Project 3

Numerical Methods for Differential Equations

Isabelle Frodé & Sara Enander
Date: December 18 2019

1 The Diffusion Equation

This first part of the project aims to construct two solvers for the diffusion equation, stated in Equation 1, using two diefferent methods.[1]

$$\begin{aligned} u_t &= u_{xx} \\ u(t,0) &= u(t,1) = 0 \\ u(0,x) &= g(x) \end{aligned} \tag{1}$$

The equation was first solved numerically using the technique called Method of Lines. Applied to this problem it means that one dimension is discretized, using Equation 2, on a grid [0,1] where $\Delta x = 1/(N+1)$, N is the number of interior points. The initial condition, g(x), is sampled on the grid at t=0. [1]

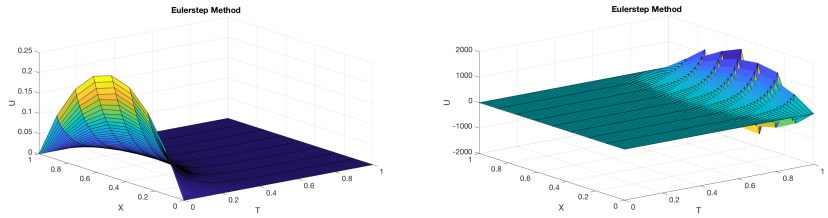
$$\dot{u} = T_{\Delta x} u \tag{2}$$

$T_{\Delta x}$ is the Toeplitz symmetric tridiagonal matrix approximating $\partial^2/\partial x^2$ to O(h^2). Using the Euler Method, Equation 2 was solved numerically in Matlab by creating a solver, which is presented below.

```
function unew = Eulerstep(Tdx, uold, dt)
yprime=Tdx*uold;
unew = uold+dt.*yprime;
end
```

The numerical solution was then plotted over the t, x plane. The result is presented in Figure 1 (a). Initial values was set to $g(x) = -x(x - t_{end})$, the number of interial grid points on the t-axis M to $M = 128$ and number of interial grid points on the x-axis N to $N = 7$. Here the CFL number, $\Delta t/\Delta x^2$, was exactly equal to 0.5.

The size of the time steps was then modified by changing $M = 128$ to $M = 115$. Figure 1 (b) shows the plot for the $M = 115$-case, where $\Delta t/\Delta x^2$=0.5565 which means that the CFL conditions are being violated, and the solution is unstable. For smaller M, the solution explodes quickly and goes to infinity.



(a) $M = 128$ and $N = 7$ **(b)** $M = 115$ and $N = 7$

Figure 1: Solutions using the Eulerstep Method, N is the number of interior grid points on the x-axis and M the number of interior grid points on the t-axis

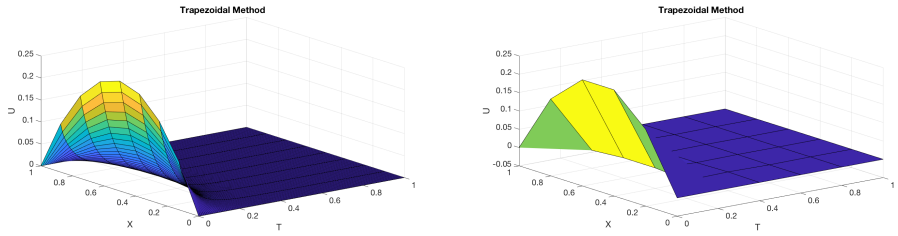
For the next part of this section, another method was used to solve the diffusion equation numerically, the Crank-Nicolson method. The solution is obtained by applying the trapezoidal rule to the equation, shown in Equation 3.[1]

$$u^{n+1} = u^n + \frac{\Delta t}{2}(T_{\Delta x}u^n + T_{\Delta x}u^{n+1}) \tag{3}$$

A new function, presented below, was then implemented in Matlab.

```
function unew = TRstep(Tdx, uold, dt)
I = speye(size(Tdx));
unew = ((I-dt.*Tdx)\uold(:,end));
end
```

The same diffusion equation as before was then solved using the new function with the same initial- and boundary conditions as when the CFL condition previously was violated (Figure 1 (b)). The plot obtained with the Crank-Nicolson method is presented in Figure 2 (a) and shows no signs of instability problems. The plot verifies that the Crank-Nicolson method obtains stable solutions where the Euler Method fails to do so. Figure 2 (b) shows that it is actually possible to run the program with much larger time steps with this method than the previous. Figure 2 (b) is showing the plot when the number of internal grid points on the t-axis are changed to $M = 4$ (N=4).



(a) $M = 115$ and $N = 7$ **(b)** $M = 5$ and $N = 4$

Figure 2: Solutions using the Trapezoidal Method, N the number of interial grid points on the x-axis and M the number of interial grid points on the t-axis

2 The Advection Equation

In this section a Lax-Wendroff solver was created to solve the linear advection equation[1], presented in Equation 4.

$$u_t + au_x = 0 \tag{4}$$

The solution was stored in vectors with N points, on an equidistant grid, at time t. Periodic boundary values was applied: $u(t, 0) = u(t, 1)$. This means that the solution get represented over the entire grid, including boundaries. The initial condition was set to

$$g(x) = e^{-100(x-0.5)^2} \tag{5}$$

which satisfies $g(0) = g(1)$ and $g'(0) = g'(1)$.

The Lax-Wendroff scheme is a second order method. To obtain this scheme an approximation of a time step using a Taylor series expansion must be done:

$$u(t + \Delta t, x) = u(t, x) + \Delta t u_t + \frac{\Delta t^2}{2!} u_{tt} + O(\Delta t^3) \tag{6}$$

The differential equation gives expressions for u_t and u_{tt}: $u_t = -au_x$ and $u_{tt} = -a^2 u_{xx}$ and these were substituted so that the equation only contains space derivatives:

$$u(t + \Delta t, x) = u(t, x) - a\Delta t u_x + \frac{a^2 \Delta t^2}{2!} u_{xx} + O(\Delta t^3) \tag{7}$$

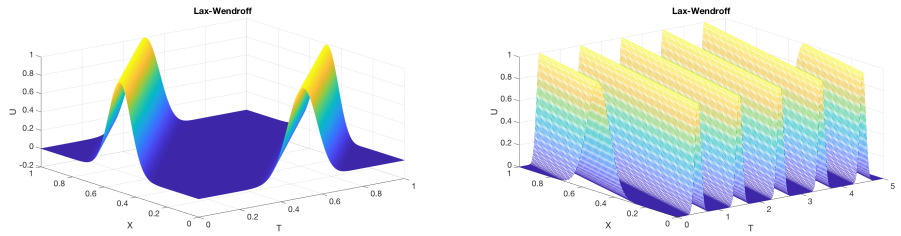
Now the Lax-Wendroff scheme can be written as:

$$u_j^{n+1} = \frac{a\mu}{2}(1 + a\mu)u_{j-1}^n + (1 - a^2\mu^2)u_j^n - \frac{a\mu}{2}(1 - a\mu)u_{j+1}^n \tag{8}$$

with μ=Δt/Δx. The solver is presented below:

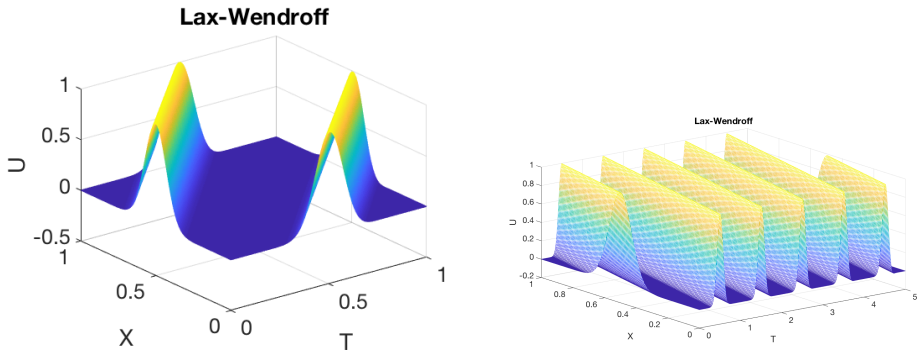
```
function unew = LaxWen(uold, amu)
N = length(uold);

sub = 1./2.*amu.*(1+amu).*ones(1, N-1);
main = (1-amu.^2).*ones(1, N);
sup = 1./2.*(-1).*amu.*(1-amu).*ones(1, N-1);
Tdx = (diag(sub,-1) + diag(main,0) + diag(sup,1));

unew = Tdx*uold;

end
```

The Lax-Wendroff solver was run for t_{end}=1 and t_{end}=5, and worked for both positive and negative values of a. The solutions are plotted for CFL number ($a\Delta t/\Delta x$) 1 in Figure 3 and CFL number 0.9 in Figure 4, both with positive values of a. Figure 5 verifies that the Lax-Wendroff solver works for negative values of a, showing the plot of the solution when $a = -1$. The plot is, as expected, similar to the plot in Figure 4 (b) but with a differnece due to the change in flow direction.



(a) t_{end}=1 **(b)** t_{end}=5

Figure 3: Solutions using the Lax-Wendroff solver, with CFL number 1



(a) t_{end}=1 **(b)** t_{end}=5

Figure 4: Solutions using the Lax-Wendroff solver, with CFL number 0.9

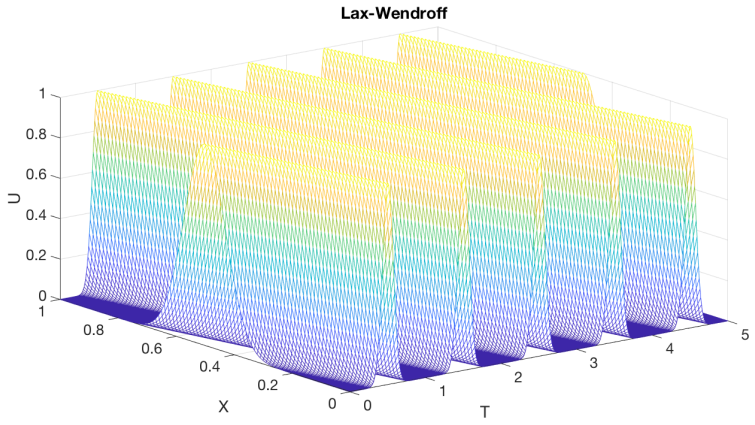
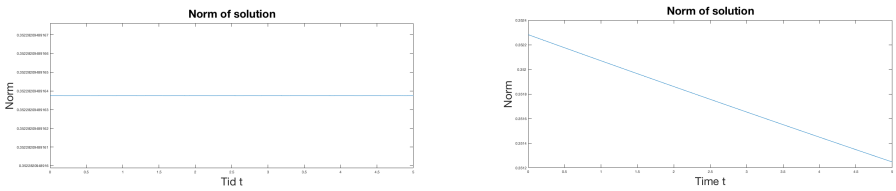


Figure 5: The solution of the advection equation using the Lax-Wendroff solver, with negative a

The L_2-norm for the solution was also studied. It was shown that the L_2-norm remained constant in time when $a\Delta t/\Delta x$=1. This means that the amplitude is not damped, which is expected since there is no diffusion in the advection equation. However, when $a\Delta t/\Delta x$=0.9 the norm is slowly decreasing, which corresponds to the amplitud being damped. This is not physical, since the advection equation is not supposed to contain any diffusion. Therefore it is important to make sure that the CFL is set to 1. The L_2-norms are plotted in Figure 6.



(a) Norm for $a\Delta t/\Delta x$=1 **(b)** Norm for $a\Delta t/\Delta x$=0.9

Figure 6: L^2-norm of the solution as a function of time.

3 The Convection-Diffusion Equation

This section investigates a solution method to the linear convection-diffusion equation, stated in Equation 9.

$$u_t + au_x = du_{xx} \tag{9}$$

The constant a is representing the convection velocity and d the diffusivity. The boundary conditions are u(t,0) = u(t,1). g(x) was set to $g(x) = e^{-100(x-0.5)^2}$, which means that $g(0) = g(1)$ and $g'(0) = g'(1)$.

The trapezoidal method was furthermore used for the time stepping. The method could be used since the problem is stiff, due to the diffusivity. To solve the problem equation 10 was used.[1]

$$\dot{u} = (d \cdot T_{\Delta x} - a \cdot S_{\Delta x} u) \tag{10}$$

$T_{\Delta x}$ is symmetric with negative real eigenvalues and $S_{\Delta x}$ is skew-symmetric with imaginary eigenvalues. The differential operator has negative real eigenvalues, hence the discretization should to, for the solution to be balanced between convection and diffusivity. This is described mathematically as the Péclet number $Pe = |a/d|$. The condition could be written as $Pe\Delta x < 2$ which is called the mesh Péclet number. The CFL conditions could moreover roughly be expressed as shown in Equation 11.

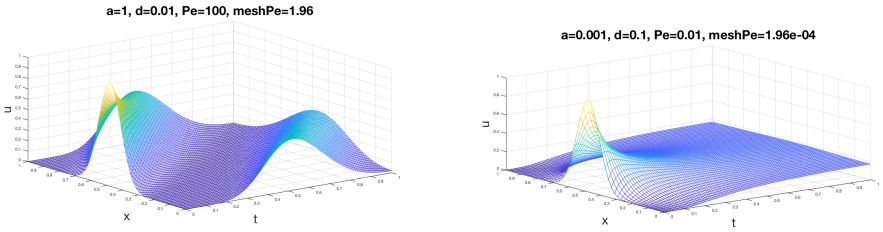
$$d\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{11}$$

A solver for the convection-diffusion equation was constructed on [0,1] for t on the interval [0,1]. The solver itself is presented below.

```
function unew = convdif(uold, a, d, dx, dt)
N = length(uold);
sub = (d./(dx.^2)+(a./(2.*dx))).*ones(1, N-1);
main = -2.*(d./(dx.^2)).*ones(1, N);
sup = (d./(dx.^2)-(a./(2.*dx))).*ones(1, N-1);
Tdx = (diag(sub,-1) + diag(main,0) + diag(sup,1));
Tdx(1, N) = (d./(dx.^2)+(a./(2.*dx)));
Tdx(N, 1) = (d./(dx.^2)-(a./(2.*dx)));
unew = TRstep(Tdx, uold, dt);
end
```

The convection a and the diffusivity d was varied. It is important that $d \geq 0$ since the diffusivity as a fenomena always is positive; it measures the rate of which fluids, particles or heat spread, it could not go backwards in time. If d would be ≤ 0 it would go against one of the most fundamental laws of physics.

The solver was tested for different Péclet numbers to observe how the solution change when the balance between diffusivity and convection was varied. Figure 7 shows the solution when a) convection is dominant, b) diffusivity is dominant.



(a) Péclet number $Pe = 100$ **(b)** Péclet number $Pe = 0.01$

Figure 7: Numerical solutions to the Convection-Diffusion Equation, varied Peclét- and mesh Péclet numbers

Figure 7 (b) clearly shows, after comparing with the results in section 1 of this report, that when the Peclét number is low (≤ 1) the diffusivity is dominant, as expected. After looking at section 2 of this report, Figure 7 (a) shows that convection is dominant. The numerical solutions to the convection-diffusion equation are in accordance with the theory, hence the solver works as expected.

4 The Viscous Burgers Equation

In this chapter the nonlinear viscous Burgers equation was studied, showed in equation 12[1]:

$$u_t + uu_x = d \cdot u_{xx} \tag{12}$$

The viscous Burgers equation is a convection-diffusion equation. In the first step to solve this, a Taylor expansion of a time step was made:

$$u(t + \Delta t, x) \approx u(t, x) + \Delta t u_t + \frac{\Delta t^2}{2!} u_{tt} \tag{13}$$

The next step was to replace the time derivatives with space derivatives. These were calculated from the inviscid Burgers equation[1], showed in equation 14:

$$u_t = -uu_x \tag{14}$$

Equation 14 was derived to obtain an expression for u_{tt}, which was calculated to be $u_{tt} = 2uu_x^2 + u^2 u_{xx}$ and the space derivative was then approximated symmetrically:

$$\begin{aligned} u_x &\approx \frac{1}{\Delta x}\left(\frac{1}{2}u_{l+1}^n - \frac{1}{2}u_{l-1}^n\right) \\ u_{xx} &\approx \frac{1}{\Delta x^2}\left(u_{l-1}^n - 2u_l^n + u_{l+1}^n\right) \end{aligned} \tag{15}$$

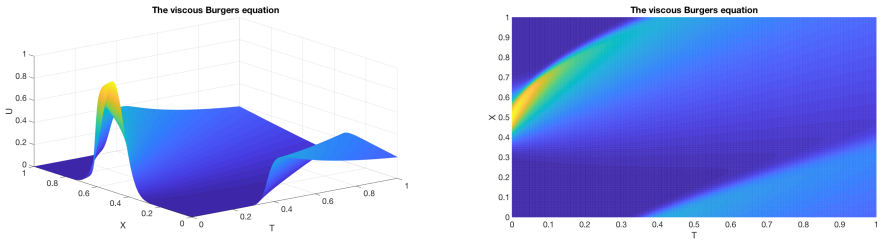
The scheme was then implemented to a MATLAB function, which takes one time step, the script is presented below.

```
function unew = LW(uold, dt, Tdx1, Tdx2)

unew=uold-dt.*uold.*(Tdx1*uold)+dt.^2.*uold.*((Tdx1*uold).^2+uold.*(Tdx2*uold));

end
```

Here the nonlinear convection term is taken care of. The diffusion term had to be added as well, which was done using the trapezoidal rule. The total expression is shown in equation 16[1]:

$$unew = \left(\left(I - d\frac{\Delta t}{2} T_{\Delta x} \right)^{-1} \cdot LW(uold) + d\frac{\Delta t}{2} T_{\Delta x} \cdot uold \right) \tag{16}$$

This was then used to solve the equation, with N=300, M=1000, d=0.01, a=1, shown in figure 8:



(a) Solution **(b)** Characteristics

Figure 8: Solution to the Viscous Burgers equation, with charecteristics.

Figure 8 shows waves with steep gradients, without the method going unstable. Figure 8(b) shows the characteristics of the solution and it is very clear that the equation is nonlinear, since the characteristics are not parallel. When characteristics collide it means that shock waves are formed [2].

5 Project summary

In this project, partial differential equations were studied. The first part of the project consisted of constructing a solver for the diffusion equation, followed by a solver for the advection equation. The last part of the project aimed to put these solvers together, creating solvers for linear and nonlinear partial differential equations. The project taught us that it is possible to solve a problem which at first sight seems impossible, like the Viscous Burgers Equation, by breaking the problem down. In the end of the project it all made a lot of sense to us to put the solvers together, in combination with some, not too complicated, mathematics. We would like to thank Tony

for helping us by confirming some of our results and plots and also acknowledge our classmates Louise Karsten, Felicia Segui, Ebba Rickard and Anna Sjerling for discussing problems and sharing ideas.

References

[1] T Stillfjord, G Söderlind, Project manual: *Project 3 in FMNN10 and NUMN12*, LTH, 2019

[2] T Stillfjord, G Söderlind, Lecture notes: *Numerical Methods for Differential Equations, Chapter 6: PDEs- Waves and hyperbolics*, LTH, 2019

6 Appendix 1

6.1 Eulerstep

```
function unew = Eulerstep(Tdx, uold, dt)
yprime=Tdx*uold;
unew = uold+dt.*yprime;
end
```

6.2 Testfunktion 1.1

```
M = 190;
N = 90;
tend = 1;
dt= tend./M;
dx= 1./(N+1);
g = @(x) -x.*(x-tend);
xx=linspace(0,1,N+2)';
tt=linspace(0,tend,M+1);
uoldlong = g(xx);
uold = uoldlong(2:N+1);
uMatrix = zeros(N+2,M+1);
uMatrix(2:N+1,1) = uold;
[T,X]=meshgrid(tt,xx);

sub = ones(1, N-1);
main = -2.*ones(1, N);
sup = ones(1, N-1);
Tdx = 1./(dx.^2).*(diag(sub,-1) + diag(main,0) + diag(sup,1));
a=1;
amu=a.*(dt/dx^2);

for i=1:M
    unew = Eulerstep(Tdx, uold, dt);
    uMatrix(2:N+1,i+1) = unew;
    uold = unew;
end

surf(T,X,uMatrix)
```

6.3 TRstep

```
function unew = TRstep(Tdx, uold, dt)
I = speye(size(Tdx));
```

```
unew = ((I-dt.*Tdx)\uold(:,end));
end
```

6.4 Testfunktion 1.2

```
M = 128;
N = 90;
tend = 1;
dt= tend./M;
dx= 1./(N+1);
g = @(x) -x.*(x-tend);
xx=linspace(0,1,N+2)';
tt=linspace(0,tend,M+1);
uoldlong = g(xx);
uold = uoldlong(2:N+1);
uMatrix = zeros(N+2,M+1);
uMatrix(2:N+1,1) = uold;
[T,X]=meshgrid(tt,xx);
a=1;

sub = ones(1, N-1);
main = -2.*ones(1, N);
sup = ones(1, N-1);
Tdx = 1./(dx.^2).*(diag(sub,-1) + diag(main,0) + diag(sup,1));
for i=1:M
    unew = TRstep(Tdx, uold, dt);
    uMatrix(2:N+1,i+1) = unew;
    uold = unew;
end
uMatrix
surf(T,X,uMatrix)
```

7 Appendix 2

7.1 Testfunktion 2

```
M = 100;
N = 100;
tend = 1;
dt= tend./M;
dx= 1./N;
a=1;
amu=a.*(dt./dx);
```

```
g = @(x) exp(-100.*(x-0.5).^2);

xx=linspace(0,1,N+2)';
tt=linspace(0,tend,M+1);

uoldlong = g(xx);
uold = uoldlong(2:N+1);

uMatrix = zeros(N+2,M+1);
uMatrix(2:N+1,1) = uold;

[T,X]=meshgrid(tt,xx);

for i=1:M
    unew = LaxWen(uold, amu);
    uMatrix(2:N+1,i+1) = unew;
    uold = unew;
end

for i=1: M+1
rmsvector(i)= rms(uMatrix(:, i));
end

figure(3)
surf(T, X, uMatrix)

figure(2)
plot(tt, rmsvector)
```

8 Appendix 3

```
M = 100;
N = 200;
tend = 1;
dt= tend./M;
dx= 1./N;
d=0.1;
a=10;
g = @(x) exp(-100.*(x-0.5).^2);

xx=linspace(0,1,N+1)';
tt=linspace(0,tend,M+1);

uoldlong = g(xx);
uold = uoldlong(2:N+1);
uMatrix = zeros(N+1,M+1)

uMatrix(1:N,1) = uold;

[T,X]=meshgrid(tt,xx);

for i=1:M
    unew = convdif(uold, a, d, dx, dt);
    uMatrix(1:N,i+1) = unew;
    uold = unew;
end

radett = uMatrix(1, :);
uMatrix(N+1, :) = radett;
uMatrix;

Pe= abs(a/d)
meshPe= dx*Pe

figure(3)
mesh(T, X, uMatrix)
```

9 Appendix 4

9.1 Testfunction 4

```
%% Task 4.1
clear all
a=1;
d=0.01;
N=300;
M=1000;
tend=1;
dx=1/(N);
dt=tend/(M);
g=@(x) exp(-100*(x-0.5).^2);

xx = linspace(0,1,N+1)';
tt = linspace(0,tend,M+1);
[T,X]=meshgrid(tt,xx);

uoldlong=g(xx);
uold=uoldlong(1:N);
umatrix=zeros(N+1,M+1);
umatrix(1:N,1)=uold;

I=eye(length(uold),length(uold));
sub1=-1*ones(1,N-1);
sup1=ones(1,N-1);
main1=zeros(1,N);
Tdx1 = 1./(2*dx).*(diag(sub1,-1) + diag(main1,0) + diag(sup1,1));
Tdx1(1,length(uold))=- 1./(2*dx);
Tdx1(length(uold),1)= 1./(2*dx);

sub2=ones(1,N-1);
main2=-2.*ones(1,N);
sup2=ones(1,N-1);
Tdx2 = 1./(dx.^2).*(diag(sub2,-1) + diag(main2,0) + diag(sup2,1));
Tdx2(1,length(uold))= 1./(dx.^2);
Tdx2(length(uold),1)= 1./(dx.^2);

 for i=1:M
       unew=(I-d*dt./2.*Tdx2)\(LW(uold,dx,dt, Tdx1, Tdx2)+d*dt./2.*Tdx2*uold);
       umatrix(1:N,i+1)=unew;
       uold=unew;
 end
```

```
 umatrix(N+1,:)=umatrix(1,:);

figure(4)
mesh(T,X,umatrix)
xlabel('T','fontsize', 25)
ylabel('X','fontsize', 25)
zlabel('U','fontsize', 25)
title('The viscous Burgers equation')
ax = gca;
ax.FontSize = 25;
grid on
```