Magneto-static Analysis of a Brushless DC Motor

Tom Ginsberg, Brendan Posehn

Overview

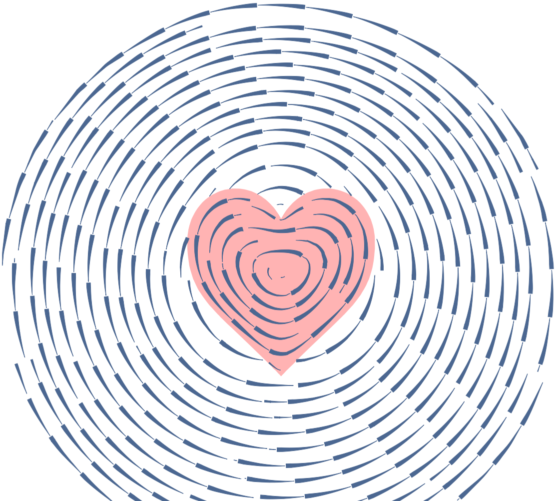
- Background
  Brushless DC Motors
- Theory
  Magneto-statics, Permanent Magnets, Non Linear Materials, Non Linear FEM
- Implementation
  Meshing, Matrix Assembly, Solvers, BLDC Problem
- Results
  Diagrams, Torques

Theory: Magnetostatics in a 2D Cross Section

If current is only perpendicular to the 2D plane, then

$$\nabla \times \nu B = J \text{ and } \nabla \times A = B$$

$$\nabla \times \nu \nabla \times A = J \iff \nabla \cdot \nu \nabla A = J$$



Theory: Permanent Magnets

Maxwell's Equation's for Permanent Magnets

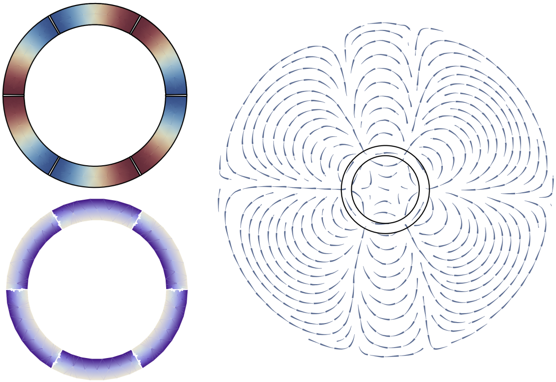
$$B = \mu H + \mu_0 M_r \text{ and } \nabla \times H = J \implies \nabla \times \nu B = J + \nabla \times \nu \mu_0 M_r$$

$$J_m \triangleq \nabla \times \nu \mu_0 M_r \implies \nabla \times \nu B = J + J_m$$

Permanent Magnets

We use a *2D-coil* model for permanent magnets.

- **Assumption**: A bar magnet has a similar field to a current carrying coil
- **Heuristic**: Any magnet geometry can be approximated from gluing together curved bar magnets



Theory: Non Linear FEM

How can we use FEM to solve problems in the form of

$$\nabla \cdot (\alpha(\|\nabla\phi\|) * \nabla\phi) = f(x, y)$$

First thing: Determine $\|\nabla\phi\|$ on a single element in terms of node values of ϕ: (ϕ_i, ϕ_j, ϕ_k)

$$\left|\left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}\right)\right| = \left|\left(\frac{\partial\vec{N}}{\partial x}, \frac{\partial\vec{N}}{\partial y}\right)^{\mathsf{T}} \cdot \begin{pmatrix} \phi_i \\ \phi_j \\ \phi_k \end{pmatrix}\right|$$

Solving the Non Linear System

Now our element matrix equation looks like:

$$F(\vec{\phi}) = \mathbf{K} \cdot \vec{\phi} * \alpha(\vec{\phi}) - \vec{b} = 0$$

No more linear solver :(we need to use Newton's Method

Consider the vector equation $f(\vec{x}) = 0$

- Start with the guess $\vec{x} \approx \vec{x}_0$
- Expand in power series

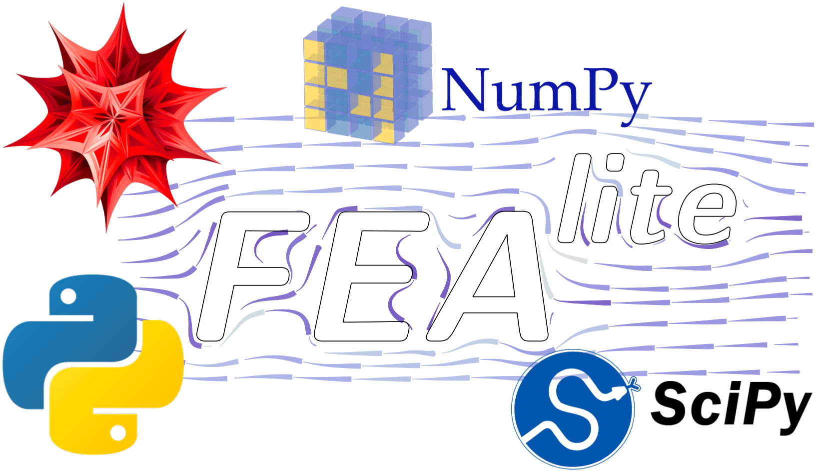
$$f(\vec{x}) \approx f(\vec{x}_0) + \mathbf{J}_{f(\vec{x}_0)}(\vec{x} - \vec{x}_0) \approx 0$$

- Solve a linear system for $\vec{x}_0$ to find a better guess for $\vec{x}$. Iterate.

$$\mathbf{J}_{f(\vec{x}_n)}\left(\vec{x}_n - \vec{x}_{n+1}\right) = f(\vec{x}_n)$$

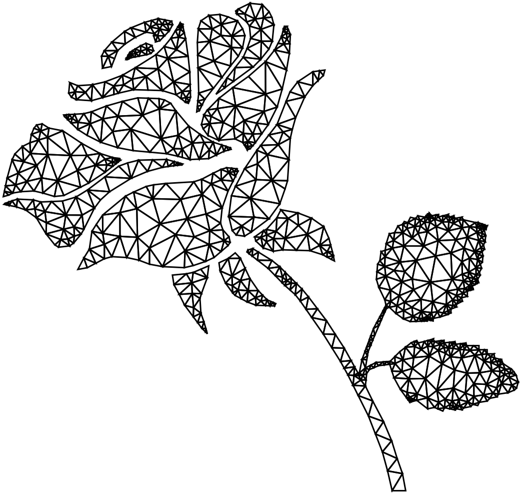
$$\mathbf{J}_{F(\vec{\phi}_n)} = \mathbf{K}\left(\mathbf{I} * \alpha(\vec{\phi}_n) + \left(\vec{\phi}_n\right)^{\mathsf{T}} \cdot \nabla_{\vec{\phi}}\alpha(\vec{\phi}_n)\right)$$

Implementation: FEA*lite*



Mesh

- Generated with Mathematica
- Triangular Elements
- Linear Shape Functions $A = \sum_{i=1}^{3} N_i A_i$

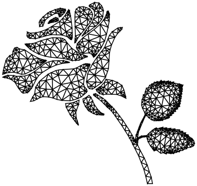


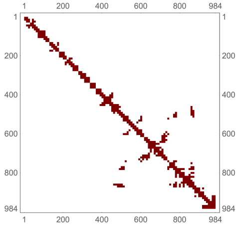
Matrix Assembly

Dense element matrices are assembled into a sparse global matrix using by substituting equality constraints at shared nodes

$$K_{i,j} = \sum_{e \in E(i,j)} \sum_{k=l=0}^{3} K^{e}_{k,l} * \alpha(e)$$

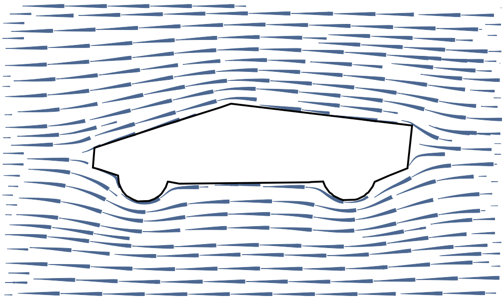
$e = \{v_a, v_b, v_c\}$ is an element, E is the element set, $E(i,j)$ is the set of elements $\{\, e \,|\, e \in E,\ (\{v_i\} \cup \{v_j\}) \subset e\}$, K^e is the local stiffness matrix of e, α is a function of the element properties.





Solvers

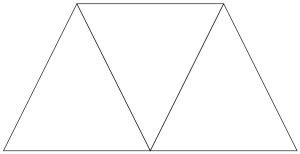
- Linear FEM
  Sparse linear systems solved with
  `scipy.linalg.sparse.spsolve`[1]
- Non Linear FEM Sparse non linear systems solved with `scipy.optimize.fsolve`[2]. Jacobian is analytically computed to speed up the solver.



[1] Wrapper for SuperLU 4.0 (LU decomposition)
[2] Wrapper for MINIPACK hybrj (Newton's Method + Gradient Descent)

Appendix: Mesh Generation and Format

- Meshes are generated using the Mathematica 12 mesh generator
- Exported as:
  list of coordinates $[x,\ y]^v$
  list of mesh elements $[v_1,\ v_2,\ v_3,\ \text{marker}]^e$
  list of boundary elements $[v_1,\ v_2,\ \text{marker}]^{be}$



```
# Coordinates-5
0.0000  0.0000
1.0000  0.0000
0.5000  1.0000
2.0000  0.0000
1.5000  1.0000
# Triangle Elements-5
1       2       3       1
2       3       5       1
2       5       4       1
# Boundary Elements-5
1       2       1
2       4       1
4       5       1
5       3       1
3       1       1
```

Appendix: Matrix Assembly

Global stiffness matrix assembly code for reference

```
[(element[i], element[j], shp_fn.stiffness_matrix[i, j]
    * alpha(marker)) for i in range(3) for j in range(3)
    for element, shp_fn, marker in
    zip(elements, shp_fns, markers)]
```

List of all (row, col, val) tuples, overlapping values are summed.