Ausarbeitung Übung 9

Studienarbeit von Dominik Schiller, Constanze Kramer, Simon Arnold & Tobias Lingenberg
Datum: 3. Februar 2021

Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Ausarbeitung Übung 9

Studienarbeit von Dominik Schiller, Constanze Kramer, Simon Arnold & Tobias Lingenberg

Datum: 3. Februar 2021

Darmstadt

Inhaltsverzeichnis

1 Ausarbeitung der Aufgaben

1.1 Rotationsoperator

Zur numerischen Berechnung der allgemeinen Vektordifferentialgleichung der Magnetostatik

$$\text{rot}\left(\mu^{-}1(\vec{r})\text{rot}\vec{A}(\vec{r})\right) = \vec{J}(\vec{r})$$

wird der Rotationsoperator benötigt. Dieser lässt sich, wie schon der Divergenzoperator, mit dem diskretisierten partiellen Ableitungsoperator

$$(\mathbf{P}_w)_{p,q} := \delta_{p+M_w,q} - \delta p, q = \begin{cases} -1 & \text{für } q = p \\ +1 & \text{für } q = p + M_w \text{ , wobei } w = x, y, z \\ 0 & \text{sonst} \end{cases} \tag{1.1}$$

bestimmen. Es lassen sich nun unterschiedliche Operatoren erzeugen, zunächst der Rotationsoperator

$$\mathbf{C} = \begin{bmatrix} \mathbf{0} & -\mathbf{P}_z & \mathbf{P}_y \\ \mathbf{P}_z & \mathbf{0} & -\mathbf{P}_x \\ -\mathbf{P}_y & \mathbf{P}_x & \mathbf{0} \end{bmatrix},$$

der diskrete div-Operator auf dem primalen Gitter

$$\mathbf{S} = \begin{bmatrix} \mathbf{P}_x & \mathbf{P}_y & \mathbf{P}_z \end{bmatrix},$$

sowie der duale Divergenzoperator

$$\tilde{\mathbf{S}} = \begin{bmatrix} -\mathbf{P}_x^T & -\mathbf{P}_y^T & -\mathbf{P}_z^T \end{bmatrix}.$$

Die angehängte Methode `fit_operator` berechnet diese Operatoren auf einem der Methode übergebenen Gebiet und liefert sie zurück.
Allgemein gilt $\nabla \cdot (\nabla \times \vec{A}) = 0$, diese Identität lässt sich auch mit den primalen Rotations- und Divergenzoperator nachprüfen. Multipliziert man den Divergenzoperator **S** auf den Rotationsoperator ergibt sich

$$\begin{aligned} \mathbf{SC} &= \begin{bmatrix} \mathbf{P}_x & \mathbf{P}_y & \mathbf{P}_z \end{bmatrix} \cdot \begin{bmatrix} \mathbf{0} & -\mathbf{P}_z & \mathbf{P}_y \\ \mathbf{P}_z & \mathbf{0} & -\mathbf{P}_x \\ -\mathbf{P}_y & \mathbf{P}_x & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{P}_y\mathbf{P}_z - \mathbf{P}_z\mathbf{P}_y & -\mathbf{P}_x\mathbf{P}_z + \mathbf{P}_z\mathbf{P}_x & \mathbf{P}_x\mathbf{P}_y - \mathbf{P}_y\mathbf{P}_x \end{bmatrix}. \end{aligned} \tag{1.2}$$

Nutzt man nun die Eigenschaften des Satz von Schwarz und der Kommutativität zweier Matrizen aus, so lässt sich die (1.2) als

$$\mathbf{SC} = \begin{bmatrix} \mathbf{P}_y\mathbf{P}_z - \mathbf{P}_y\mathbf{P}_z & -\mathbf{P}_x\mathbf{P}_z + \mathbf{P}_x\mathbf{P}_z & \mathbf{P}_x\mathbf{P}_y - \mathbf{P}_x\mathbf{P}_y \end{bmatrix} \tag{1.3}$$

schreiben, wodurch sich

$$\mathbf{SC} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

ergibt.

Interpretiert man das Ergebnis geometrisch, wird über jede Fläche eines Volumens integriert. Hierzu wird jede Fläche über Kantenintegrale ausgedrückt. Unter Beachtung der Flächennormalen und der damit verbundenen Integrationsrichtung wird über jede Kante zweimal in unterschiedlicher Richtung integriert. Somit heben sich diese auf, wodurch die ganze Rechnung einen Nullvektor erzeugt.

1.2 Materialmatrix

Um ein allgemeines magnetostatisches Problem zu lösen benötigt man die Materialmatrix der Reluktivitäten $\mathbf{M}_\nu$. Im folgenden werden in jeder Elementarzelle homogene Materialeigenschaften angenommen. Daher lässt sich mit der gemittelten Reluktivität $\bar{\nu}_n$ zwischen zwei benachbarten Zellen, jedes Element $\mathbf{M}_\nu(n,n)$ näherungsweise mit

$$\frac{\int_{\tilde{L}_n} \vec{H}\mathrm{d}\vec{s}}{\int_{A_n} \vec{B}\mathrm{d}\vec{A}} \approx \frac{\bar{\nu}_n|\tilde{L}_n|}{|A_n|} =: \mathbf{M}_\nu(n,n)$$

bestimmen. Die dabei entstehende Matrix $\mathbf{M}_\nu$ hat Diagonalform.

Die MATLAB Routine `fit_calc_elements` (siehe Code(1.3)) berechnet die primalen und dualen Flächeninhalte `dA` und `ddA` sowie die Kantenlängen `ds` und `dds` in Abhängigkeit von den vorgegebenen Gittergrößen `xmesh, ymesh` und `zmesh`. Die primalen Kanten entlang einer Achsenrichtung haben alle die selbe Länge `stepw`, die dualen Kanten berechnen sich je zur Hälfte aus den daneben liegenden primalen Kanten. Daraus folgt das duale Kanten am Rand des Rechengebiets genau halb so lang sind wie die Kanten im Inneren. Die jeweiligen Flächeninhalte ergeben sich aus der Multiplikation der Kantenlänge der angrenzenden Kanten.

Die Funktion `fit_calc_avgny` (siehe Code (1.2)) ermittelt den Mittelwert der Reluktiviät zwischen zwei beliebigen, benachbarten Zellen.

Mit Hilfe der zwei schon genannten Funktionen gibt die Funktion `createMny` (siehe Code(1.1)) die Materialmatrix $\mathbf{M}_\nu$ zurück. Die Einträge sind dabei in $x-$, $y-$ und $z-$Richtung sortiert.

Da am Rand des Rechengebiets zur Mittelung der Reluktivität, zur Hälfte eine nicht vorhandene Zelle mit Reluktivität $\nu = 0$, in die Rechnung einfließt gibt es hier eine kleine Ungenauigkeit und $\bar{\nu}$ ist hier kleiner als es eigentlich sein sollte.

1.3 Definition eines Stromflusses, Octave

Wir betrachten nun das Problem zwei parallel verlaufender Kabel (siehe Abb. 1.1). Kabel 1 soll den Strom $1\mathrm{A}$ und Kabel 2 entgegengesetzt $-1\mathrm{A}$ führen. Angenommen wird, dass die magnetische Flussdichte an den Randflächen Γ des Rechengebiets abgeklungen ist und

$$\vec{A}_t(\vec{r}) = 0 \qquad \text{für alle } \vec{r} \in \Gamma \tag{1.4}$$

gilt.

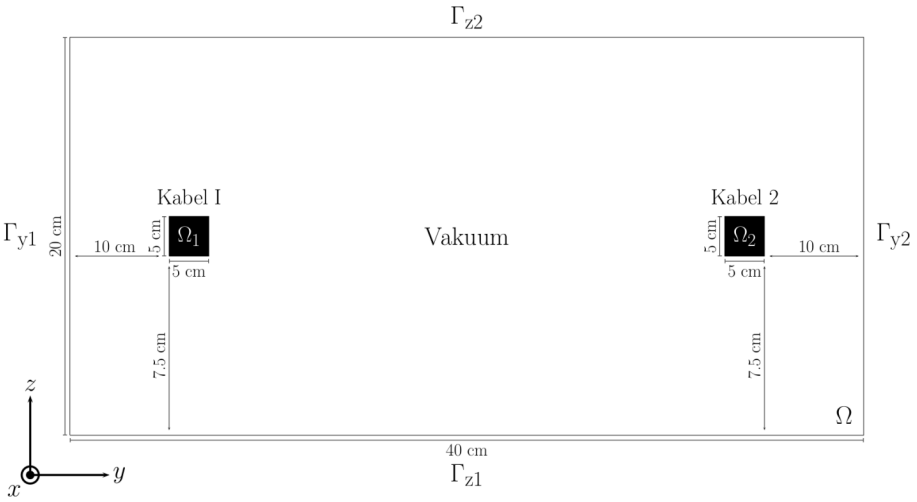

Abbildung 1.1: Zwei Kabel in der yz-Ebene

Für die Simulation wird ein äquidistantes Rechengitter definiert. Wir haben uns dazu entschieden zu Testzwecken zunächst einmal ein sehr grobes Gitter zu wählen, mit Anzahl Knoten $N_x = 2, N_y = 9, N_z = 9$ und einer Schrittweite $h_x = 20\mathrm{cm}, h_y = 5\mathrm{cm}, h_z = 2,5\mathrm{cm}$ (siehe Code (1.4)). Mithilfe der zuvor geschriebenen Methoden `fit_operator` wird der benötigte FIT-Operator C erzeugt. Es folgt die Bestimmung der Indexmenge `indxT` aller Randkanten, die wir als Vektor definiert haben. Hierbei werden erst alle Kanten in x-Richtung durch gegangen, daran angehängt die in y- und z-Richtung. Im Vektor steht nun entweder eine 1, falls die Kante eine Randkante ist, ansonsten eine 0.

Ähnlich wird bei Bestimmung des Anregungsvektors $\vec{j}$ vorgegangen. Der Vektor wird am Index einer Kante, die zum einen in Ω_1 oder Ω_2 liegt und zusätzlich in Stromrichtung (also x-Richtung) zeigt, auf ± 1 gesetzt. Anschließend wird der Vektor noch mit dem Faktor $\left(\frac{h_x}{5\mathrm{cm}}\right)^2$ multipliziert.

Nun soll die Gleichung

$$K\vec{a} = \vec{j} \qquad \text{mit } K = C^T M_v C \tag{1.5}$$

gelöst werden. Hierzu wird zunächst noch mithilfe der schon geschriebenen Routine `createMny` die Matrix M_v erzeugt. Das magnetische Vektorpotential $\vec{a}$ enthält momentan schon Werte die uns bekannt sind, jene, die durch die Randbedingung (1.4) festgelegt sind. Es wird eine Zerlegung des Gleichungssystems in Bekannte und Unbekannte vorgenommen. Hierzu werden aus dem Gleichungssystem (1.5) alle Gleichungen entfernt die Randkanten beschreiben. Mithilfe des Vektors `indxT` werden in K und $\vec{j}$ alle Zeilen (bzw. bei K auch die

Spalten) gelöscht, die dem Index einer Randkante entsprechen. Das verkleinerte Gleichungssystem lässt sich nun eindeutig lösen. Die Randbedingung fungiert als eine Form von Eichung.

Zuletzt können nun noch die beiden Gleichungen

$$\vec{b} = C\vec{a},$$
$$L = (\vec{j})^T \frac{\vec{a}}{I^2}$$

berechnet werden (siehe Code (1.4))

Eine Simulation in FEMM ergab das resultierende magnetische Feld (Abb. 1.2) und den Stromfluss (Abb. 1.3). Der Stromfluss verhält sich nach dem Skineffekt so wie erwartet (Am Rand des Kabels ist ein erhöhter Stromfluss erkennbar). Wieder den Erwartungen verhält sich jedoch das Magnetfeld (Abb. 1.2) durch Bedingung (1.4) sollte es am Rand des Rechengebiets 0 sein.

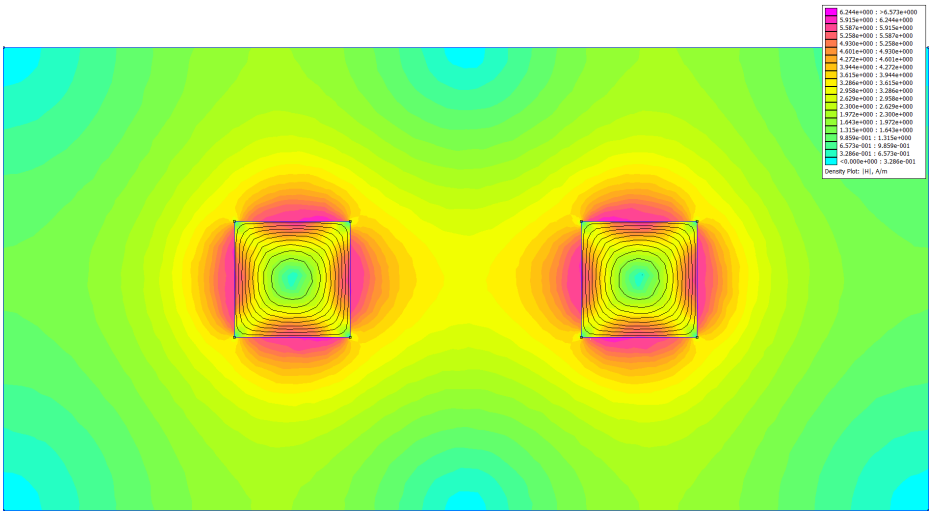

Abbildung 1.2: Simulation des Magnetischen Feldes in FEMM

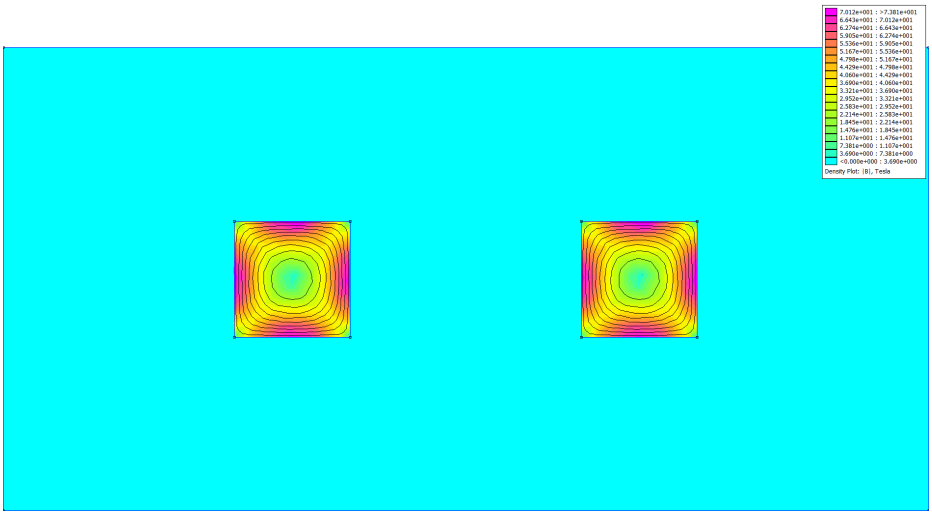

Abbildung 1.3: Simulation des Stromflusses in FEMM

1.4 Anhang

```
function [C,S,Ss] = fit_operator(Nx,Ny,Nz)
Mx = 1;
My = Nx;
Mz = Ny * Nx;
Np = Nx * Ny * Nz;

Px = zeros(Np,Np);
Py = zeros(Np,Np);
Pz = zeros(Np,Np);

for p = 1 : Np
    for q = 1 : Np
        if (q==p)
            Px(p,q) = -1;
            Py(p,q) = -1;
            Pz(p,q) = -1;
        end
         if (q==p+Mx)
             Px(p,q) = 1;
         end
         if (q==p+My)
             Py(p,q) = 1;
         end
         if (q==p+Mz)
             Pz(p,q) = 1;
         end
    end
end

C = [zeros(Np,Np) -Pz Py; Pz zeros(Np,Np) -Px; -Py Px zeros(Np,Np)];
%C = sparse(C);
S = [Px Py Pz];
%S = sparse(S);
Ss = [-Px' -Py' -Pz'];
%Ss = sparse(Ss);
```

data/fit_operator.m

1.4.1 Skript Aufgabe 9.2

```
  function Mny = createMny(xmesh,ymesh,zmesh,ny)
  Nx = size(xmesh);
  Nx = Nx(1,2);
  Ny = size(ymesh);
  Ny = Ny(1,2);
  Nz = size(zmesh);
  Nz = Nz(1,2);
  Np = Nx*Ny*Nz;
  Mx = 1;
  My = Nx;
  Mz = Nx*Ny;

  Mny = sparse(3*Np,3*Np);
```

```
[ddS, dS, ddA, dA] = fit_calc_elements(xmesh,ymesh,zmesh);

for n = 1 : Np
if n-Mx>=1
avgny = fit_calc_avgny(dS(n,1),dS(n-Mx,1),ddS(n,1),ny(n,1),ny(n-Mx,1));
else
avgny = fit_calc_avgny(dS(n,1),0,ddS(n,1),ny(n,1),0);
end

if dA(n,1) == 0
Mny(n,n) = 0;
else
Mny(n,n) = (avgny*ddS(n,1))/dA(n,1);
end
end

for n = 1 : Np
if n-My>=1
avgny = fit_calc_avgny(dS(n,2),dS(n-My,1),ddS(n,2),ny(n,1),ny(n-My,1));
else
avgny = fit_calc_avgny(dS(n,2),0,ddS(n,2),ny(n,1),0);
end

if dA(n,2) == 0
Mny(n,n) = 0;
else
Mny(Np+n,Np+n) = (avgny*ddS(n,2))/dA(n,2);
end
end

for n = 1 : Np
if n-Mz  >= 1
avgny = fit_calc_avgny(dS(n,3),dS(n-Mz,3),ddS(n,3),ny(n,1),ny(n-Mz,1));
else
avgny = fit_calc_avgny(dS(n,3),0,ddS(n,3),ny(n,1),0);
end

if dA(n,3) == 0
Mny(n,n) = 0;
else
Mny(2*Np+n,2*Np+n) = (avgny*ddS(n,3))/dA(n,3);
end
end
Mny = sparse(Mny);
end
```

Listing 1.1: Aufgabe 9.2 Berechnung der Materialmatrix

```
function nymid = fit_calc_avgny(L1,L2,dL1,ny1,ny2)
nymid = (L1*ny1+L2*ny2)/(2*dL1);
end
```

Listing 1.2: Aufgabe 9.2 Berechnung gemittelte Reluktivität

```
function [ddS,dS,ddA,dA] = fit_calc_elements(xmesh,ymesh,zmesh)
Nx = size(xmesh);
Nx = Nx(1,2);
Ny = size(ymesh);
```

```
Ny = Ny(1,2);
Nz = size(zmesh);
Nz = Nz(1,2);
Np = Nx*Ny*Nz;
Mx = 1;
My = Nx;
Mz = Nx*Ny;


stepx = calc_steps(Nx,xmesh(1,Nx)-xmesh(1,1));
stepy = calc_steps(Ny,ymesh(1,Ny)-ymesh(1,1));
stepz = calc_steps(Nz,zmesh(1,Nz)-zmesh(1,1));

dA = sparse(Np,3);
ddA = sparse(Np,3);
dS = sparse(Np,3);
ddS = sparse(Np,3);

dx = sparse(Np,1);
dy = sparse(Np,1);
dz = sparse(Np,1);

%%%%% Berechnung dS und ddS %%%%%
n = 1;

for nz = 1 : Nz
for ny = 1 : Ny
for nx = 1 : Nx
if nx < Nx
dx(n) = stepx;
end
if ny < Ny
dy(n) = stepy;
end
if nz < Nz
dz(n) = stepz;
end
n = n+1;
end
end
end

dS = [dx dy dz];

ddx = sparse(Np,1);
ddy = sparse(Np,1);
ddz = sparse(Np,1);

z = 1;
for i = 1 : Np

if (i <= Nx*Ny*z-Nx) && (z < Nz)
if mod(i,Nx) == 0
ddx(i) = dx(i-1)/2;
elseif mod(i,Nx) == 1
ddx(i) = dx(i)/2;
else
ddx(i) = dx(i)/2+dx(i-Mx)/2;
```

```
63  end
64  end
65
66  if (z < Nz)
67  if i < Nx*z
68  ddy(i) = dy(i)/2;
69  elseif (i < Nx*Ny*z) && (i > (Nx*Ny*z-Nx))
70  ddy(i) = dy(i-My)/2;
71  elseif mod(i,Nx) == 0
72  ddy(i) = 0;
73  else
74  ddy(i) = dy(i)/2+dy(i-My)/2;
75  end
76  end
77
78  if (z <= Nz)
79  if (mod(i,Nx) ~= 0) && (i < Nx*(Ny-1))
80  ddz(i) = dz(i)/2;
81  elseif (mod(i,Nx) ~= 0) && (i < Nx*Ny*z-Nx) && (z < Nz)
82  ddz(i) = dz(i)/2+dz(i-Mz)/2;
83  elseif (mod(i,Nx) ~= 0) && (i < Nx*Ny*z-Nx)
84  ddz(i) = dz(i-Mz)/2;
85  else
86  ddz(i) = 0;
87  end
88  end
89
90  if mod(i,Nx*Ny) == 0
91  z = z+1;
92  end
93
94  end
95
96  ddS = [ddx ddy ddz];
97
98
99  %%%%% Berechnung dA und ddA %%%%%
100 for i = 1 :  Np
101 dA(i,1) = dS(i,2)*dS(i,3);
102 dA(i,2) = dS(i,1)*dS(i,3);
103 dA(i,3) = dS(i,2)*dS(i,1);
104
105 ddA(Np+1-i,1) = ddS(Np+1-i,2)*ddS(Np+1-i,3);
106 ddA(Np+1-i,2) = ddS(Np+1-i,1)*ddS(Np+1-i,3);
107 ddA(Np+1-i,3) = ddS(Np+1-i,2)*ddS(Np+1-i,1);
108 end
109
110 end
```

Listing 1.3: Aufgabe 9.2 Berechnung Kantenlänge und Flächeninhalte

Skript Aufgabe 9.3

```
1  clear;
2  Nx = 2;
3  Ny = 9;
```

```
Nz = 9;
Np = Nx*Ny*Nz;

xmesh = [0:20:20];
ymesh = [0:5:40];
zmesh = [0:2.5:20];

nx = sparse(Np,1);
ny = sparse(Np,1);
nz = sparse(Np,1);

indxOmega1 = sparse(3*Np,1);
indxOmega2 = sparse(3*Np,1);
stepY = calc_steps(Ny,40);
stepZ = calc_steps(Nz,20);
h = calc_steps(Nx,20);


n = 1;
for z = 1 : Nz
for y = 1 : Ny
for x = 1 : Nx
if (z*stepZ >= 7.5 && z*stepZ <= 12.5)
if (y*stepY >= 10 && y*stepY <= 15)
indxOmega1(n) = 1;
end
if (y*stepY >= 25 && y*stepY <= 30)
indxOmega2(n) = -1;
end

end
if (x < Nx) && ((y == 1) || (y == Ny) || (z == 1) || (z == Nz))
nx(n) = 1;
end

if (y < Ny) && ((x == 1) || (x == Nx) || (z == 1) || (z == Nz))
ny(n) = 1;
end

if (z < Nz) && ((x == 1) || (x == Nx) || (y == 1) || (y == Ny))
nz(n) = 1;
end

n = n+1;
end
end
end

indxT = [nx;ny;nz];

j = (h/5)^2*(indxOmega1+indxOmega2);
j = sparse(j);
jr = j;

Mny = createMny(xmesh,ymesh,zmesh,ones(Np,1));
[C,S,Ss] = fit_operator(Nx,Ny,Nz);
K = C'*Mny*C;

```

```
%Loesche Rand raus
for i=length(indxT):-1:1
if indxT(i) == 1
K(i,:)=[];
K(:,i)=[];
j(i,:)=[];
end
end

a = K\j;

%Fuege Rand wieder hinzu
ar = sparse(length(indxT),1);
counter = 1;
for i=1:length(indxT)
if indxT(i) == 0
ar(i) = a(counter);
counter = counter +1;
end
end

b = C*ar;

%fit_write_vtk(xmesh, ymesh, zmesh, 'b-Fluss.vtr', {'j',j;'b',b})

L = jr' * ar;
```

Listing 1.4: Berechnung der Aufgaben 9.3 a) bis g)

Abbildungsverzeichnis