Implementation

# Task

**Create a wing-profile generator program that is also capable for creating simplified mesh for the generated geometry.**

On the consultation we agreed, that representing the wing geometry as points in 3D is sufficient, and that the user should be able to give proportions of the end and start of the wing, so the program could also generate the leading and trailing edges of the wing.

# Introduction

I chose the NACA 4-digit wing profile for the base of implementation. There are of course much more complex models describing wing profiles, but I thought this will be enough for this example, because it does not require too much explanation, and it provides nice results.

## NACA 4

The NACA-4 defines the wing with the following three parameters

* The first digit (**M**) describes the maximum the maximum camber as percentage of the chord[[1]](#footnote-1).
* The second digit (**P**) gives the distance of the maximum camber from the leading edge in tens of percents of the chord.
* The last two digits (**XX**) describe the maximum thickness of the wing as percent of the chord.

So for example, the NACA 4410 airfoils parameters as percentages of the chord:

* **Camber:** 4%
* **Max camber location:** 40%
* **Maximum thickness:** 10%

## Used Technology

### MATLAB

For the realization of this wing profile I used MATLAB numerical computing programming language and environment for the calculations, and its App Designer feature, to implement the user interface.

### GIT

I used Git distributed version control system, to back up my development, so I don’t lose my progress in case of a sudden hardware failure, and also I can easily revert to a previous state, if I took a wrong turn in the development.

# Implementation

## Calculations

I planned to do an application with a user interface, where the user can give the NACA parameters and the wing start-end proportions, and after pressing a button, the application plots the profile, the edge, and the profile combined with the edge in 3D.

For starters, I developed in simple MATLAB scripts, I left the App Designer for later, until I had all the calculations and plots figured out.

First I implemented the calculations for the NACA profile. For this, the following equations were needed.

**Half thickness of the wing**

* **yt** is the half thickness at a given **x** value.
* **x** is the position along the chord from 0 to 1.
* **t** is the maximum thickness, coming from the last two digits of the NACA 4-digit model

**Mean camber**

* **yc** is the camber at a given **x** value.
* **m** is the maximum camber.
* **p** is the location of the maximum camber.

**Location of upper and lower points of the profile**

The thickness needs to be applied perpendicular to the camber line, so we need to calculate the angle of the deviation compared to the horizontal plane, and use that angle to determine the upper and lower points.

This angle can be calculated with the gradient of **yc** and **x**.

I put the four digit parameters into variables, and created a variable for the resolution for the calculation, called **numberOfPoints**. The calculations seen above are realized in the following function[[2]](#footnote-2):

**[xyUpper, xyLower, xyCamber] = calculate\_2d\_profile(M, P, XX, numberOfPoints)**

I created three vectors to hold the coordinates of the camber line, the upper and lower profile. Then used a for loop, to iterate through the **x** values, so I can calculate the values for each one.

for i=1:1:numberOfPoints

% derivative of yc

dyc=zeros(numberOfPoints, 1);

xi=xyCamber(i,1);

yCamberi=0;

% first: 0 <= x <= P

if(0 <= xi && xi < P)

yCamberi=(M/P^2)\*((2\*P\*xi)-(xi\*xi));

dyc(i)=2\*M/P^2\*(P-xi);

% second: P <= x <= numberOfPoints

elseif (P <= xi && xi <= numberOfPoints)

yCamberi=(M/(1-P)^2)\*(1-(2\*P)+(2\*P\*xi)-(xi\*xi));

dyc(i)=2\*M/(1-P)^2\*(P-xi);

end

xyCamber(i,2)=yCamberi;

% calculate half-thickness at every x

% a0=0.2969; a1=0.1260; a2=0.3516; a3=0.2843; a4=0.1015;

yHalfThickness(i)=5\*(XX/100)\*(0.2969\*sqrt(xi)-0.1260\*xi-0.3516\*xi^2+0.2843\*xi^3-0.1015\*xi^4);

% calculate angle of deviation

phi(i)=atan(dyc(i));

% calculate upper and lower part of the airfoil

% xUpper=x-yt\*sin(phi) and xLower=x+yt\*sin(phi)

% yUpper=yc+yt\*cos(phi) and yLower=yt-yt\*cos(phi)

xyUpper(i,1)=xi-yHalfThickness(i)\*sin(phi(i)); xyUpper(i,2)=yCamberi+yHalfThickness(i)\*cos(phi(i));

xyLower(i,1)=xi+yHalfThickness(i)\*sin(phi(i)); xyLower(i,2)=yCamberi-yHalfThickness(i)\*cos(phi(i));

end

After this, I plotted the three resulting coordinate vectors on the same figure. I use a black background, to create contrast for better visibility. I put the plotting in a separate function.

**plot\_2d\_profile(xyCamber, xyUpper, xyLower, numberOfPoints)**

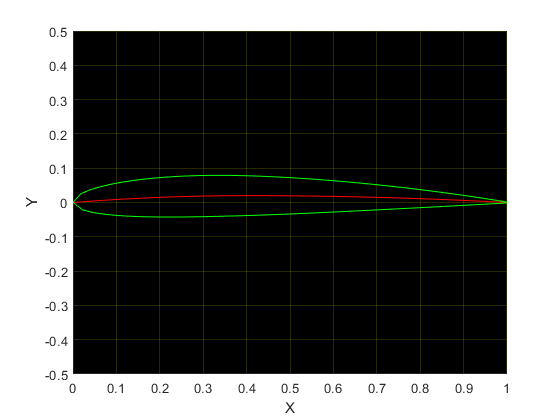


Figure 1 - Plot of profile, NACA-2412

For the next step, I wanted to create a three dimensional version of this profile, first, without the edges, to try out the plotting in 3D. To do this, I defined a length for the wing, which is 10, and created a vector called **zValues** containing **numberOfPoints** number of values equally spaced from 0 to 10. Then I plotted them in two ways, as separate points, and as a triangle mesh.

**plot\_3d\_profile\_points(zValues, wing\_length, xyUpper, xyLower, numberOfPoints)**

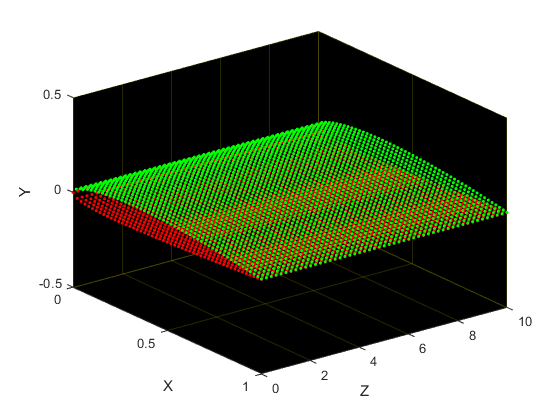


Figure 2 - 3D profile with points

**plot\_3d\_profile\_mesh(zValues, wing\_length, xyUpper, xyLower, numberOfPoints)**

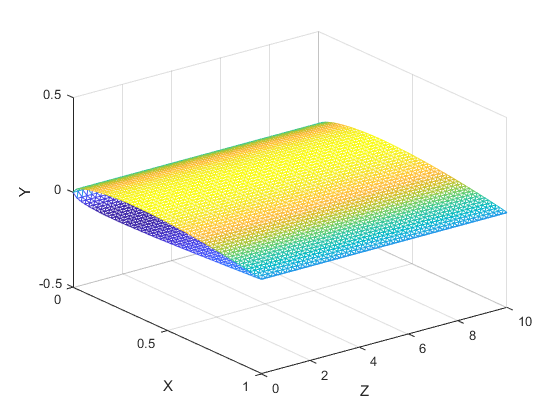


Figure 3 - 3D profile with trimesh

Plotting with mesh was much faster than plotting with the points, and it could be rotated much faster using the mouse, so I planned to use that for the 3D wing with edge.

In the next step I calculated the edges of the wing. I created two vectors **edgeFront** and **edgeBack**, and given the start-end ratio of the wing, I increased the x values from 0 to (1-start-end-ratio) for edgeFront, proportionally as I go along the z axis, and decrease them from 1 for edgeBack, the same way. It’s realized in the function seen below.

**[edgeFront, edgeBack] = calculate\_edges\_of\_wing(numberOfPoints, wing\_length, startEndProportions)**

halfOfStartEndDiff = (1 - startEndProportions)/2;

xpoints = linspace(0,wing\_length,numberOfPoints);

edgeFront = zeros(numberOfPoints,1);

edgeBack = zeros(numberOfPoints,1);

for i=1:1:numberOfPoints

edgeFront(i) = 0 + halfOfStartEndDiff\*(xpoints(i)/wing\_length);

edgeBack(i) = 1 - halfOfStartEndDiff\*(xpoints(i)/wing\_length);

end

The resulting edge with 0.5 start-end ratio.

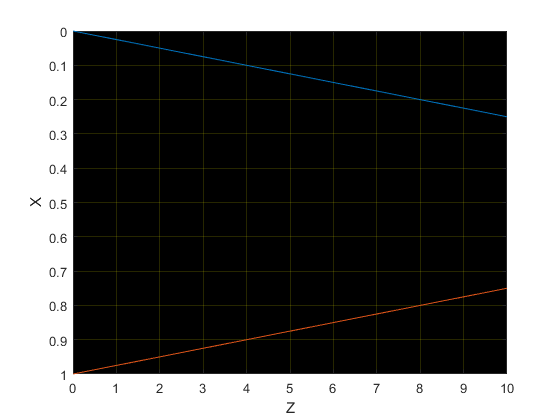


Figure 4 - Edge of wing

The way it’s represented is, that the vertical axis is **x**, the horizontal is **z**, in the 3D coordinate space.

The next step was to put together the 3D wing with this edge. For this, I created a nested for loop, with the outer loop I’m traversing through the edge values I created in the previous step, and in the inner loop I calculate the deviation from the profile considering the edge value in that iteration. Then I’m adding these calculated points to a three dimensional vector, for later use in the plotting. The function realizing this is the following.

**[xyUpperCurved, xyLowerCurved] = calculate\_3d\_wing\_edged(xyUpper, xyLower, edgeFrontX, edgeBackX, numberOfPoints)**

xyUpperCurved=zeros(numberOfPoints,numberOfPoints,2);

xyLowerCurved=zeros(numberOfPoints,numberOfPoints,2);

xyUpperTemp=xyUpper;

xyLowerTemp=xyLower;

for j=1:1:numberOfPoints

for i=1:1:numberOfPoints

if(xyUpper(i,1)<.5)

xyUpperTemp(i,1) = ((xyUpper(i,1) + (1-(xyUpper(i,1)/.5))\*edgeFrontX(j)));

else

xyUpperTemp(i,1) = -(xyUpper(i,1) + (1-(xyUpper(i,1)/.5))\*edgeBackX(j)) + 1;

end

if(xyLower(i,1)<.5)

xyLowerTemp(i,1) = ((xyLower(i,1) + (1-(xyLower(i,1)/.5))\*edgeFrontX(j)));

else

xyLowerTemp(i,1) = -(xyLower(i,1) + (1-(xyLower(i,1)/.5))\*edgeBackX(j)) + 1;

end

xyUpperCurved(j,i,1) = xyUpperTemp(i,1);

xyUpperCurved(j,i,2) = xyUpperTemp(i,2);

xyLowerCurved(j,i,1) = xyLowerTemp(i,1);

xyLowerCurved(j,i,2) = xyLowerTemp(i,2);

end

end

**plot\_3d\_wing\_edged(zValues, xyUpperCurved, xyLowerCurved, wing\_length, numberOfPoints)**

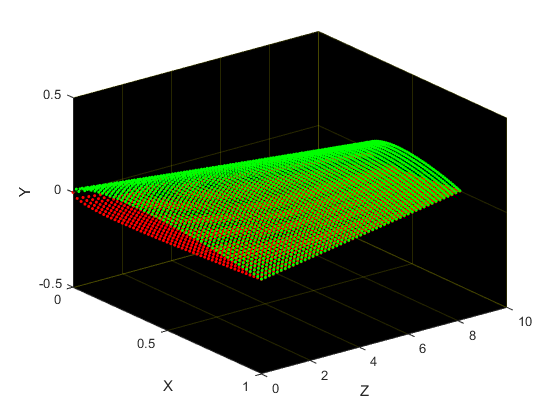


Figure 5 - 3D wing with edge

As you can see on the image below, the points are spread evenly across the axes, and the end the edges are at 2.5 and 7.5, so the half ratio shows nicely as well.

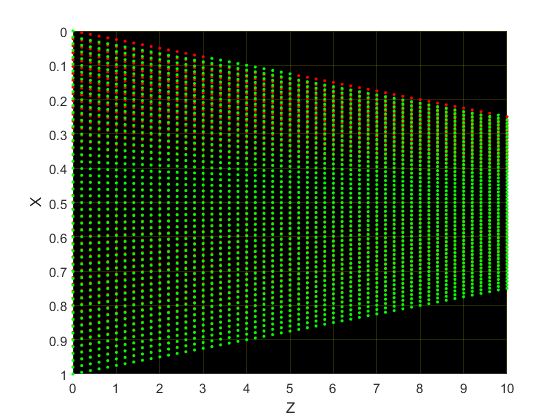
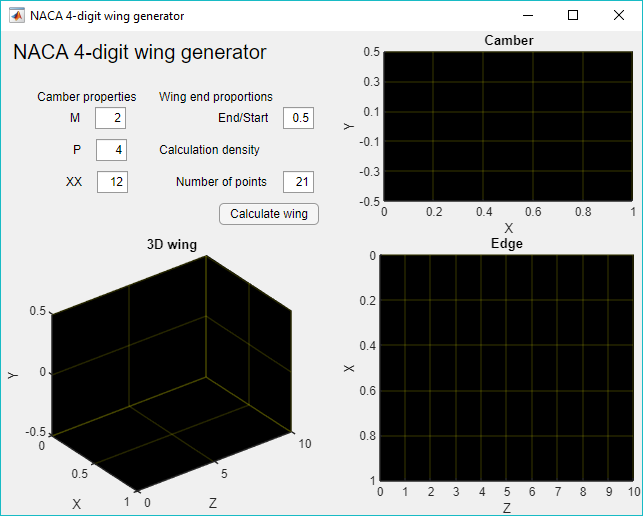


Figure 6 - 3D wing with edge seen from above

I wanted to use triangulate mesh with this plot also, but I couldn’t find a way, to create a mesh for a point cloud like this, only for point clouds, that can be interpreted as two of its axes define a grid, and the remaining is the data in the intersection of the two. So I stuck with the plotting of points.

## User Interface

After I had all the plots, I created the user interface. I decided to put the profile, edge and 3D wing with edge plots on the gui, and let the user, to change the parameters M, P, XX, the start-end ratio and the number of points used for the resolution.



I created two additional functions for the user interface, a function called **initPlots(app)**, it first gets called when the program starts, and it’s used to set the colours, limits and labels for the plot. The other function is the **calculatewingButtonPushed(app, event)**, which is called when the button is pushed. It reads the values provided by the user, and call the functions I wrote for the calculations and plotting. It also calls initPlots, so at every button push, we start with a clean plot. Below I show some working examples.

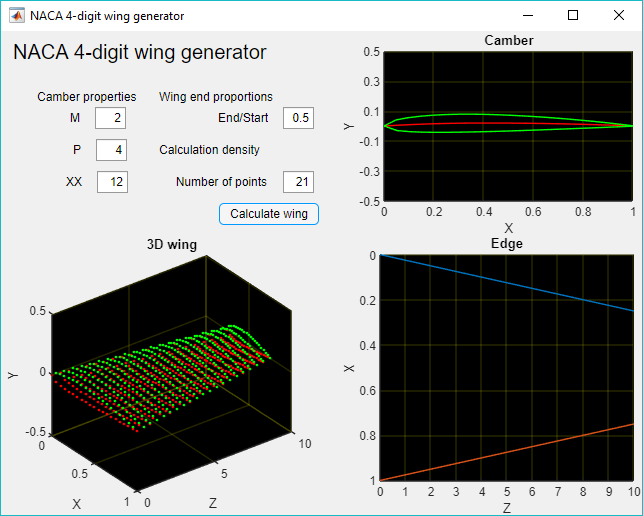


Figure 7 - NACA-2412, s/e:0.5, nop: 21

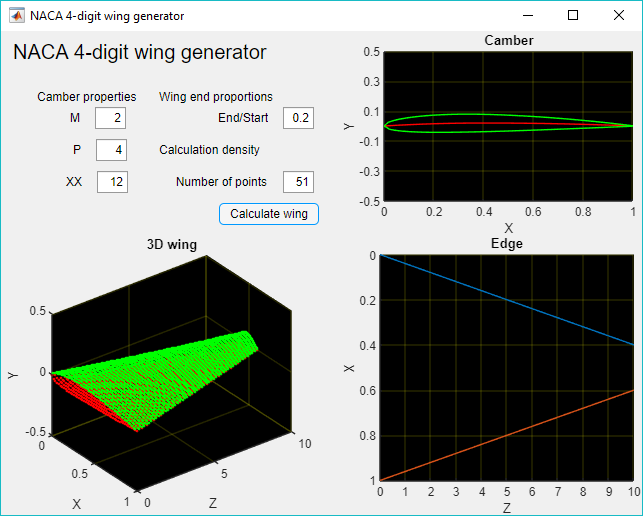


Figure 8 - NACA-2412, s/e:0.2, nop:51

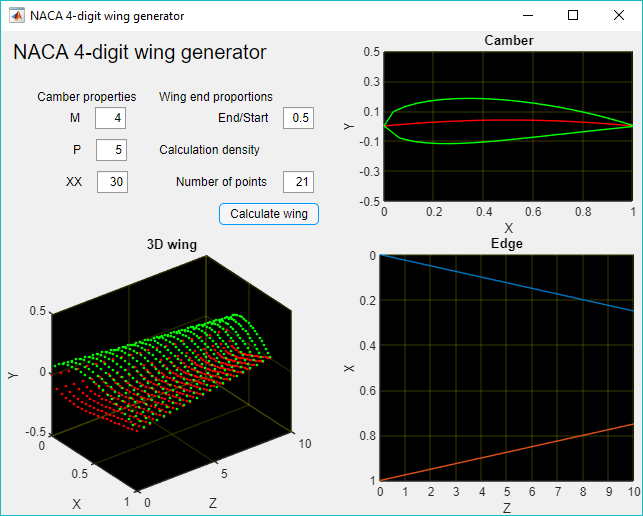


Figure 9 - NACA-4530, s/e:0.5, nop:21

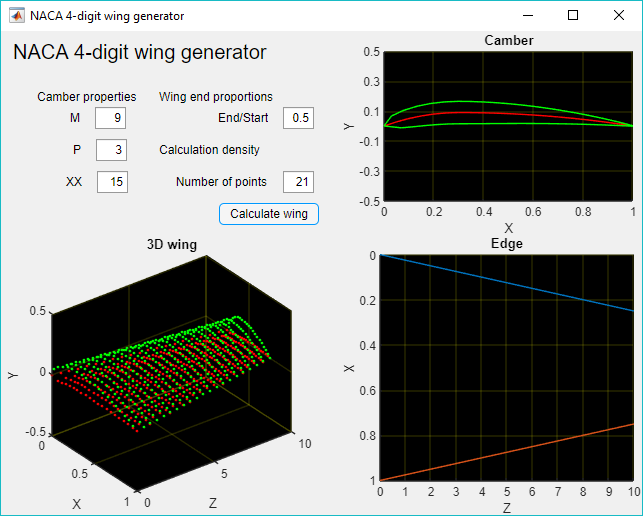


Figure 10 - NACA-9315, s/e:0.5, nop:21

1. In aeronautics, a chord is the imaginary straight line joining the leading and trailing edges of an aerofoil. [↑](#footnote-ref-1)
2. It is a general good practice to break up our script into components and functions in a logical manner, because this way we can better structure it, it’s easier to work with, it’s easier to find faults, it can be easier to explain to someone else, so it just makes our life much simpler. [↑](#footnote-ref-2)