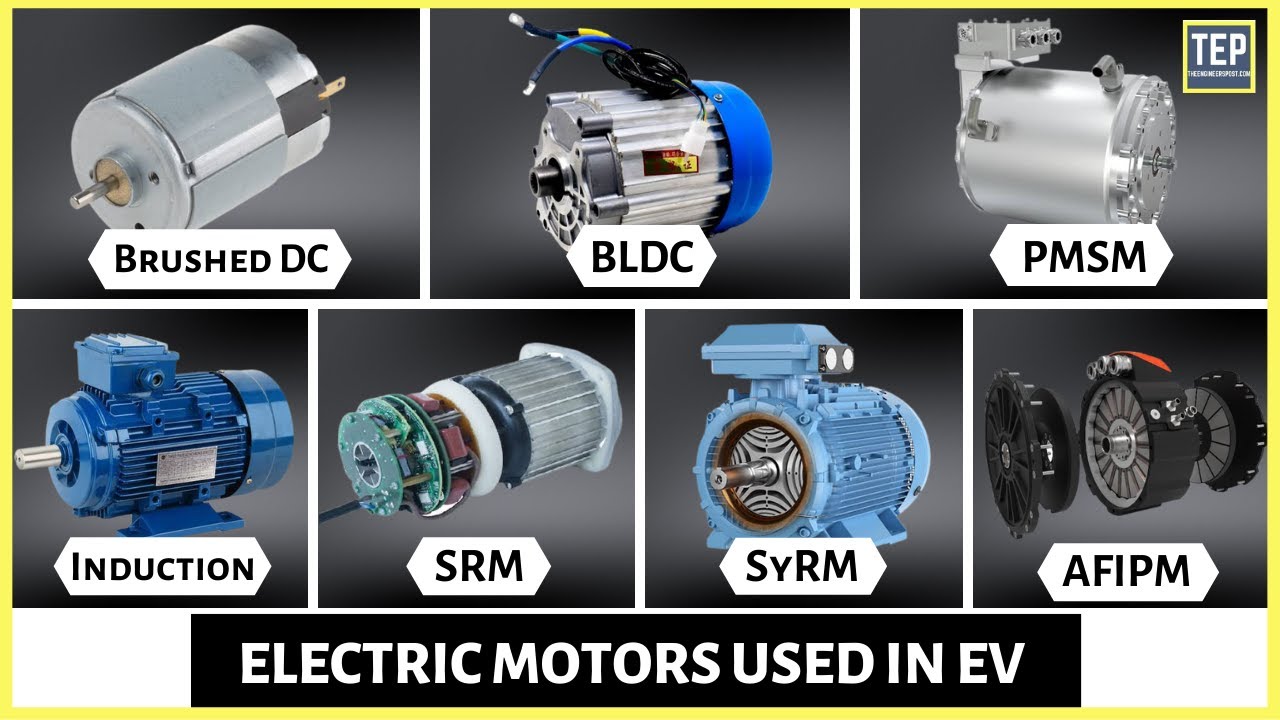
**Name: Kokosalakis Nikolaos Anastasios**

**ID: 60706**

**ELECTRICAL MACHINES DESIGN FOR TRANSPORTATION**



**INTRODUCTION**

We can say that the number of the basic problems that we must solve for an electric machine design are 3:

1) Electromagnetic analysis

* It deals with the calculation and analysis of magnetic fields, magnetic fluxes, currents and electrical losses.
* It ensures that the motor will perform properly in terms of torque, efficiency and energy consumption.

2) Mechanical analysis

* It studies the mechanical strength and deformations of the motor (e.g. stresses, material strength, vibrations, imbalances).
* It ensures that the motor can withstand the loads during its operation without damage or failure.

3) Thermal analysis

* It examines the production and dissipation of heat in the motor as well as its cooling.
* It ensures that the temperature of the components remains at safe levels for long-term operation.

At this experiment with the using of PSO (Particle Swarm Optimization) and MOPSO (Multi-Objective Particle Swarm Optimization) in FEMM falls mainly into the first category – electromagnetic analysis.

**Specifically:**

I used FEMM (Finite Element Method Magnetics) to simulate and analyze the magnetic field, magnetic flux density, iron losses, induced voltage, etc.

With PSO I optimized the geometric characteristics of the machine to improve efficiency or reduce mass, finding the optimal design parameters.

**PSO**

Calculation of Hours:

Hours = = = = 372 hours = 15,5 days

Hours = = = = 38,75 hours = 1,6 days

The names of the archives that I used for every part of the experiment are:

main\_peirama.mat, ObjFunc\_Peirama.mat, PSO\_peirama.mat, MASS\_EFF\_CODE.mat

For the reason why we must run the code for 5 variables

problem.nVar = 5;

beta = X(4);% pole-pitch ratio

betad = X(1); % tooth to slot ratio

ThickyokeR = X(2); % rotor yoke thickness

ThickyokeS = X(3); % stator yoke thickness

Thickmagnet= X(5); % magnet thickness

I put two new limits for the new variables beta = X(4);% pole-pitch ratio and Thickmagnet= X(5); % magnet thickness.

The limits for the variables were:

problem.ub = [X1(0.5),X2(30),X3(30),X4(0.8),X5(4)]; % Upper bounds of search variables

problem.lb = [X1(0.2),X2(10),X3(10),X4(0.4),X5(1)]; % Lower bounds of search variables.

**RUNS**

If we choose to do all the runs with one play, MATLAB will write the Iterations of every new run over the old.

Because I couldn’t find a script(code) which make the program write the results of every run separately at the end of one play I chose to make separately runs(5 different plays).

**RESULTS(EFFICIENCY & MASS)**

The reason why the script did not give us the efficiency (%) and M=Mideal/Mmotor for each iteration is that it was initially designed only to track the best values of the objective function and not to extract the efficiency and mass for all iterations.

**eff\_all = zeros(25,1);**

**mass\_all = zeros(25,1);**

**for i = 1:25**

**filename = sprintf('SO Results after iteration # %d.mat', i);**

**data = load(filename);**

**X = data.Swarm.GBEST.X;**

**try**

**[~, eff, M] = ObjFunc\_peirama(X); % FEMM**

**catch**

**fprintf('FEMM failed on iteration %d\n', i);**

**eff = 0;**

**M = Inf;**

**end**

**eff\_all(i) = eff;**

**mass\_all(i) = M;**

**fprintf('Iter %2d: Eff = %.2f %% | Mass = %.3f kg\n', i, eff\*100, M);**

**end**

1) The instruction **data = load(filename);** reads the files of every Iteration.

2) The instruction **try**

**[~, eff, M] = ObjFunc\_peirama(X); % FEMM**

**catch**

**fprintf('FEMM failed on iteration %d\n', i);**

**eff = 0;**

**M = Inf;**

**end**

gives the three values **[~, eff, M]** of every Iteration.

3) The instruction **eff\_all(i) = eff;** stores the results of every iteration.

**mass\_all(i) = M;**

4) The instruction **fprintf('Iter %2d: Eff = %.2f %% | Mass = %.3f kg\n', i, eff\*100, M);** prints the results of every Iteration.

After this I put the eff and M as extra outputs to the function [opti,eff,M] = ObjFunc(X).

**Data extraction and storage**

I made 5 different runs one of them burned because the code of the efficiency and mass extraction gave fails for every Iteration.

The excel archive has values from 4 different runs and the two of them are similar.

We can see all the convergence curves at the diagram below:

In order to take the results of Mmotor per Iteration I divided the Mideal with the Mass factor :

For example:

Mmotor = Mideal/M = 541/0.49 => Mmotor = 1104.081 kg

The results are included at the relevant excel archive.

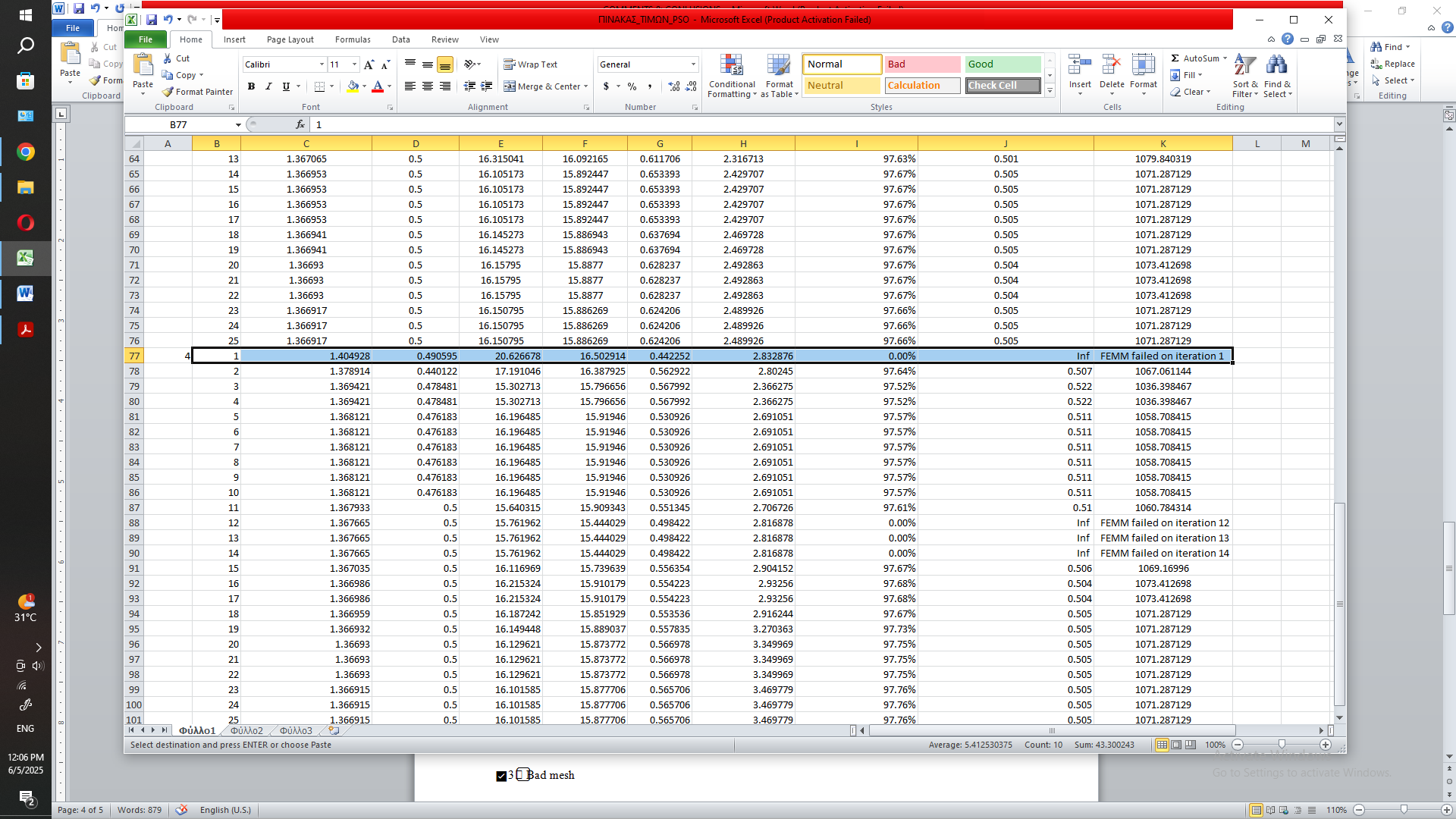
The RUNS with the similar values are the 2nd and the 4th.

The values of every run are into the excel archive with the name ΠΙΝΑΚΑΣ\_ΤΙΜΩΝ\_PSO.

You can find every convergence curve that I got straight form the MATLAB for every run at the PSO SO Iteration files from ….. run files that they are attached at the zip file.

**Comments for some errors that we got from FEMM**

Example of a failed Iteration:



As we can see at the failed iteration example above the FEMM gave us back values but when we tried to calculate the efficiency and the Mass we took back Inf for the efficiency and FEMM failed on iteration 1 for the Mass.

This error usually occurs for one of the following reasons – here is the list of possible causes:

1) Abnormal parameter values ​​(X)

E.g. if the parameters (e.g. betad, ThickyokeR, ThickyokeS, beta, Thickmagnet) are “outside” of the limits that can be drawn in FEMM or “excessive” (e.g. very small dimensions or negative values).

2) Very bad geometry

E.g. overlapping areas (e.g. magnets intersecting with air, lines that do not close properly). FEMM cannot construct the mesh and fails.

3) Bad mesh

When the mesh is not satisfactory, FEMM cannot solve the field equation correctly. This can happen if e.g. you have very “thin” or "strange" regions.

4) Numerical failure in FEMM

In very extreme geometries, FEMM numerical methods fail to converge (e.g. due to singularities or very large gradients in the field).

5) Insufficient boundary conditions

If the boundary conditions are not defined correctly (or are “missing” e.g. some boundary does not have A=0), FEMM “gets lost” and stops

**QUESTION (l)**

I ran the design code again with the best values that I took.

The best values were in the 5th run (4th in excel archive):

Best ObjF value so far Betad(X1) ThickyokeR(X2) ThickyokeS(X3) beta(X4) Thickmagnet(X5)

1.366915 0.5 16.101585 15.877706 0.565706 3.469779

In order to take the drawing from FEMM I used this code:

Best\_Values.m (archive) in PSO File

Code:

X = [0.5, 16.101585, 15.877706, 0.565706, 3.469779]; % best values

[opt, eff] = ObjFunc\_peirama(X); % creates the drawing of the best values



From the piture above we can observe:

Satisfactorily distributed magnetic field throughout the motor topology, with values ​​ranging approximately between 0.4–1.6 Tesla, without supersaturation points (>1.7 T). This indicates good utilization of the magnetic materials and limited saturation losses.

The magnetic flux lines trace closed paths from the permanent magnets to the stator, without discontinuities, confirming effective magnetic coupling and correct geometry.

The magnets, stator and teeth operate within safe flux density levels, contributing to the reliable and efficient operation of the motor.

**MOPSO**

Calculation of hours

Hours = = = = 372 hours = 15,5 days

Hours = = = = 47,91 hours = 1,99 days

The archives that I used for MOPSO results are main\_mopso\_peirama.mat, MOPSO\_peirama.mat, MObjFunc\_peirama.mat, MASS\_EFF\_CODE.mat.

For the reason why we must run the code for 5 variables

problem.nVar = 5;

beta = X(4);% pole-pitch ratio

betad = X(1); % tooth to slot ratio

ThickyokeR = X(2); % rotor yoke thickness

ThickyokeS = X(3); % stator yoke thickness

Thickmagnet= X(5); % magnet thickness

I put two new limits for the new variables beta = X(4);% pole-pitch ratio and Thickmagnet= X(5); % magnet thickness.

The limits for the variables were:

problem.ub = [X1(0.5),X2(30),X3(30),X4(0.8),X5(4)]; % Upper bounds of search variables

problem.lb = [X1(0.2),X2(10),X3(10),X4(0.4),X5(1)]; % Lower bounds of search variables.

I have replaced with the names MOPSO & MObjFunc every name in the code that has names PSO & ObjFunc.

**RUNS**

If we choose to do all the runs with one play, MATLAB will write the Iterations of every new run over the old.

Because I couldn’t find a script(code) which make the program write the results of every run separately at the end of one play I chose to make separately runs(5 different plays).

I followed the same way as I follow to solve the PSO problem.

**RESULTS(EFFICIENCY & MASS)**

The reason why the script did not give us the efficiency (%) and Mmotor (kg) for each iteration is that it was initially designed only to track the best values of the objective function, not to extract the efficiency and mass for all iterations. To address this I used the same script as for the PSO method, but I modified it slightly so that it would be compatible with the MOPSO and would also provide these additional results:

**eff\_all = zeros(25,1);**

**mass\_all = zeros(25,1);**

**for i = 1:25**

**filename = sprintf('MO Results after iteration # %d.mat', i);**

**data = load(filename);**

**X = data.Swarm.GBEST.X;**

**try**

**[~, eff, M] = MObjFunc\_peirama(X); % FEMM**

**catch**

**fprintf('FEMM failed on iteration %d\n', i);**

**eff = 0;**

**M = Inf;**

**end**

**eff\_all(i) = eff;**

**mass\_all(i) = M;**

**fprintf('Iter %2d: Eff = %.2f %% | Mass = %.3f kg\n', i, eff\*100, M);**

**end**

1) The instruction **data = load(filename);** reads the files of every Iteration.

2) The instruction **try**

**[~, eff, M] = MObjFunc\_peirama(X); % FEMM**

**catch**

**fprintf('FEMM failed on iteration %d\n', i);**

**eff = 0;**

**M = Inf;**

**end**

gives the three values **[~, eff, M]** of every Iteration.

3) The instruction **eff\_all(i) = eff;** stores the results of every iteration.

**mass\_all(i) = M;**

4) The instruction **fprintf('Iter %2d: Eff = %.2f %% | Mass = %.3f kg\n', i, eff\*100, M);** prints the results of every Iteration.

After this I put the eff and M as extra outputs to the function [opti,eff,M] = MObjFunc(X).

**Data extraction and storage**

I made 6 different runs one of them burned because the code of the efficiency and mass extraction gave fails for every Iteration.

The excel archive has values from 5 different runs and the three of them are similar.

In order to take the results of Mmotor per Iteration I divided the Mideal with the Mass factor :

For example:

Mmotor = Mideal/M = 541/0.438 => Mmotor = 1235.159 kg

The results are included at the relevant excel archive.

The RUNS with the similar values are the 1st , the 2nd and the 4th and the burned was the 5th.

The values of every run are into the excel archive with the name ΠΙΝΑΚΑΣ\_ΤΙΜΩΝ\_MOPSO.

You can find every pareto front that I got straight form the MATLAB for every run at the

MOPSO Iteration files from ….. run files that they are attached at the zip file.

**The analysis of the code that I found and used to extract the best pareto points per Iteration is below:**

% CALCULATION FOR PARETO

if Vis==1

ideal = [100, 0];

(There the code has an ideal point for efficiency = 100% and mass = 0 kg its as theoretical point for reference.)

all\_best\_points = [];

(It creates an empty table to store the best points of each iteration)

figure;

hold on;

(Creates a new figure (chart window). hold on => will hold multiple charts on the same plot)

for zr = 1:repetitions

(It starts repeating for each run (repetitions times))

f1 = (1 ./ cgCurve(zr,:,1)) \* 100;

f2 = (1 ./ cgCurve(zr,:,2)) \* 1000;

(For each run, it takes:

f1 = 1/efficiency, converted to percentage (%).

f2 = 1/mass, converted to kg.

(It reverses the values ​​because in PSO you have 1/efficiency etc – so it reverses it again to show efficiency/mass as we want))

distances = sqrt((f1 - ideal(1)).^2 + (f2 - ideal(2)).^2);

[minDist, idx] = min(distances);

(Calculates the Euclidean distance of each point from the "ideal".

Finds which point is closest to the ideal (the "best compromise" point)

distances => array of distances

minDist => smallest distance

idx => location of this point)

best\_eff = f1(idx);

best\_mass = f2(idx);

(It gets the actual efficiency & mass values ​​for the best point)

all\_best\_points = [all\_best\_points; zr, best\_eff, best\_mass, minDist];

(Saves the results: run # efficiency mass distance from ideal)

plot(f1, f2, 'b'); hold on;

plot(best\_eff, best\_mass, 'ro', 'MarkerSize', 8, 'LineWidth', 2);

set(gca, 'XDir','reverse')

(Draws the Pareto front (blue line)

Puts a red circle ('ro') at the best point

Flips the X-axis (for better viewing))

end

grid on;

title('Pareto front(s) with best Euclidean points');

xlabel('f1 - efficiency (%)');

ylabel('f2 - mass (kg)');

legend('Pareto fronts','Best point per run')

saveas(gcf, 'Pareto\_with\_best\_points.png');

(Beautifies the chart:

grid, title, axis names, legend

Saves the chart as Pareto\_with\_best\_points.png)

%

T = array2table(all\_best\_points, ...

'VariableNames', {'Run','Efficiency','Mass','EuclideanDist'});

writetable(T, 'Best\_Points\_Euclidean.csv');

(Converts the table to table format (for Excel/csv)

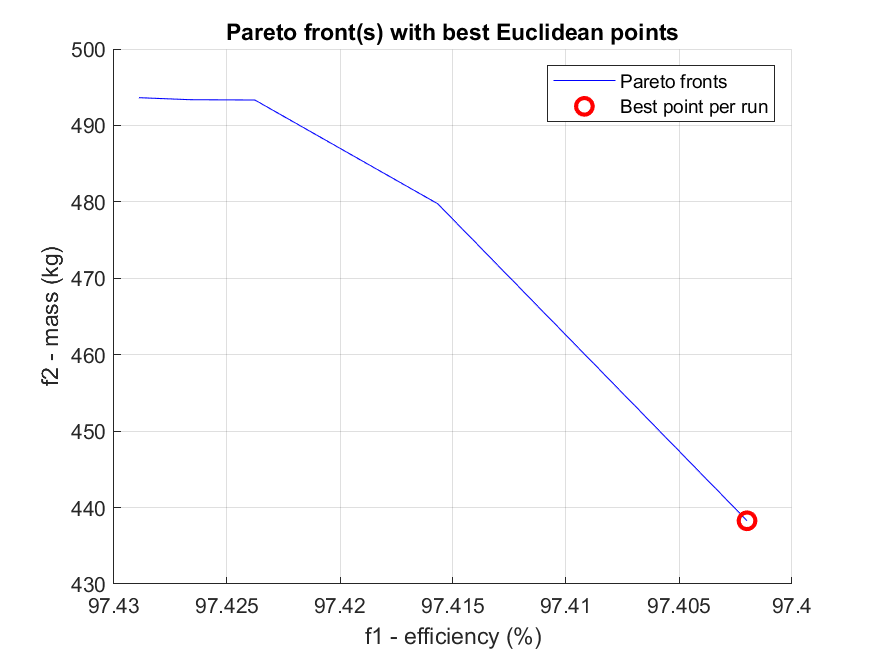
Saves the table to a file Best\_Points\_Euclidean.csv)

end

end

**DIAGRAMS FOR EVERY ITERATION AND BEST POINTS**

ITERATION 1



Comments:

Best Euclidian Point:

Efficiency: ~97.41% Mass: ~440 kg

The x-axis shows the efficiency (%).

The y-axis shows the mass (kg).

The trend is downward: as the mass decreases, the efficiency also decreases (inverse relationship, trade-off).

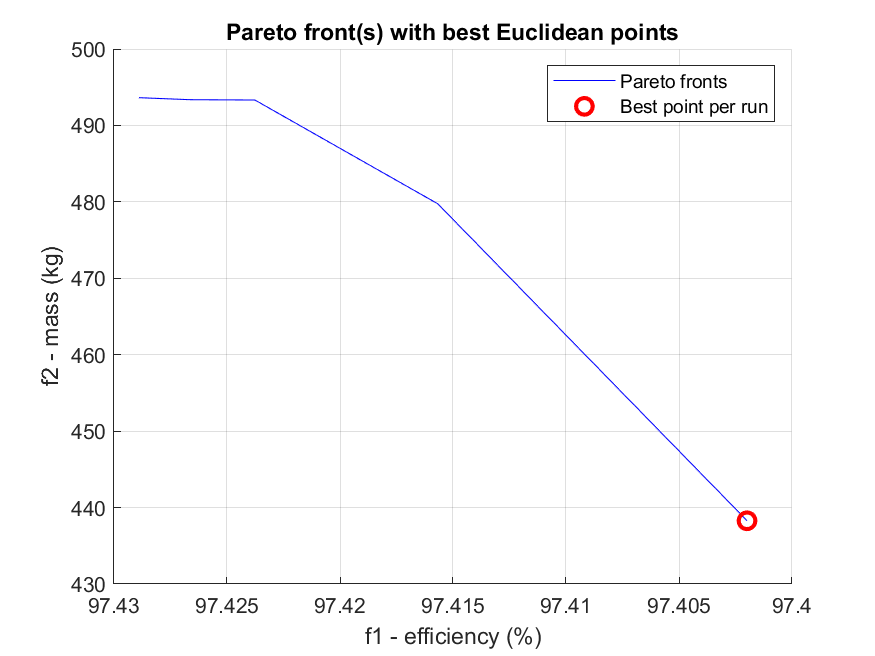
The point at the end of the graph (with the red dot) represents the “most balanced” solution (optimal compromise solution).

It is important that:

The Pareto front confirms that there is a limitation (we cannot have both perfect efficiency and minimum weight at the same time).

The “Best point per run” point (red dot) is very important because it tells us which design is proposed as the most practical choice with the optimal trade-off.

ITERATION 2



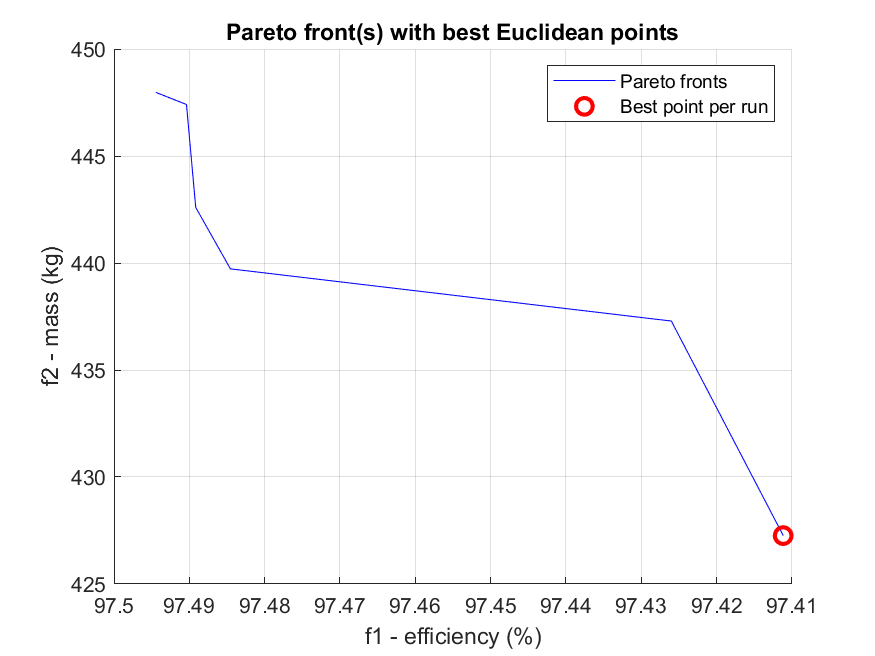
Comments:

Best Euclidian Point:

Efficiency: ~97.41% Mass: ~440 kg

The diagram has the same values with the 1st.

ITERATION 3



Comments:

Best Euclidian Point:

Efficiency: ~97.41% Mass: ~425,5 kg

1)Axes:

The x-axis shows the efficiency percentage (efficiency %).

The y-axis shows the mass (mass kg).

Blue line (Pareto fronts):

Each blue line represents the set of points achieved for different engine configurations.

These points are all on the Pareto front – that is there is no improvement in one parameter (e.g. efficiency) without sacrificing the other (mass).

2)Red dot:

It shows the point with the minimum Euclidean distance from the ideal (100% efficiency and 0 kg mass).

It is the most balanced point between the two goals.

4)Trend:

We see a clear inverse relationship between efficiency and mass:

as the mass decreases, the efficiency also decreases slightly.

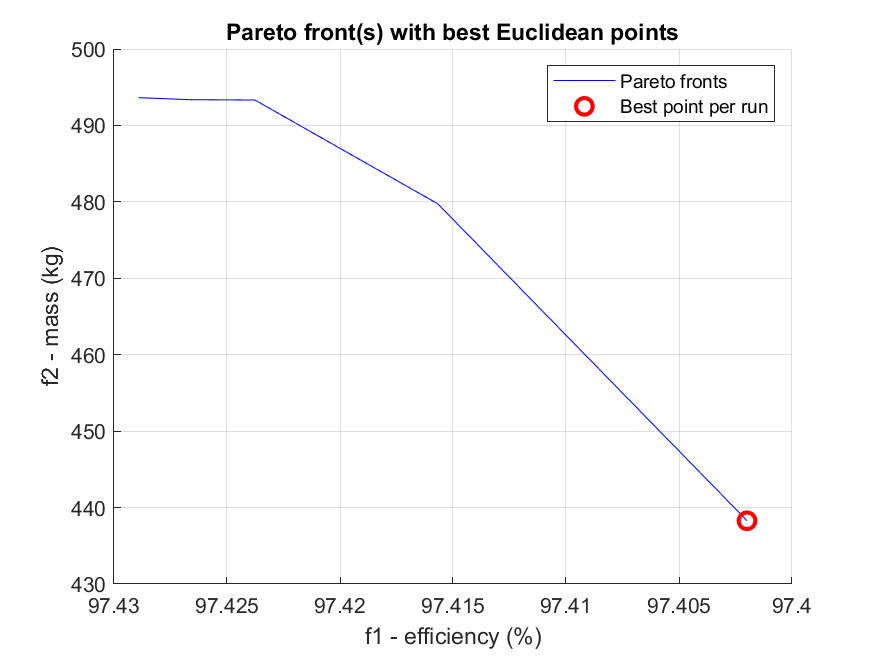
The best point has a slightly lower efficiency but a much better mass.

5)Importance for our project:

This particular red dot can be used as the best “real” design choice (optimized engine with a reasonable compromise between efficiency and mass).

The Pareto front analysis gives us arguments for how you optimized the design and which choices you ultimately propose.

ITERATION 4

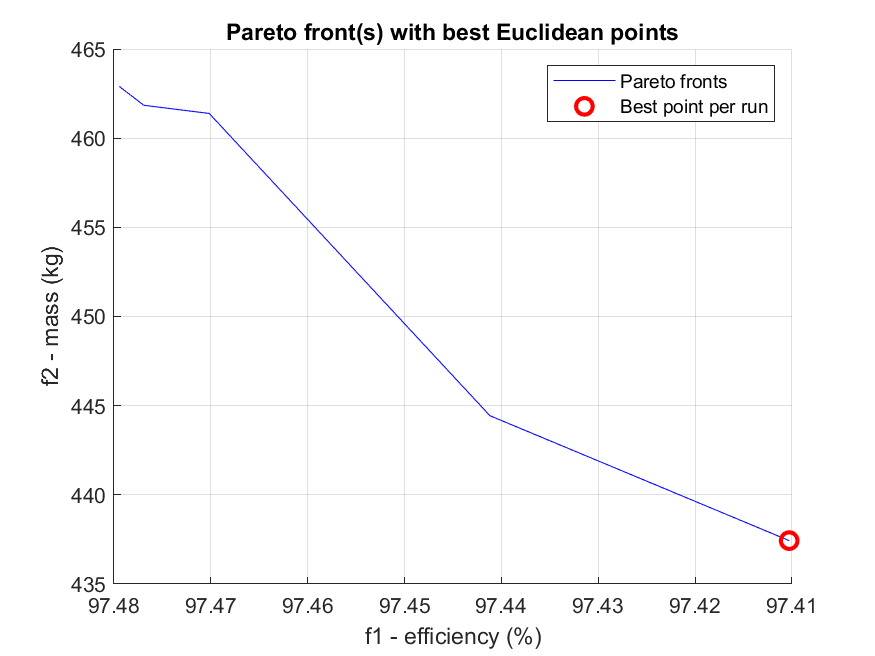


Comments:

The diagram has the same values with the 1st and the 2nd.

ITERATION 5(BURNED)

ITERATION 6



Comments:

Best Euclidian Point:

Efficiency: ~97.41% Mass: ~435,5 kg

Axes

Horizontal (x): Efficiency (efficiency %)

Vertical (y): Mass (mass kg)

Pareto line (blue)

The blue line shows the path of the optimal (Pareto) points.

We observe the reverse trend: as efficiency increases, mass also increases – this is the “tradeoff” of the machine.

Red dot

The red point is the optimal in terms of Euclidean distance from the “ideal” point (100% efficiency, 0 kg mass).

In practice: this point shows us the most balanced result for the given run.

Meaning

The Pareto line shows how difficult it is to achieve high efficiency without significantly increasing mass.

The curvature of the line (non-linearity) shows that there is a “tipping point”: initially we can reduce the mass without a big drop in efficiency but after a certain point (e.g. > 97.45% efficiency) the mass increases sharply for small efficiency gains.

**QUESTION (l)**

I run the design code again with the best values that I took.

The best values were in the 5th run (4th in excel archive):

**Best ΜObjF value so far Best ΜObjF value so far Betad(X1) ThickyokeR(X2) ThickyokeS(X3) beta(X4) Thickmagnet(X5)**

**1.025699 2.232266 0.5 19.406589 18.55989 0.610403 2.811572**

In order to take the drawing from FEMM I used this code:

Best\_Values.m (archive) in MOPSO File

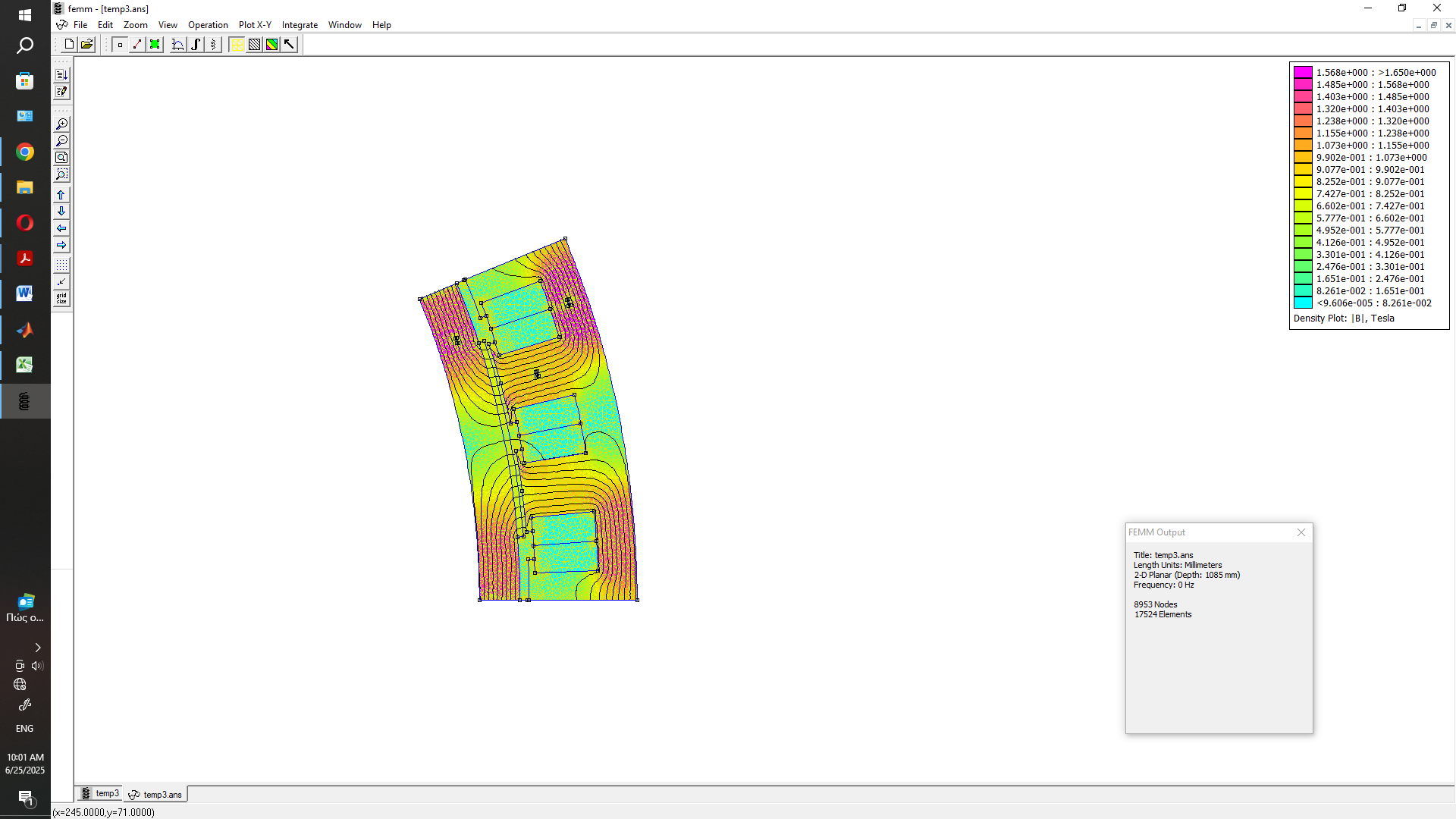
Code:

X = [0.500, 19.4066, 18.5599, 0.6104, 2.8116]; % Best Values

[f1, f2] = MObjFunc\_peirama(X); % Run FEMM design

disp(['Efficiency (reversed): ', num2str(f1)])

disp(['Mass: ', num2str(f2)])



From the picture above we can observe:

The field distribution is smooth and symmetrical, without strong saturation phenomena. The maximum value of the density reaches ~1.56 T, i.e. below the critical saturation limit of the core (~1.7 T).

The magnetic flux lines are well-shaped, without interruptions, which indicates a correct design of the active and magnetic circuit.

The permanent magnets operate in a desired range, without excessive magnetization.

The optimized geometry has achieved satisfactory flux density in the teeth and in the gap, which implies sufficient electromagnetic impulse without magnetic overload or underutilization of materials.

Conclusion:

This solution is a balance between high efficiency and low mass, which is achieved with appropriate geometry and distribution of magnetic materials. Compared to single-criteria optimization, the multi-criteria approach seems to sacrifice a small percentage of performance to achieve a greater mass reduction, but without

**PSO ARCHIVES WITH MATLAB CODES THAT I USED** :

1. main\_peirama
2. ObjFunc\_peirama
3. PSO\_peirama
4. MASS\_EFF\_CODE
5. CHECK\_BAD\_VALUES\_OF\_VARIABLES\_CODE
6. Convergence\_CURVE\_RECOVERY\_CODE
7. anaktisi\_metavlitwn

**MOPSO ARCHIVES WITH MATLAB CODES THAT I USED:**

1. main\_mopso\_peirama
2. MObjFunc\_peirama
3. MOPSO\_peirama
4. MASS\_EFF\_CODE
5. Best\_Values