')
uh = [20 ufem' 200];
%Solve analytical derived solution with a small enough step to consider
%continous and plot solution
x = [0:0.0000001:10];
u = ((20.*exp(-x)).*(-exp(x)-exp(2.*x)+(9.*exp(2.*(x+5))+exp(x+20)-(9.*exp(10))+exp(20))))/(exp(20)-
1);
figure(2)
plot(x,u)
%l2 norm for rate of convergence
iterr = 6 %amount of iterattions
error=[1:1:iterr]
n = 1
%For loop that calculates the error for different step sizes where each
%step has 10i partitions
for i = 1:iterr
n= 10*n;
h = 10/n;
x = linspace(h, 10-h,n-1)';
b1 = 20/n-40*(-1/h+((h)/12));
bj = 20*ones(n-1,1)/n;
```

```
bn = 20/n+200*(1/h+((h)/12));
B1=[1,zeros(1,n-3),0]';
B2=[0,ones(1,n-3),0]';
B3=[0,zeros(1,n-3),1]';
B4 = B1.*b1;
B5 = B2.*bj;
B6 = B3.*bn;
B = [B4+B5+B6];
e = ones(n-1,1);
K = spdiags([(-e/h)+(h/6) ((2*e)/h)+((2*h)/3)
(-e/h)+(h/6)],-1:1, n-1, n-1);
ufem=K;
uh = [20 ufem' 200];
x = [0:h:10];
u = ((20.*exp(-x)).*(-exp(x)-exp(2.*x)+(9.*exp(2.*(x+5))+exp(x+20)-(9.*exp(10))+exp(20))))/(exp(20)-
1);
error(i) =sqrt(sum(u  uh). 2  h);
end
```
n = [$1010^2 10^3 10^4 10^5 10^6$];
$\%loglogplotoferror$
$figure(3)$
$loglog(n, error)$
$title(''Rateofconvergence'')$
$xlabel(''Numberofpartitions'')$
$ylabel(''Error'')$
$RoC = log(error(6)/error(5))/log(106/105)\%calculaterateofconvergence$
$Codefor2Dproblemsasgeneratedbymatlab$

For part 3:

```
% This script is written and read by pdetool and should NOT be edited.
% There are two recommended alternatives:
% 1) Export the required variables from pdetool and create a MATLAB script
% to perform operations on these.
% 2) Define the problem completely using a MATLAB script. See % http://www.mathworks.com/help/pde/exa
for examples of this approach.
function pdemodel
pdef ig,ax
=pdeinit;
pdetool('applcb',1);
set(ax,'DataAspectRatio',[1 1 1]);
set(ax,'PlotBoxAspectRatio',[1.5 1 1]);
set(ax,'XLim',[-1.5 1.5]);
set(ax,'YLim',[-1 1]);
set(ax,'XTickMode','auto');
set(ax,'YTickMode','auto');
pdetool('gridon','on');
```

```
% Geometry description:
pderect([-1.3 1.3 -0.20000000000000001 0.20000000000000001],'R1');
pdeellip(0,0,0.69999999999999996,0.69999999999999996,... 0,'E1');
pderect([-0.20000000000000001 0.20000000000000001 -0.20000000000000001 0.20000000000000001],'R2');
set(findobj(get(pdef ig,'Children'),'Tag','PDEEval'),'String','(R1+E1)-R2')
% Boundary conditions:
pdetool('changemode',0)
pdetool('removeb',[8 9 12 13 17 20 ]);
pdesetbd(14,...
'neu',...
1,...
'0',...
'0')
pdesetbd(13,...
'neu',...
1,...
'0',...
'0') pdesetbd(12,...
'neu',...

   1,...
'0',...
'0')
pdesetbd(11,...
'neu',...
1,...
'0',...
'0')
pdesetbd(10,...
'neu',...
1,...
'0',...
'0')
pdesetbd(9,...
'neu',...
1,...
'0',...
'0')
pdesetbd(8,...
'neu',...
1,...
'0',...
'0')
pdesetbd(7,...
'neu',...
1,...
```

```
'0',...
'0')
pdesetbd(6,...
'neu',...
1,...
'2',...
'800')
pdesetbd(5,...
'neu',...
1,...
'2',...
'800')
pdesetbd(4,...
'neu',...
1,...
'2',...
'800')
pdesetbd(3,...
'neu',...
1,...
'2',...
'800')
pdesetbd(2,...
'neu',...
1,...
'2',...
'400')
pdesetbd(1,...
'neu',...
1,...
'2',...
'400')
% Mesh generation:
setappdata(pdef ig,'Hgrad',1.3);
setappdata(pdef ig,'refinemethod','regular');
setappdata(pdef ig,'jiggle',char('on','mean',''));
setappdata(pdef ig,'MesherVersion','preR2013a');
pdetool('initmesh')
pdetool('refine')
pdetool('jiggle')
pdetool('refine')
% PDE coefficients:
pdeseteq(1,...
'3.0',...
'2.0',...
'580.0',...
```

```
'0.0',...
'0:10',...
'0.0',...
'0.0',...
'[0 100]')
setappdata(pdef ig,'currparam',...
.0 ';...
'2.0 ';...
'580.0';...
'0.0 '
)
% Solve parameters:
setappdata(pdef ig,'solveparam',...
char('0','2400','10','pdeadworst',...
'0.5','longest','0','1E-4','','fixed','Inf'))
% Plotflags and user data strings:
setappdata(pdef ig,'plotflags',[1 1 1 1 1 1 6 1 0 0 0 1 1 0 0 0 0 1]);
setappdata(pdef ig,'colstring','');
setappdata(pdef ig,'arrowstring','');
setappdata(pdef ig,'deformstring','');
setappdata(pdef ig,'heightstring','');
% Solve PDE:
pdetool('solve')
```

For part 4:

```
% This script is written and read by pdetool and should NOT be edited.
% There are two recommended alternatives:
% 1) Export the required variables from pdetool and create a MATLAB
script
% to perform operations on these.
% 2) Define the problem completely using a MATLAB script. See % http://www.mathworks.com/help/pde/exa
for examples
% of this approach.
function pdemodel
pdef ig,ax
=pdeinit;
pdetool('applcb',1);
set(ax,'DataAspectRatio',[1 1 1]);
set(ax,'PlotBoxAspectRatio',[1.5 1 1]);
set(ax,'XLim',[-1.5 1.5]);
set(ax,'YLim',[-1 1]);
set(ax,'XTickMode','auto');
set(ax,'YTickMode','auto');
pdetool('gridon','on');
% Geometry description:
pderect([-1.3 1.3 -0.20000000000000001 0.20000000000000001],'R1');
```

```
pdeellip(0,0,0.69999999999999996,0.69999999999999996,... 0,'E1');
pderect([-0.20000000000000001 0.20000000000000001 -0.20000000000000001 0.20000000000000001],'R2');
set(findobj(get(pdef ig,'Children'),'Tag','PDEEval'),'String','(R1+E1)-R2')
% Boundary conditions:
pdetool('changemode',0) pdetool('removeb',[8 9 12 13 17 20 ]);
pdesetbd(14,...
'neu',...
1,...
'0',...
'0')
pdesetbd(13,...
'neu',...
1,...
'0',...
'0')
pdesetbd(12,...
'neu',...
1,...
'0',...
'0')
pdesetbd(11,...
'neu',...
1,...
'0',...
'0')
pdesetbd(10,...
'neu',...
1,...
'0',...
'0')
pdesetbd(9,...
'neu',...
1,...
'0',...
'0')
pdesetbd(8,...
'neu',...
1,...
'0',...
'0')
pdesetbd(7,...
'neu',...
1,...
'0',...
'0')
pdesetbd(6,...
```

```
'neu',...
1,...
'2',...
'800')
pdesetbd(5,...
'neu',...
1,...
'2',...
'800')

    pdesetbd(4,...
'neu',...
1,...
'2',...
'800')
pdesetbd(3,...
'neu',...
1,...
'2',...
'800')
pdesetbd(2,...
'neu',...
1,...
'2',...
'400')
pdesetbd(1,...
'neu',...
1,...
'2',...
'400')
% Mesh generation:
setappdata(pdef ig,'Hgrad',1.3);
setappdata(pdef ig,'refinemethod','regular');
setappdata(pdef ig,'jiggle',char('on','mean',''));
setappdata(pdef ig,'MesherVersion','preR2013a');
pdetool('initmesh')
pdetool('refine')
pdetool('jiggle')
pdetool('refine')
% PDE coefficients:
pdeseteq(2,...
'3.0',...
'2.0',...
'580.0',...
'1.0',...
'0:10',...
```

'290',...
'0.0',...
'[0 100]')
setappdata(pdef ig,'currparam',...
.0 ';...
'2.0 ';...
'580.0';...
'1.0 '
)
% Solve parameters:
setappdata(pdef ig,'solveparam',...
char('0','2400','10','pdeadworst',...
'0.5','longest','0','1E-4','','fixed','Inf'))
% Plotflags and user data strings:
setappdata(pdef ig,'plotflags',[1 1 1 1 1 1 6 1 0 0 0 11 1 0 0 0 0 1]);
setappdata(pdef ig,'colstring','');
setappdata(pdef ig,'arrowstring','');
setappdata(pdef ig,'deformstring','');
setappdata(pdef ig,'heightstring','');
For part 5:
% This script is written and read by pdetool and should NOT be edited.
% There are two recommended alternatives: % 1) Export the required variables from pdetool and create a MATLAB script % to perform operations on these. % 2) Define the problem completely using a MATLAB script. See % http://www.mathworks.com/help/pde/examples/index.html for examples % of this approach. function pdemodel
pdef ig,ax
=pdeinit;
pdetool('applcb',1);
set(ax,'DataAspectRatio',[1 0.5 1]);
set(ax,'PlotBoxAspectRatio',[3 2 1]);
set(ax,'XLim',[-5 1]);
set(ax,'YLim',[-1 1]);
set(ax,'XTick',[-5,...
-4,...
-3,...
-2,...
-1,...
0,...
1,...
2,...
3,...
]);
set(ax,'YTickMode','auto');
pdetool('gridon','on');
% Geometry description:

```
pdeellip(0,0,0.40000000000000002,0.40000000000000002,... 16 0,'E1');
pdeellip(0,0,0.20000000000000001,0.20000000000000001,... 0,'E2');
pderect([-4 -0.34574841883345053 -0.20000000000000001 0.20000000000000001],'R1');
pdeellip(-4,0,0.5,0.20000000000000001,... 0,'E3');
set(findobj(get(pdef ig,'Children'),'Tag','PDEEval'),'String','(E1+R1+E3)-E2')
% Boundary conditions:
pdetool('changemode',0)
pdesetbd(19,...
'dir',...
1,...
'1',...
'20')
pdesetbd(16,...
'dir',...
1,...
'1',...
'20')
pdesetbd(15,...
'dir',...
1,...
'1',...
'600')
pdesetbd(14,...
'dir',...
1,...
'1',...
'600')
pdesetbd(13,...
'dir',...
1,...
'1',...
'600')
pdesetbd(12,...
'dir',...
1,...
'1',...
'600')
pdesetbd(10,...
'neu',...
1,...
'0',...
'0')
pdesetbd(9,...
'neu',...
1,...
'0',...
```

```
'0')
pdesetbd(8,...
'neu',...
1,...
'0',...
'0')
pdesetbd(7,...
'neu',...
1,...
'0',...
'0')
pdesetbd(4,...
'neu',...
1,...
'0',...
'0')
pdesetbd(2,...
'neu',...
1,...
'0',...
'0')
% Mesh generation:
setappdata(pdef ig,'Hgrad',1.3);
setappdata(pdef ig,'refinemethod','regular');
setappdata(pdef ig,'jiggle',char('on','mean',''));
setappdata(pdef ig,'MesherVersion','preR2013a');
pdetool('initmesh')
pdetool('jiggle')
pdetool('refine')
% PDE coefficients:
pdeseteq(2,...
'1.0',...
'0.0',...
'200.0',...
'1.0',...
'0:10',...
'0.0',...
'0.0',...
'[0 100]')
setappdata(pdef ig,'currparam',...
.0 ';...
18
'0.0 ';...
'200.0';...
'1.0 '
)
```

```
% Solve parameters:
setappdata(pdef ig,'solveparam',...
char('0','8016','10','pdeadworst',...
'0.5','longest','0','1E-4','','fixed','Inf'))
% Plotflags and user data strings:
setappdata(pdef ig,'plotflags',[1 1 1 1 1 1 6 1 0 0 0 11 1 0 0 0 0 1]);
setappdata(pdef ig,'colstring','');
setappdata(pdef ig,'arrowstring','');
setappdata(pdef ig,'deformstring','');
setappdata(pdef ig,'heightstring','');
% Solve PDE:
pdetool('solve')
```