Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

De manifold-mapping optimalisatietechniek (Engelse titel: The manifold-mapping optimization technique)

Verslag ten behoeve van het
Delft Institute for Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

Lisa Wobbes

Delft, Nederland
Juli 2011

BSc verslag TECHNISCHE WISKUNDE

"De manifold-mapping optimalisatietechniek"
(Engelse titel: "The manifold-mapping optimization technique)"

Lisa Wobbes

Technische Universiteit Delft

Begeleider

Dr. D.J.P. Lahaye

Overige commissieleden

Prof.dr.ir. C. Vuik

Dr. J.G. Spandaw

Dr. J.A.M. de Groot

Juli, 2011

Delft

Voorwoord

Dit verslag is gemaakt in het kader van het bachelorproject van de opleiding Technische Wiskunde.
Het onderwerp *Manifold Mapping* sprak me gelijk aan, omdat het toepassingen had in elektromagnetisme. Tijdens mijn minor had ik bij Technische Natuurkunde het vak *Elektromagnetisme I* gevolgd. Het leek me vooral zeer interessant om de inhoud van dat vak te combineren met numerieke wiskunde.
Daarnaast wil ik Dr. D.J.P. Lahaye bedanken voor zijn begeleiding tijdens het project.

Inhoudsopgave

1 Inleiding

Bij sommige ontwerpproblemen uit de praktijk wordt ernaar gestreefd een wiskundig model aan een aantal voorwaarden te laten voldoen. Dikwijls is het niet mogelijk om een model te construeren dat exact aan deze voorwaarden voldoet en dan bestaat het ontwerpproces uit de minimalisering van het verschil tussen verkregen en gewenste resultaten. Er wordt dus een optimaliseringprobleem opgelost. Optimalisatie-algoritmen zijn typisch iteratieve processen waarbij de wiskundige modellen een aanzienlijk aantal keren doorgerekend worden.

De meeste opimaliseringsproblemen vereisen een nauwkeurig resultaat. Dit leidt er soms toe dat een voldoende goede oplossing na een zeer lange tijd gevonden wordt. De modellen die een nauwkeurig resultaat opleveren, maar veel rekentijd vereisen, worden fijne modellen genoemd. Vaak kunnen fijne modellen vereenvoudigd worden, bijvoorbeeld door een rekenrooster grover te kiezen of het onderliggende proces eenvoudiger te beschrijven. Het optimaliseren van de vereenvoudigde modellen kost minder rekenwerk en wordt de oplossing aanmerkelijk sneller gevonden. Helaas is de gevonden oplossing in het algemeen minder nauwkeurig dan de oplossing van het fijne model. De modellen die lage nauwkeurigheid hebben, maar binnen een korte tijd een oplossing vinden, worden grove modellen genoemd. Grove modellen zijn goedkoop, omdat hun evaluatie weinig tijd vereist. Daartegenover zijn fijne modellen duur.

De kwaliteit van de grove oplossing kan worden verbeterd door ook gebruik te maken van het fijne model. Om de rekentijd minimaal te houden, wordt het fijne model slechts een paar keer benut. Manifold Mapping is één van de technieken die door een slimme combinatie van een fijn en grof model snel een nauwkeurige oplossing oplevert. Het belangrijkste voordeel van de Manifold Mapping ten opzichte van andere soortgelijke methodes is dat Manifold Mapping altijd convergeert naar de oplossing van het fijne model.

In deze scriptie wordt eerst de theorie beschreven waarop het algoritme van Manifold Mapping gebaseerd is. Vervolgens wordt de methode toegepast op een ontwerpprobleem van een voice-coil actuator. Een voice-coil actuator is een toestel die elektrische signalen in trillingen omzet. Er zijn drie belangrijke onderdelen van een voice-coil actuator: een magneet, een spoel en een ferromagnetische kern waarin de magneet ingebouwd is. We proberen de optimale constructie van een voice-coil actuator te vinden zodat voldaan wordt aan de bepaalde eisen. Dit wordt gedaan voor een 1D en een 2D probleem. Bij het 1D probleem zoeken we de optimale straal van de magneet van de cilindrische voice-coil actuator. Bij het 2D probleem wordt niet alleen naar de straal gekeken maar ook naar de hoogte van de magneet. Voor beide problemen stellen we eerst een fijn en een grof model op. Deze modellen worden vervolgens gebruikt voor de Manifold Mapping.

Daarnaast kijken we naar de stabiliteit van de convergentie van de Manifold Mapping. We bespreken mogelijke oorzaken van het instabiele gedrag en proberen het verloop van het algoritme te verbeteren door het aanpassen van het grove model. Dit doen we niet alleen voor het tweedimensionale probleem, maar ook voor het probleem in 4D en 7D.

Ten slotte bespreken we de resultaten die wij krijgen en suggereren een paar richting voor het vervolgonderzoek.

2 Fijne en grove modellen in optimalisering

2.1 Optimaliseringsprobleem

We voeren eerst de belangrijke notatie in.
De data van het optimaliseringprobleem wordt genoteerd als $(\mathbf{D}, \mathbf{y}) = (\{D_i\}, \{y_i\})$ met $i = 1, ..., m$.
$\mathbf{D} \in \mathbb{R}^m$ is de vector van onafhankelijke variabelen. Dit kan bijvoorbeeld een vector zijn met verschillende posities, tijdstippen of frequenties.
Elke element van $\mathbf{y} \in Y \subset \mathbb{R}^m$ is zowel afhankelijk van de onafhankelijke variabele als van de vector met ontwerpvariabelen $\mathbf{x}$. Dus $y_i = y(D_i, \mathbf{x})$. De vector $\mathbf{x}$ wordt als input gebruikt waarmee men probeert het verschil tussen de verkregen en gewenste oplossingen te minimaliseren.

2.2 Fijn model

Het fijne model heeft een hoge nauwkeurigheid, maar is tijdrovend en dus ook duur.

De input van het fijne model wordt genoteerd als $\mathbf{x}_f$ met $\mathbf{x}_f \in X_f \subset \mathbb{R}^n$. De verzameling X_f van alle mogelijke ontwerpvariabelen is meestal een gesloten en begrensd deelverzameling van $\mathbb{R}^n$.
De output van dit model is $\mathbf{f}(\mathbf{x}_f) \in \mathbb{R}^m$. We nemen aan dat $\mathbf{f}(\mathbf{x}_f)$ continu en differentieerbaar is.
Met $\mathbf{f}(X_f)$ wordt de verzameling van alle mogelijke uitkomsten van het fijne model genoteerd.
Om het verschil tussen de oplossing van het fijne model en de gewenste oplossing te beschrijven, wordt het verschil $||\mathbf{f}(\mathbf{x}_f) - \mathbf{y}||$ gedefinieerd. Dit verschil moet zo klein mogelijk zijn. Dus we zoeken

$$\mathbf{x}_f^* = \text{argmin}_{\mathbf{x}_f \in X_f} ||\mathbf{f}(\mathbf{x}_f) - \mathbf{y}||.$$

Deze input wordt het optimum van het fijne model genoemd.

2.3 Grof model

Het grove model heeft een lagere nauwkeurigheid in vergelijking met het fijne model, maar de evaluatietijd van dit model is aanzienlijk korter. Het grove model is dus goedkoop.

De output van dit model noteren we als $\mathbf{g}(\mathbf{x}_g) \in \mathbb{R}^m$ met $\mathbf{x}_g \in X_g \subset \mathbb{R}^n$. Het verschil tussen de oplossing van het grove model en de gewenste oplossing is dan $||\mathbf{g}(\mathbf{x}_g) - \mathbf{y}||$. Dus het bijbehorende optimum is

$$\mathbf{x}_g^* = \text{argmin}_{\mathbf{x}_g \in X_g} ||\mathbf{g}(\mathbf{x}_g) - \mathbf{y}||.$$

Evenals in het geval van het fijne model nemen we aan dat $\mathbf{g}(\mathbf{x}_g)$ continu en differentieerbaar is en dat X_g een gesloten en begrensde verzameling vormt.

Het is niet moeilijk om in te zien dat beide optima $\mathbf{x}_f^*$ en $\mathbf{x}_g^*$ bestaan. Dit volgt namelijk uit het feit dat zowel $\mathbf{f}(\mathbf{x}_f)$ als $\mathbf{g}(\mathbf{x}_g)$ continue functie is op een gesloten en begrensd interval. Bovendien als X_f en X_g zorgvuldig worden gereduceerd, worden $\mathbf{x}_f^*$ en $\mathbf{x}_g^*$ uniek bepaald.

3 Space Mapping

Space Mapping is net als Manifold Mapping een techniek die gebruik maakt van een grof en een fijn model. Het grove model wordt gecorrigeerd door de aanpassing van de ontwerpvariabelen, maar niet van de output van het model. Hieronder wordt het principe van de Space Mapping beschreven.

3.1 Beschrijving

We bekijken het verschil in de output van het fijne en het grove model:

$$||\mathbf{g}(\mathbf{x}_g) - \mathbf{f}(\mathbf{x}_f)||.$$

Stel dat $\mathbf{x}_f \in X_f$ gegeven is. Dan is het handig om te weten welke $\mathbf{x}_g \in X_g$ $||\mathbf{g}(\mathbf{x}_g) - \mathbf{f}(\mathbf{x}_f)||$ minimaliseert. Deze informatie kan worden gebruikt om het grove model te verbeteren. Hiervoor wordt de space-mapping functie geïntroduceert. De space-mapping functie $\mathbf{p} : X_f \to X_g$ ziet er als volgt uit:

$$\mathbf{p}(\mathbf{x}_f) = \text{argmin}_{\mathbf{x}_g \in X_g} ||\mathbf{g}(\mathbf{x}_g) - \mathbf{f}(\mathbf{x}_f)||.$$

Uit de bovenstaande definitie volgt dat $\mathbf{g}(\mathbf{p}(\mathbf{x}_f)) \approx \mathbf{f}(\mathbf{x}_f)$. Het space-mapping optimum is dan

$$\mathbf{x}^*_{sm} = \text{argmin}_{\mathbf{x}_f \in X_f} ||\mathbf{g}(\mathbf{p}(\mathbf{x}_f)) - \mathbf{y}||.$$

Deze methode levert dus een benadering van het optimum van het fijne model. Zie [1] voor het algoritme van de Space Mapping.

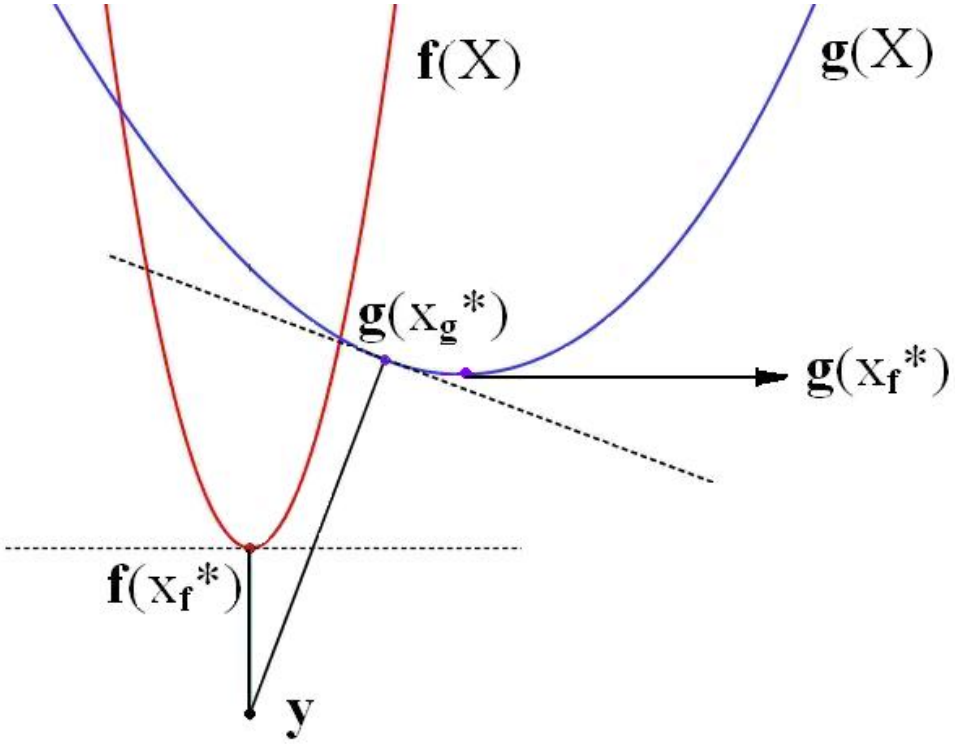
3.2 Voor- en nadelen

Elke evaluatie van de space-mapping functie is minstens zo duur als een evalutie van het fijne model. In de praktijk brengt Space Mapping voordelen mee, omdat fijne en grove modellen van een ontwerpprobleem op elkaar lijken. Hierdoor wordt $\mathbf{p}$ bijna een lineaire functie, die makelijker kan worden benaderd dan het fijne model.
Het nadeel van dit algoritme is dat dikwijls $\mathbf{x}^*_{sm}$ niet samenvalt met $\mathbf{x}^*_f$. De space-mapping functie is *perfect* als $\mathbf{p}(\mathbf{x}^*_f) = \mathbf{x}^*_g$. Als $\mathbf{p}$ perfect en injectief is, dan $\mathbf{x}^*_{sm} = \mathbf{x}^*_f$. De injectiviteit levert meestal geen moeilijkheden op, maar er zijn heel weinig problemen waarbij space-mapping functie perfect is. Hoewel vaak is de space-mapping functie bij benadering perfect, waardoor de methode een acceptabele oplossing vindt.

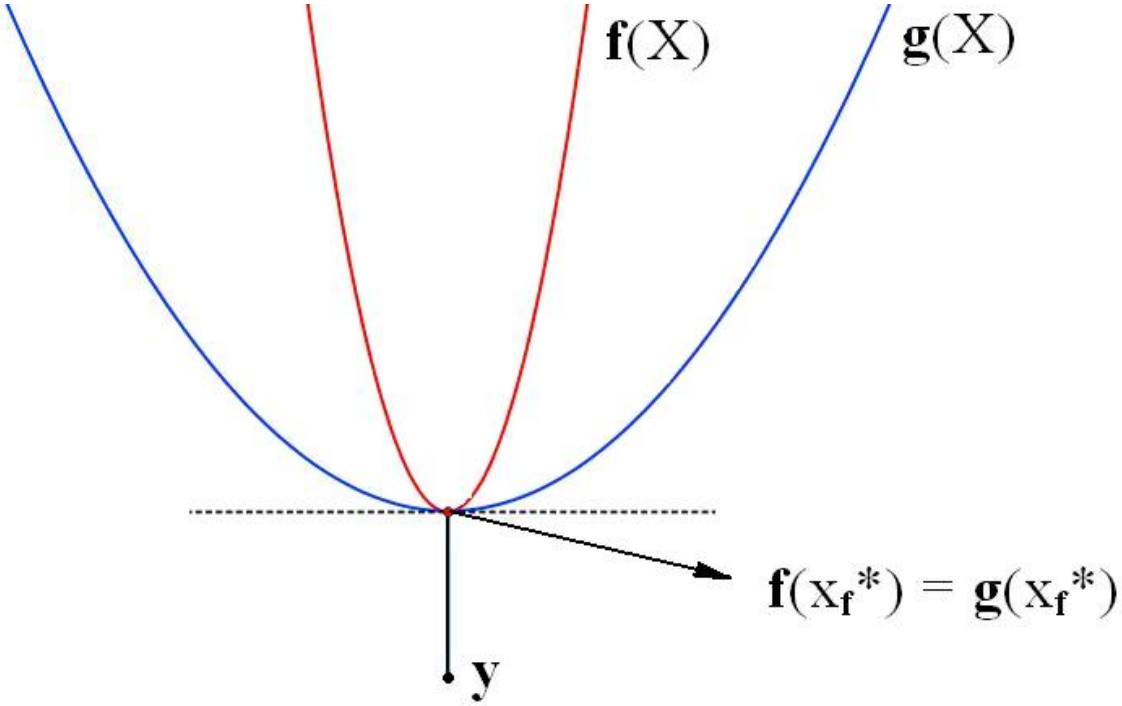
4 Manifold Mapping

4.1 Intuïtieve uitleg

De kwaliteit van de benadering die de Space Mapping oplevert, is afhankelijk van het gedrag van $\mathbf{f}(X_f)$ en $\mathbf{g}(X_g)$. In het algemeen verschillen deze verzamelingen in de buurt van de gewenste oplossing $\mathbf{y}$. Daarom vallen $\mathbf{x}^*_{sm}$ en $\mathbf{x}^*_f$ niet samen. We beschouwen zo'n situatie. De outputten



Figuur 1: Typisch gedrag van $\mathbf{f}(X_f)$ en $\mathbf{g}(X_g)$ in de buurt van de gewenste oplossing $\mathbf{y}$.



Figuur 2: Het getransleerde en geroteerde grove model en het fijne model.

$\mathbf{f}(X_f)$ en $\mathbf{g}(X_g)$ zijn te zien in Figuur 1. In dit hoofdstuk nemen we aan dat $\mathbf{x}_f^*$ zowel een element van X_f als van X_g is.
Met behulp van een translatie en een rotatie kan de output van het grove model worden gecorrigeerd zodat het op $\mathbf{f}(X_f)$ gaat lijken. De translatie wordt zo gekozen dat het punt $\mathbf{g}(\mathbf{x}_f^*)$ wordt afgebeeld op $\mathbf{f}(\mathbf{x}_f^*)$. Vervolgens wordt het grove model geroteerd om $\mathbf{g}(\mathbf{x}_f^*)$ zodanig dat de raaklijn aan $\mathbf{g}(X)$ in $\mathbf{g}(\mathbf{x}_f^*)$ wordt afgebeeld op de raaklijn aan $\mathbf{f}(X)$ in punt $\mathbf{f}(\mathbf{x}_f^*)$. De rotatie zorgt dus ervoor dat de output van beide modellen die bij $\mathbf{x}_f^*$ hoort, de kleinste afstaand oplevert tot de gewenste oplossing $\mathbf{y}$.

Het getransleerde en geroteerde grove model vindt dus hetzelfde optimum als het fijne model. Bovendien zijn de outputten van beide modellen in het optimum gelijk aan elkaar. Het gecorrigeerde grove model en fijne model zijn weergegeven in Figuur 2.

Vaak komt het voor dat de afstaand tussen $\mathbf{g}(\mathbf{x}_g^*)$ en $\mathbf{y}$ kleiner is dan de afstaand tussen $\mathbf{f}(\mathbf{x}_f^*)$ en $\mathbf{y}$. Met andere woorden $||\mathbf{g}(\mathbf{x}_g) - \mathbf{y}|| < ||\mathbf{f}(\mathbf{x}_f) - \mathbf{y}||$. Het lijkt dan alsof het grove model een betere oplossing oplevert dan het fijne model. Maar gezien het feit dat het grove model minder goed de realiteit beschrijft, wil men alsnog dat $\mathbf{g}(X_g)$ op $\mathbf{f}(X_f)$ in de buurt van het optimum gaat lijken.

4.2 Beschrijving

Manifold Mapping is een techniek die hetzelfde effect heeft op het grove model als hierboven beschreven translatie en rotatie. De Manifold Mapping wordt gedefinieerd als een affiene transformatie (m.a.w. een translatie en een lineaire transformatie) $\mathbf{S} : \mathbf{g}(X_g) \to \mathbf{f}(X_f)$ zodanig dat $\mathbf{S}(\mathbf{g}(\mathbf{x}_f^*)) = \mathbf{f}(\mathbf{x}_f^*)$ en het raakvlak aan $\mathbf{g}(X)$ in $\mathbf{g}(\mathbf{x}_f^*)$ op het raakvlak aan $\mathbf{f}(X)$ in $\mathbf{f}(\mathbf{x}_f^*)$ wordt afgebeeld. De transformatie $\mathbf{S}$ is niet uniek.

Hieronder bekijken we een aantal mogelijke transformaties en twee algoritmen voor één van deze transformaties. Deze algoritmen zijn ook te gebruiken voor elke andere goed gedefinieerde $\mathbf{S}$. Alle Manifold Mapping algoritmen maken gebruik van Singuliere Waarden Ontbinding. Daarom wordt deze matrix-factorisatie techniek uitgelegd in (9.1).

Voor het gemak wordt aangenomen dat $X_f = X_g = X$ en mogen we dus $\mathbf{x}_f$ en $\mathbf{x}_g$ vervangen door $\mathbf{x}$. De notatie voor de optima blijft onveranderd. (Als in de werkelijkheid $X_f \neq X_g$, dan kan er een niet-singuliere functie gedefinieerd worden van X_f naar X_g. Het invoeren van zo'n functie zou geen invloed hebben op het principe van de Manifold Mapping.)
Bovendien wordt alleen het geval $n < m$ besproken, waarbij n en m respectievelijk de dimensie van de input en de output zijn. Bij $n \geq m$ kan het probleem opgelost worden met behulp van een niet-lineair systeem.
Deze notatie en de aanname dat n kleiner is dan m worden ook gebruikt in de rest van het verslag.

Er was al gezegd dat $\mathbf{f}$ en $\mathbf{g}$ continue en differentieerbare functies zijn. Het is bekend dat rond $\mathbf{x}_f^*$ $\mathbf{f}(\mathbf{x})$ en $\mathbf{g}(\mathbf{x})$ bij benadering gelijk zijn aan hun lineariseringen. Oftewel,

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_f^*) + J_f(\mathbf{x}_f^*)(\mathbf{x} - \mathbf{x}_f^*), \tag{1}$$

$$\mathbf{g}(\mathbf{x}) \approx \mathbf{g}(\mathbf{x}_f^*) + J_g(\mathbf{x}_f^*)(\mathbf{x} - \mathbf{x}_f^*). \tag{2}$$

Hier is J_f de Jacobiaan van $\mathbf{f}$ en J_g is de Jacobiaan van $\mathbf{g}$. Zowel J_f als J_g is een $m \times n$-matrix

en heeft rang n. Uit (2) volgt dat

$$J_g(\mathbf{x}_f^*)(\mathbf{x} - \mathbf{x}_f^*) \approx \mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_f^*).$$

Omdat J_g geen vierkante matrix is, bestaat er geen inverse van J_g. Daarom vervangen we J_g^{-1} door de pseudo-inverse matrix $J_g^\dagger$ die bepaald wordt met behulp van de Singuliere Waarden Ontbinding. Bij de pseudo-inverse matrices geldt nog steeds dat $J_g^\dagger J_g = I$. Voor verdere uitleg zie (9.1).
Hieruit volgt dat

$$J_g^\dagger(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*)(\mathbf{x} - \mathbf{x}_f^*) \approx J_g^\dagger(\mathbf{x}_f^*)(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_f^*)).$$

En dus

$$\mathbf{x} - \mathbf{x}_f^* \approx J_g^\dagger(\mathbf{x}_f^*)(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_f^*)). \tag{3}$$

Door (3) in te vullen in (2) krijgen we het volgende resultaat:

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_f^*) + J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_f^*)).$$

Kies nu de Manifold Mapping transformatie gelijk aan

$$\mathbf{S}(\mathbf{g}(\mathbf{x})) = \mathbf{f}(\mathbf{x}_f^*) + S(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_f^*)),$$

waarbij $S = J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)$ een $m \times m$ matrix is met rang n.
Merk op: deze transformatie voldoet exact aan de definitie van de Manifold Mapping. Namelijk,

$$\mathbf{S}(\mathbf{g}(\mathbf{x}_f^*)) = \mathbf{f}(\mathbf{x}_f^*) + S(\mathbf{g}(\mathbf{x}_f^*) - \mathbf{g}(\mathbf{x}_f^*) = \mathbf{f}(\mathbf{x}_f^*).$$

Ook geldt er dat

$$J_{mm} = \frac{\partial \mathbf{S}(\mathbf{g})}{\partial \mathbf{x}} = \frac{\partial \mathbf{S}}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = SJ_g,$$

waarbij J_{mm} de Jacobiaan van de Manifold Mapping is. Hieruit volgt dat

$$J_{mm}(\mathbf{x}_f^*) = SJ_g(\mathbf{x}_f^*) = J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*) = J_f(\mathbf{x}_f^*).$$

Dit betekent dat de raakvlakken van het fijne model en de Manifold Mapping samenvallen in de buurt van de oplossing.
$\mathbf{S}(\mathbf{g}(\mathbf{x}))$ is dus bij benadering gelijk aan $\mathbf{f}(\mathbf{x})$ voor $\mathbf{x}$ in de buurt van het optimum $\mathbf{x}_f^*$. Dit betekent dat de optimalisatie van de Manifold Mapping hetzelfde optimum oplevert als de optimalisatie van het het fijne model.

We hebben al opgemerkt dat de transformatie $\mathbf{S}$ niet uniek is. Een andere mogelijke transformatie kun je creëren door $S = J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)$ te vervangen door bijvoorbeeld

$$S = J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*) + A(I - U_g U_g^T)$$

met A een $m \times m$ matrix en U_g zodanig dat $J_g = U_g \Sigma_g V_g^T$.
Het is niet moeilijk om in te zien dat in dit geval $\mathbf{S}(\mathbf{g}(\mathbf{x}_f^*)) = \mathbf{f}(\mathbf{x}_f^*)$. Verder geldt er ook weer dat $J_{mm} = SJ_g$. We moeten nog laten zien dat $J_{mm}(\mathbf{x}_f^*) = J_f(\mathbf{x}_f^*)$.

$$J_{mm}(\mathbf{x}_f^*) = (J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*) + A(I - U_g U_g^T))J_g(\mathbf{x}_f^*) =$$

$$J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*) + A(J_g(\mathbf{x}_f^*) - U_g U_g^T J_g(\mathbf{x}_f^*)) =$$

$$J_f(\mathbf{x}_f^*) + A(J_g(\mathbf{x}_f^*) - U_g U_g^T U_g \Sigma_g V_g) =$$

$$J_f(\mathbf{x}_f^*) + A(J_g(\mathbf{x}_f^*) - U_g I \Sigma_g V_g) =$$

$$J_f(\mathbf{x}_f^*) + A(J_g(\mathbf{x}_f^*) - J_g(\mathbf{x}_f^*)) = J_f(\mathbf{x}_f^*).$$

Hieruit volgt dat deze transformatie ook voldoet aan onze definitie.
Je kunt ook een andere S kiezen zolang er het volgende geldt:

$$SJ_g(\mathbf{x}_f^*) = J_f(\mathbf{x}_f^*).$$

Omdat deze transformaties zeer op elkaar lijken, gaan we verder met

$$\mathbf{S}(\mathbf{g}(\mathbf{x})) = \mathbf{f}(\mathbf{x}_f^*) + J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{x}_f^*)).$$

Voor andere goed gedefinieerde transformaties verandert de theorie van dit Hoofdstuk niet.

Het is opvallend dat de hierboven beschreven Manifold Mapping transformaties gebaseerd zijn op het optimum van het fijne model, terwijl ons doel was zo min mogelijk met het fijne model te werken. Om de rekentijd toch kort te houden, wordt transformatie $\mathbf{S}$ benadert met $\mathbf{S}_k$ ($k \geq 0$), die tijdens het iteratieve optimaliseringproces bepaald wordt. Voor elke k is er dus een matrix S_k die $S = J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)$ benadert. We beginnen met de definitie van $J_f(\mathbf{x})$:

$$J_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix},$$

waarbij

$$\frac{\partial f_j}{\partial x_i} = \lim_{\Delta x_i \to 0} \frac{f(D_j, x_1, x_2, ..., x_{i-1}, x_i + \Delta x_i, x_{i+1}, ..., x_n) - f(D_j, x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)}{\Delta x_i}$$

voor $i = 1, 2, ..., n$ en $j = 1, 2, ..., m$.
Dit noteren we kort als

$$J_f(\mathbf{x}) = \left[\begin{array}{cccc} \frac{(\partial \mathbf{f})_1}{\partial x_1} & \frac{(\partial \mathbf{f})_2}{\partial x_2} & \cdots & \frac{(\partial \mathbf{f})_n}{\partial x_n} \end{array} \right].$$

Met behulp van voorwaartse differentie kan deze matrix als volgt benaderd worden:

$$J_f(\mathbf{x}) \approx \left[\begin{array}{cccc} \frac{(\Delta \mathbf{f})_1}{\Delta x_1} & \frac{(\Delta \mathbf{f})_2}{\Delta x_2} & \cdots & \frac{(\Delta \mathbf{f})_n}{\Delta x_n} \end{array} \right],$$

waarbij Δx_i het verschil is tussen x_i bij iteratie $k+1$ en x_i bij iteratie k en

$$(\Delta \mathbf{f})_i = \mathbf{f}(x_1, x_2, ..., x_{i-1}, x_i + \Delta x_i, x_{i+1}, ..., x_n) - \mathbf{f}(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)$$

voor $i = 1, ..., n$.
Het is niet moeilijk om in te zien dat er geldt

$$J_f(\mathbf{x}) \approx \left[\begin{array}{cccc} (\Delta \mathbf{f})_1 & (\Delta \mathbf{f})_2 & \cdots & (\Delta \mathbf{f})_n \end{array} \right] \left[\begin{array}{cccc} \frac{1}{\Delta x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\Delta x_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\Delta x_n} \end{array} \right].$$

Een soortgelijke benadering geldt er ook voor $J_g(\mathbf{x})$. Dan

$$J_g^\dagger(\mathbf{x}) \approx \left[\begin{array}{cccc} \frac{1}{\Delta x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\Delta x_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\Delta x_n} \end{array} \right]^\dagger \left[\begin{array}{cccc} (\Delta \mathbf{g})_1 & (\Delta \mathbf{g})_2 & \cdots & (\Delta \mathbf{g})_n \end{array} \right]^\dagger$$

met

$$(\Delta \mathbf{g})_i = \mathbf{g}(x_1, x_2, ..., x_{i-1}, x_i + \Delta x_i, x_{i+1}, ...x_n) - \mathbf{g}(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ...x_n)$$

voor $i = 1, ..., n$.
Merk op:

$$\left[\begin{array}{cccc} \frac{1}{\Delta x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\Delta x_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\Delta x_n} \end{array} \right]^\dagger = \left[\begin{array}{cccc} \Delta x_1 & 0 & \cdots & 0 \\ 0 & \Delta x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Delta x_n \end{array} \right].$$

Hieruit volgt dat

$$J_f(\mathbf{x}) J_g^\dagger(\mathbf{x}) = \left[\begin{array}{cccc} (\Delta \mathbf{f})_1 & (\Delta \mathbf{f})_2 & \cdots & (\Delta \mathbf{f})_n \end{array} \right] \left[\begin{array}{cccc} (\Delta \mathbf{g})_1 & (\Delta \mathbf{g})_2 & \cdots & (\Delta \mathbf{g})_n \end{array} \right]^\dagger.$$

Door in de bovenstaande uitdrukking $\mathbf{x} = \mathbf{x}_f^*$ in te vullen, krijg je een benadering van $S = J_f(\mathbf{x}_f^*)J_g^\dagger(\mathbf{x}_f^*)$. We noteren S_k als

$$S_k = \Delta F(\mathbf{x}_k)\Delta G^\dagger(\mathbf{x}_k).$$

Er bestaan verschillende algoritmen die Manifold Mapping techniek vertegenwoordigen. Dit wordt veroorzaakt door het feit dat je zowel de gewenste oplossing kan vasthouden en $\mathbf{S}_k(\mathbf{g})$ kunt aanpassen tijdens elke iteratie als het model onveranderd laten en $\mathbf{y}$ corrigeren tijdens elke stap van de optimalisatie. Hieronder worden twee mogelijke algoritmen beschreven. Algoritme 1 bewerkt steeds $\mathbf{S}_k(\mathbf{g})$ en Algoritme 2 past $\mathbf{y}$ aan.

4.3 Algoritme 1

1. Neem $k = 0$ en $S_0 = I_{m\times m}$. Bepaal het optimum van het grove model:

   $$\mathbf{x}_g^* = \text{argmin}_{\mathbf{x}_g\in X}||\mathbf{g}(\mathbf{x}_g) - \mathbf{y}||.$$

   Laat $\mathbf{x}_k = \mathbf{x}_g^*$.

2. Bereken $\mathbf{f}(\mathbf{x}_k)$ en $\mathbf{g}(\mathbf{x}_k)$.

3. Bepaal $\mathbf{x}_{k+1} = \text{argmin}_{\mathbf{x}\in X_f}||\mathbf{S}_k(\mathbf{g}(\mathbf{x})) - \mathbf{y}||$.

4. Vervang k door $k+1$. Reken uit $\mathbf{f}(\mathbf{x}_k)$ en $\mathbf{g}(\mathbf{x}_k)$.

5. Bepaal

   $$\Delta F = \Delta F(\mathbf{x}_k) = \left[\mathbf{f}(\mathbf{x}_k) - \mathbf{f}(\mathbf{x}_{k-1}), ..., \mathbf{f}(\mathbf{x}_k) - \mathbf{f}(\mathbf{x}_{\max(k-n,0)})\right],$$

   $$\Delta G = \Delta G(\mathbf{x}_k) = \left[\mathbf{g}(\mathbf{x}_k) - \mathbf{g}(\mathbf{x}_{k-1}), ..., \mathbf{g}(\mathbf{x}_k) - \mathbf{g}(\mathbf{x}_{\max(k-n,0)})\right].$$

   Met behulp van Singuliere Waarden Ontbinding van ΔG bereken $U_{\Delta G}, \Sigma_{\Delta G}$ en $V_{\Delta G}$. Reken uit $\Delta G^\dagger = V_{\Delta G}\Sigma_{\Delta G}^\dagger U_{\Delta G}^T$.

6. Neem $S_k = \Delta F\Delta G^\dagger$. Bepaal

   $$\mathbf{x}_{k+1} = \text{argmin}_{\mathbf{x}\in X}||\mathbf{S}_k(\mathbf{g}(\mathbf{x})) - \mathbf{y}||,$$

   waarbij $\mathbf{S}_k\bullet = \mathbf{f}(\mathbf{x}_k) + S_k(\bullet - \mathbf{g}(\mathbf{x}_k))$.

7. Controleer de stopvoorwaarde. Als de huidige oplossing niet aan deze voorwaarde voldoet, ga terug naar stap 4.

4.4 Algoritme 2

1. Neem $k = 0$ en $T_0 = I_{m\times m}$. Bepaal het optimum van het grove model:

$$\mathbf{x}_g^* = \text{argmin}_{\mathbf{x}_g \in X} ||\mathbf{g}(\mathbf{x}_g) - \mathbf{y}||.$$

   Laat $\mathbf{x}_k = \mathbf{x}_g^*$.

2. Bereken $\mathbf{f}(\mathbf{x}_k)$ en $\mathbf{g}(\mathbf{x}_k)$.

3. Definieer $\mathbf{y}_k = \mathbf{g}(\mathbf{x}_k) - T_k(\mathbf{f}(\mathbf{x}_k) - y)$. Bepaal $\mathbf{x}_{k+1} = \text{argmin}_{\mathbf{x}\in X_f} ||\mathbf{g}(\mathbf{x}) - \mathbf{y}_k||$.

4. Vervang k door $k + 1$. Reken uit $\mathbf{f}(\mathbf{x}_k)$ en $\mathbf{g}(\mathbf{x}_k)$.

5. Bepaal

$$\Delta F = \Delta F(\mathbf{x}_k) = \left[\mathbf{f}(\mathbf{x}_k) - \mathbf{f}(\mathbf{x}_{k-1}), ..., \mathbf{f}(\mathbf{x}_k) - \mathbf{f}(\mathbf{x}_{\max(k-n,0)})\right],$$

$$\Delta G = \Delta G(\mathbf{x}_k) = \left[\mathbf{g}(\mathbf{x}_k) - \mathbf{g}(\mathbf{x}_{k-1}), ..., \mathbf{g}(\mathbf{x}_k) - \mathbf{g}(\mathbf{x}_{\max(k-n,0)})\right].$$

   Met behulp van Singuliere Waarden Ontbinding van ΔF bereken $U_{\Delta F}, \Sigma_{\Delta F}$ en $V_{\Delta F}$. Reken uit $\Delta F^\dagger = V_{\Delta F}\Sigma_{\Delta F}^\dagger U_{\Delta F}^T$.

6. Neem $T_k = \Delta G \Delta F^\dagger$. Bepaal

$$\mathbf{x}_{k+1} = \text{argmin}_{\mathbf{x}\in X} ||\mathbf{g}(\mathbf{x}) - \mathbf{y}_k||$$

   met $\mathbf{y}_k = \mathbf{g}(\mathbf{x}_k) - T_k(\mathbf{f}(\mathbf{x}_k) - y)$.

7. Controleer de stopvoorwaarde. Als de huidige oplossing niet aan deze voorwaarde voldoet, ga terug naar stap 4.

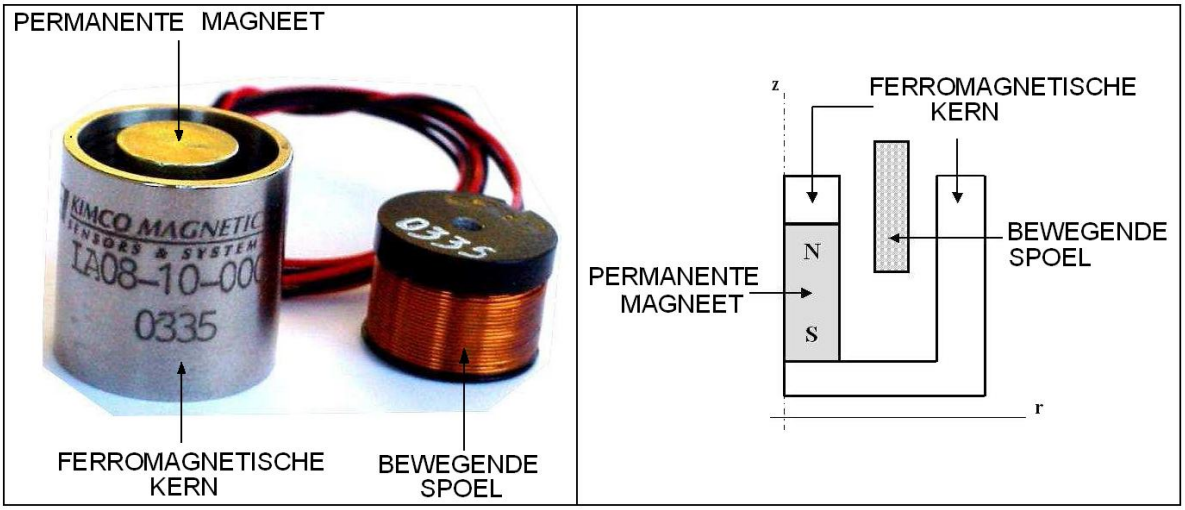
Aan de hand van 4.3 en 4.4 zien we dat de manifold-mapping techniek uit meerdere optimalisatieniveaus bestaat.

5 Toepassing in elektromagnetisme

5.1 Probleemstelling

Om elektrische signalen in trillingen om te zetten worden vaak voice-coil actuatoren gebruikt. De typische toepassingen van voice-coil actuatoren zijn luidsprekers (vandaar ook de naam), harde schijven, lasersnijmachines en pick-n-place systemen.

Een voice-coil actuator bevat twee belangrijke onderdelen: een spoel en een permanente magneet. De magneet is ingebouwd in een ferromagnetische kern en de spoel is vrij om te bewegen in de luchtspleet binnen de kern. In dit verslag wordt een cylindrische voice-coil actuator besproken. Cylinder symmetrie maakt het mogelijk om zonder verlies van algemeenheid de structuur van een voice-coil actuator voor een bepaalde hoek te beschouwen (zie Figuur 3).



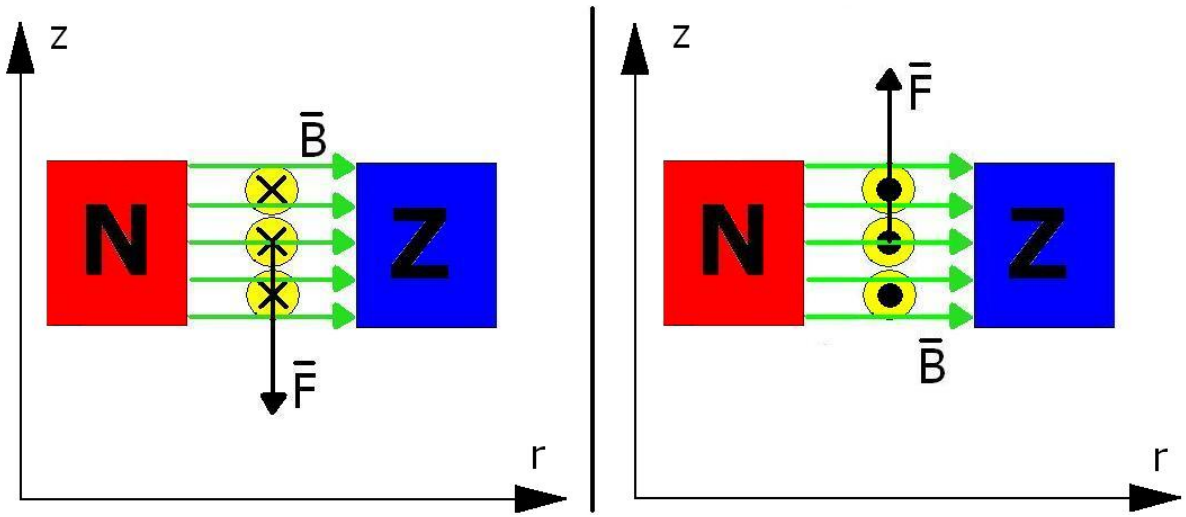
Figuur 3: Links: voice-coil actuator. Rechts: het probleem voor een bepaalde hoek.

Omdat de magneet in een ferromagnetische kern ingebouwd is, ontstaat er een magnetisch veld in de luchtspleet. Wanneer de spoel (deels) in de spleet wordt geplaatst, gaat er een kracht op de spoel werken. Dit volgt uit de wet van Lorentz. Deze wet zegt dat een stroomvoerende geleider, geplaatst in een magnetisch veld, een kracht ondervindt:

$$\mathbf{F} = \int (\mathbf{I} \times \mathbf{B}) da.$$

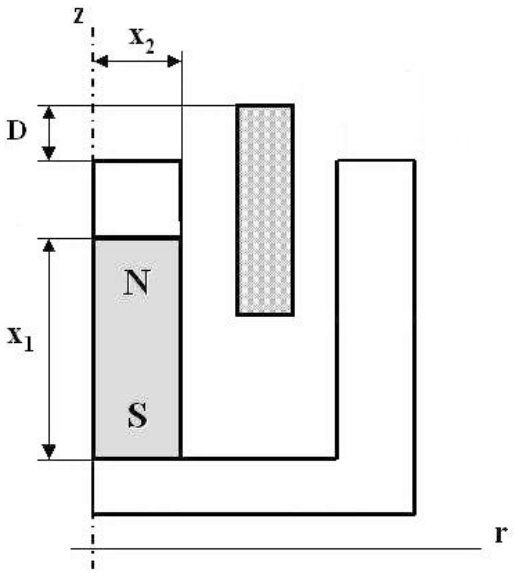
Hierbij is $\mathbf{I}$ de stroom door de spoel, $\mathbf{B}$ de magnetische fluxdichtheid en a is de oppervlakte van de spoel.

De richting van Lorentzkracht $\mathbf{F}$ is dus afhankelijk van de richting van de stroom en van het magnetisch veld. Het magnetisch veld is constant. Bij een versimpelde situatie kan aangenomen worden dat het veld uniform is.
Door de spoel loopt een wisselstroom. Hierdoor verandert de richting van de Lorentzkracht. Een versimpelde situatie is weergegeven in Figuur 4.



Figuur 4: Versimpelde situatie: drie windingen binnen de luchtspleet. Links: stroom papier in. Rechts: stroom papier uit.

We zien dat als de stroom het papier in gaat, wijst de Lorentzkracht naar beneden. Hierdoor gaat de spoel naar beneden bewegen. Maar als de vector **I** het papier uit wijst, ontstaat er een naar boven gerichte kracht, waardoor de spoel naar boven gaat bewegen. De spoel gaat dus trillen. Zo worden elektrische signalen met behulp van voce-coil actuatoren in trillingen omgezet.



Figuur 5: Schematische weergave van het probleem.

De afstand D van de bovenkant van de spoel tot de bovenkant van de ferromagnetische kern varieert van 0 tot 4 mm (zie Figuur 5). De totale kracht op de spoel wordt gemeten voor negen equidistante posities van de spoel D_i (met i van 1 tot en met 9) binnen dit interval. Het doel is het vinden van straal x_1 en dikte x_2 van de magneet zodanig dat de kracht voor alle negen posities ongeveer gelijk wordt aan 12 N. Hiermee kan men bijvoorbeeld voorkomen dat de spoel

tegen de onderkant van de kern aan knalt. Bovendien moet er gelden dat $x_1 \in [7\text{ mm}, 45\text{ mm}]$ en $x_2 \in [1\text{ mm}, 45\text{ mm}]$.

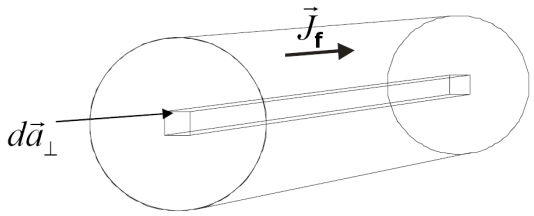
5.2 Fijn model

5.2.1 Beschrijving

Om de gevraagde Lorentzkracht op de spoel te vinden, moeten we zowel de fluxdichtheid van het magnetische veld **B** kennen als de elektrische stroom **I** die door de spoel loopt. Deze waarden moeten bepaald worden aan de hand van afmetingen van de voice-coil actuator, eigenschappen van de materialen en volumestroomdichtheid in de spoel.

In dit hoofdstuk wordt een afleiding gegeven van de parti/ële differentiaalvergelijking voor de spoel en de kern. De vergelijking van de magneet is soortgelijk, maar houdt ook rekening mee met de magnetisatie.

Volumestroomdichtheid is gedefinëerd als stroom per oppervlakte-eenheid loodrecht op de stroom en is weergegeven in Figuur 6.



Figuur 6: Volumestroomdichtheid en oppervlakte-eenheid loodrecht op de stroom.

Dus $\mathbf{J}_f = \frac{d\mathbf{I}}{d\mathbf{a}_\perp}$.
In onze situatie is van belang dat $\mathbf{J}_f$ gedurende de tijd van richting verandert. In 5.1 hebben we al uitgelegd dat hierdoor de spoel ten opzichte van de magneet gaat bewegen. Dit zorgt ook voor de verandering van de magnetische flux die door de spoel loopt. De wet van Faraday zegt:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

met **E** het elektrische veld. Hieruit volgt dat er een geïnduceerd elektrisch veld onstaat in de spoel. Dit veld genereert op zijn beurt geïnduceerde stromen, Eddy currents. Dit is een consequentie van de wet van Ohm:

$$\mathbf{E} = \frac{\mathbf{J}}{\sigma}.$$

Hierbij is σ de geleidbaarheid van een materiaal. Eddy currents werken tegen de verandering in magnetische flux. Per definitie geldt er:

$$\text{Eddy currents } \propto \sigma \int (\mathbf{v} \times \mathbf{B}) \cdot d\mathbf{a},$$

waarbij **v** de snelheid is waarmee de spoel beweegt. Dus de stroom die uiteindelijk door de spoel

heen gaat $\mathbf{J}_{tot}$ wordt gelijk aan de "vrije" stroom $\mathbf{J}_f$ en de totale stroom die geïnduceerd is. $\mathbf{J}_{tot}$ kan als volgt uitgedrukt worden (zie [4]):

$$\mathbf{J}_{tot} = \sigma\mathbf{E} + \sigma(\mathbf{v} \times \mathbf{B}) + \mathbf{J}_f. \tag{4}$$

Uit de verbeterde door Maxwell versie van de wet van Ampère volgt:

$$\frac{1}{\mu}(\nabla \times \mathbf{B}) = \mathbf{J}_{tot} + \frac{\partial \mathbf{D}}{\partial t}. \tag{5}$$

Hierin is μ de permeabiliteit van het lineaire medium. $\mathbf{D}$ is de elektrische verplaatsing.

Let op: we gebruiken voor de elektrische verplaatsing dezelfde notatie als voor de vector van onafhankelijke variabelen in (2.1). In de rest van het verslag zullen we proberen duidelijk aan te geven over welke $\mathbf{D}$ het gaat.

Door (4) en (5) te combineren, krijgen we de volgende uitdrukking:

$$\frac{1}{\mu}(\nabla \times \mathbf{B}) = \sigma\mathbf{E} + \sigma(\mathbf{v} \times \mathbf{B}) + \mathbf{J}_f + \frac{\partial \mathbf{D}}{\partial t}. \tag{6}$$

Verder bestaat er een wet in elektromagnetisme die zegt dat de magnetische flux door een gesloten oppervlak gelijk is aan nul:

$$\nabla \cdot \mathbf{B} = 0.$$

$\mathbf{B} = \nabla \times \mathbf{A}$ voldoet altijd aan de bovenstaande eis, want voor elke vector $\mathbf{u}$ geldt:

$$\nabla \cdot (\nabla \times \mathbf{u}) = 0.$$

Bovendien is bekend dat $\nabla \times (\nabla \mathbf{u}) = 0$. We kunnen dus altijd een gradiënt bij $\mathbf{A}$ optellen zodanig dat $\nabla \cdot \mathbf{A} = 0$. Zo'n $\mathbf{A}$ wordt de magnetische vectorpotentiaal genoemd.

Door het invullen van $\mathbf{B} = \nabla \times \mathbf{A}$ in (6) wordt het volgende resultaat verkregen:

$$\frac{1}{\mu}(\nabla \times (\nabla \times \mathbf{A})) - \sigma(\mathbf{v} \times (\nabla \times \mathbf{A})) = \sigma\mathbf{E} + \mathbf{J}_f + \frac{\partial \mathbf{D}}{\partial t}. \tag{7}$$

Bovendien geldt voor elektrodynamische problemen dat $\mathbf{E} = -\nabla V - \frac{\partial A}{\partial t}$. Dit geeft:

$$\frac{1}{\mu}(\nabla \times (\nabla \times \mathbf{A})) - \sigma(\mathbf{v} \times (\nabla \times \mathbf{A})) = -\sigma\nabla V - \sigma\frac{\partial A}{\partial t} + \mathbf{J}_f + \frac{\partial \mathbf{D}}{\partial t}. \tag{8}$$

De oplossing van deze partiële differentiaalvergelijking samen met de randvoorwaarden levert ons de magnetische vectorpotentiaal op en dus ook $\mathbf{B}$, waarmee de Lorentzkracht op de spoel uitgerekent kan worden.

5.2.2 *COMSOL*

Het is duidelijk dat (8) en de analoge vergelijking voor de magneet te ingewikkeld zijn om analytisch opgelost te worden. Daarom wordt de oplossing benadert met behulp van eindige-elementenmethode. Deze methode deelt de constructie op in een aantal elementen en verbindt ze met behulp van knooppunten. Er wordt dus een rekenrooster gemaakt. Aan elke knoop worden bepaalde eisen gesteld. Deze eisen zijn afhankelijk van het oorspronkelijke probleem en het soort van element. Het gedrag van de elementen is geanalyseerd aan de hand van de knopen. De analyse van een element resulteert zich in een kleine matrix P die voldoet aan:

$$P\mathbf{w} = \mathbf{b},$$

waarbij $\mathbf{v}$ een vector met verplaatsingen van de knooppunten is en $\mathbf{b}$ een vector met krachten op de knooppunten.
Wanneer zo'n matrix voor elk element bepaald is, wordt er een grote matrix van het hele systeem gemaakt die al die kleine matrices combineert. We noemen deze matrix $\tilde{P}$. Het doel is het bepalen van de verplaatsingsvector $\tilde{\mathbf{w}}$ die voldoet aan $\tilde{P}\tilde{\mathbf{w}} = \tilde{\mathbf{b}}$ met $\tilde{\mathbf{b}}$ de kracht op de knooppunten in alle richtingen.

Om de oplossing van de partiële differentiaalvergelijking te vinden met eindige-elementenmethode, gebruiken we softwarepakket *COMSOL*. Dit pakket is in staat om een-, twee- en driedimensionale problemen op te lossen. *COMSOL* kan bovendien vanuit $MATLAB$ bediend worden.

In 5.1 hebben we al gezegd dat door cylinder symmetrie kunnen we het driedimensionale probleem vervangen door een tweedimensionaal probleem. Daarom wordt binnen *COMSOL* met *Azimuthal Induction Currents* class gewerkt. Bovendien is het mogelijk om in *COMSOL* de positie van de spoel aan te geven en dus een statische probleem te beschouwen. Dit maakt (8) aanzienlijk simpeler, want de tijdsafhankelijke termen verdwijnen:

$$\frac{1}{\mu}(\nabla \times (\nabla \times \mathbf{A})) - \sigma(\mathbf{v} \times (\nabla \times \mathbf{A})) = -\sigma\nabla V + \mathbf{J}_f. \tag{9}$$

Verder moet opgemerkt worden dat het elektrische veld door de cylinder symmetrie alleen in de ϕ-richting ongelijk aan nul is. Dus $\nabla V = -\frac{V_{loop}}{2\pi r}$. Het magnetische veld heeft juist geen component in de ϕ-richting en omdat $\mathbf{B} = \nabla \times \mathbf{A}$ moet $\mathbf{A}$ loodrecht staan op $\mathbf{B}$. $\mathbf{A}$ kan dus worden vervangen door $\mathbf{A}_\phi$. De vergelijking die wij uiteindelijk krijgen ziet er zo uit:

$$\frac{1}{\mu}(\nabla \times (\nabla \times \mathbf{A}_\phi)) - \sigma(\mathbf{v} \times (\nabla \times \mathbf{A}\phi)) = \sigma\frac{V_{loop}}{2\pi r} + \mathbf{J}_f. \tag{10}$$

COMSOL levert een numerieke benadering voor de oplossing van (10) voor de spoel en de kern en van analoge vergelijking voor de magneet zodra het rooster, de afmetingen, materiaaleigenschappen en randvoorwaarden gedefinieerd zijn. Met deze oplossing kan ook de Lorentzkracht op de spoel bepaald worden. Voor elke positie van de spoel moet de vergelijking opnieuw opgelost worden. Dus voor het vinden van $\mathbf{f}(\mathbf{x})$ hebben we negen verschillende oplossingen van de bovenstaande partiële differentiaalvergelijking nodig. Omdat de oplossing nauwkeurig moet zijn, kiezen we een redelijk fijne rekenrooster.
Bovendien is het fysisch relevant om de permeabiliteit van ijzer (die wij als het materiaal van de kern van de voice-coil actuator nemen) niet-lineair te maken. Helaas heeft *COMSOL* wat moeite

met het bepalen van de oplossing, als we de permeabiliteit gelijk afhankelijk van de magnetische fluxdichtheid maken. Daarom lossen we eerst het probleem op met constante permeabiliteit. We gebruiken de oplossing van lineaire probleem als beginoplossing van het niet-lineaire probleem. Deze twee factoren (fijne rekenrooster en niet-lineaire permeabiliteit) zorgen ervoor dat de evaluatietijd van het fijne model zeer lang wordt.

Voor meer informatie over het fijne model zie Hoofdstuk 9.2.

5.3 Grof model

Er bestaan meerdere technieken die gebruikt kunnen worden als grof model. Aan de ene kant zijn er grove modellen die onafhankelijk zijn van het fijne model. Hiermee wordt bedoeld dat er zonder $\mathbf{f}(\mathbf{x})$ wordt gewerkt. Dit kan bereikt worden door bijvoorbeeld het probleem lineair te maken of een ruwere rekenrooster te kiezen. Aan de andere kant zijn er technieken die wel gebruik maken van het fijne model. Eén van dit soort modellen is multiquadric interpolatie, die slechts een paar evaluaties van het fijne model heeft nodig om de fijne oplossing te benaderen. Deze interpolatietechniek wordt in ons voorbeeld als het grove model genomen. We baseren onze keuze aan het feit dat multiqudric interpolatie weinig tijd kost en flexibel is (zie [3]). Deze techniek levert goede benadering op voor ingewikkelde functies, die bijvoorbeeld een groot aantal extrema hebben.

5.3.1 Beschrijving

Uit de verzameling X van alle mogelijke ontwerpvariabelen worden N verschillende punten genomen. Voor elke van de N punten wordt het output van het fijne model bepaald.
We definiëren $\mathbf{q}(\mathbf{x}) = [q_1, q_2 ... q_N]^T$, waarbij multiquadric $q_j(\mathbf{x})$ $(j = 1, 2, ..., N)$ gelijk is aan

$$q_j(\mathbf{x}) = \sqrt{||\mathbf{x} - \mathbf{x}_j||^2 + h}.$$

Hier is h de zogenaamde *shift-parameter*, die gelijk is aan $\frac{\sum_{j=1}^{N} ||\mathbf{x}_j||}{KN}$ met K een constante die gekozen kan worden.
Vervolgens bepalen we hoe "zwaar" elke multiquadrics meetelt voor $\mathbf{g}(\mathbf{x})$ met $N \times m$ coëfficëntenmatrix a:

$$a = \begin{bmatrix} \alpha_1(D_1) & \alpha_1(D_2) & \cdots & \alpha_1(D_m) \\ \alpha_2(D_1) & \alpha_2(D_2) & \cdots & \alpha_2(D_m) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_N(D_1) & \alpha_N(D_2) & \cdots & \alpha_N(D_m) \end{bmatrix}.$$

Deze matrix moet voldoen aan:

$$a = Q^{-1}\tilde{F}.$$

Hierbij is Q een $N \times N$ matrix met multiquadrics geëvalueerd in de steunpunten:

$$Q = \begin{bmatrix} q_1(\mathbf{x}_1) & q_2(\mathbf{x}_1) & \cdots & q_N(\mathbf{x}_1) \\ q_1(\mathbf{x}_2) & q_2(\mathbf{x}_2) & \cdots & q_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ q_1(\mathbf{x}_N) & q_2(\mathbf{x}_N) & \cdots & q_N(\mathbf{x}_N) \end{bmatrix}.$$

Merk op: uit de definitie van multiquadric en onze keuze van steunpunten volgt dat Q symmetrisch en inverteerbaar is.
$N \times m$-matrix $\tilde{F}$ weergeeft de output van het fijne model voor de steunpunten:

$$\tilde{F} = \begin{bmatrix} f(D_1, \mathbf{x}_1) & f(D_2, \mathbf{x}_1) & \cdots & f(D_m, \mathbf{x}_1) \\ f(D_1, \mathbf{x}_2) & f(D_2, \mathbf{x}_2) & \cdots & f(D_m, \mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f(D_1, \mathbf{x}_N) & f(D_2, \mathbf{x}_N) & \cdots & f(D_m, \mathbf{x}_N) \end{bmatrix}.$$

De output van het grove model wordt dan als volgt uitgerekend:

$$\mathbf{g}(\mathbf{x}) = a^T q.$$

5.3.2 *Shift-parameter h*

Stel dat $n = m = 1$ en steunpunten zijn x_1 en x_2 met $x_1 < x_2$.
Als constante K heel groot is, dan $h \approx 0$. We krijgen dat $q = [||x - x_1||, ||x - x_2||]^T = [|x - x_1|, |x - x_2|]^T$. Bovendien geldt er

$$\begin{bmatrix} 0 & |x_1 - x_2| \\ |x_2 - x_1| & 0 \end{bmatrix} \begin{bmatrix} a_1(D_1) \\ a_2(D_1) \end{bmatrix} = \begin{bmatrix} f(D_1, x_1) \\ f(D_1, x_2) \end{bmatrix}.$$

Hieruit volgt dat

$$g(x) = \frac{|x - x_1|}{|x_2 - x_1|} f(x_2) + \frac{|x - x_2|}{|x_1 - x_2|} f(x_1) = \frac{x - x_1}{x_2 - x_1} f(x_2) + \frac{x - x_2}{x_1 - x_2} f(x_1).$$

Dus voor $n = m = 1$, $N = 2$ en K heel groot wordt de multiquadrics interpolatie gelijk aan lineaire interpolatie. Naarmate we K kleiner maken, wordt de benadering minder lineair. Dus h bepaalt de kromming van $\mathbf{g}(\mathbf{x})$. We nemen K gelijk aan 50, want uit [3] blijkt dat $K = 50$ in praktijk een goede keuze is.

5.4 Nelder-Mead simplex

We gebruiken Nelder-Mead simplex voor het vinden van $\text{argmin}_{\mathbf{x}_g \in X} ||\mathbf{g}(\mathbf{x}_g) - \mathbf{y}||$.
Nelder-Simplex wordt gebruikt voor de niet-lineaire problemen zonder voorwaarden. Deze methode vervangt één van de hoekpunten van het simplex door een ander op grond van de functiewaarden. Er wordt met de volgende operaties gewerkt *spiegeling, expansie* en *contractie* (zie

[2] voor meer informatie), uitgaande van een simplex met hoekpunten $x_0, x_1, ..., x_n$ die zo genummeerd zijn dat $h(x_0) \geq h(x_1) \geq ... \geq h(x_n)$. We gaan de theorie niet induiken, omdat er in *MATLAB* een functie (`fminsearch`) bestaat die deze methode automatisch toepast.
Er moet wel opgemerk worden dat deze functie slechts een lokaal minimuum vindt.

Omdat onze verzameling van de mogelijke ontwerpvariabelen X begrensd is, vervangen we `fminsearch` door een soortgelijke functie `fminsearchbnd` die wel rekening houdt met de grenzen.

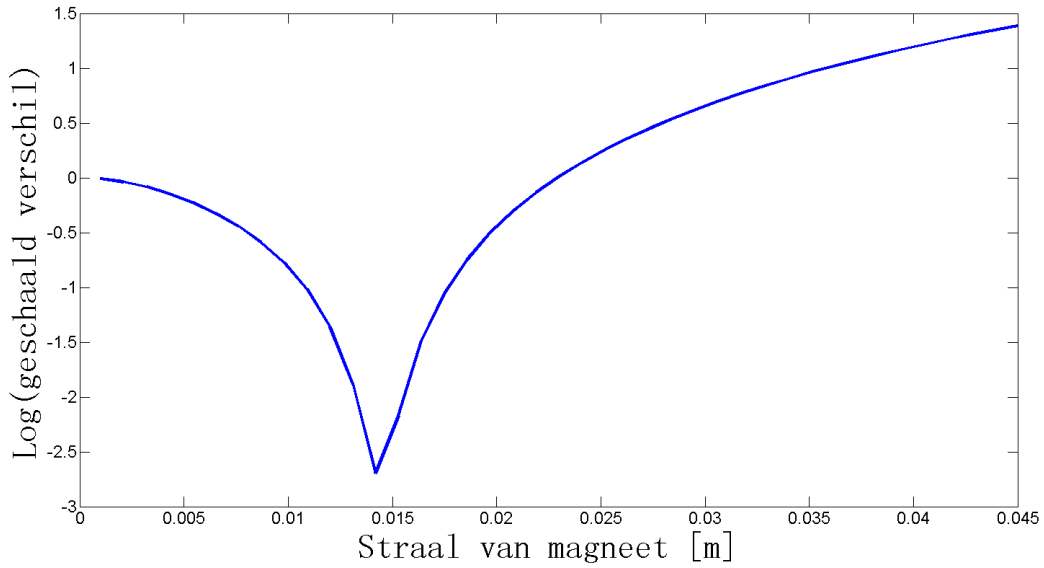
6 Resultaten

6.1 Probleem met één ontwerpvariabele

Bij dit probleem kijken we alleen naar de straal van de magneet x_2. Ons doel is het vinden van de straal die

$$||\mathbf{f}(\mathbf{x}) - \mathbf{y}||$$

met $\mathbf{y} = 12$ N minimaliseert. Het is niet moeilijk om in te zien dat hetzelfde optimum zal het geschaalde verschil $\frac{||\mathbf{f}(\mathbf{x})-\mathbf{y}||}{||\mathbf{y}||}$ minimaliseren. In Figuur 7 ziet u de logaritme van het geschaalde verschil tussen de gewenste oplossing en de output die met het fijne model wordt gevonden (verder *geschaald verschil* genoemd).



Figuur 7: Logaritme van het geschaalde verschil tussen gewenste oplossing en output van het fijne model voor verschillende stralen van de magneet.

Uit Figuur 7 is te zien dat er een uniek minimum bestaat dat bij de straal die ongeveer gelijk is aan 14 mm hoort. Hetzelfde resultaat kan ook gevonden worden door Nelder-Mead simplex toe te passen op het fijne model. De oplossing die gevonden wordt na 41 evaluaties van het fijne model, is onafhankelijk van het beginpunt die we aan `fminsearchbnd` geven. Het verkregen optimum levert 0.0629 als geschaald verschil.

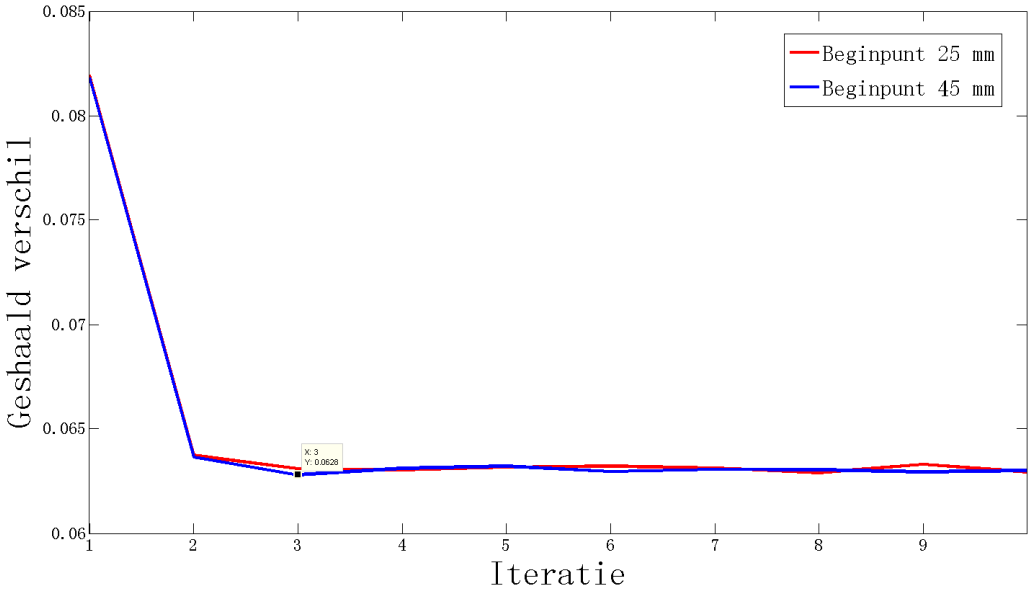
Dezelfde optimaliseringtechniek passen we toe op het grove model met slechts twee steunpunten. Als steunpunten kiezen we punten die op gelijke afstaand liggen van één van de grenzen van $[1 \text{ mm}, 45 \text{ mm}]$ als van het middelpunt van de interval tussen 1 mm en 45 mm. Namelijk, 12 mm en 34 mm. In dit geval is de waarde van het optimum wel afhankelijk van het beginpunt. Het geschaalde verschil dat bij de optima van het grove model hoort, is voor elke beginpunt groter dan 0.080 en kleiner dan 0.085. Dit resultaat proberen wij te verbeteren met behulp van

Manifold Mapping.

De oplossing die met Manifold Mapping bepaald wordt, is afhankelijk van het beginpunt dat we voor de optimalisatie van het grove model gebruiken. Daarom nemen we 10 equidistante punten uit [1 mm, 45 mm] als beginpunten. We maken gebruik van het algoritme dat beschreven is in (4.4) met

$$S_k = \Delta F \Delta G^{\dagger} + (I - UgUg^T).$$

Het aantal iteraties van de Manifold Mapping zetten we gelijk aan 10 en kijken naar de minimale waarde van het geschaalde verschil die tijdens het optimaliseringsproces gevonden wordt. Dit doen we omdat de oplossing die bij iteratie $k+1$ bepaald wordt, kan groter zijn dan de oplossing van de vorige iteratie, k. Dit kunt u zien in Figuur 8. Zo'n gedrag wordt uitgebreid besproken in Hoofdstuk 7.



Figuur 8: Geschaald verschil voor 10 iteraties van de Manifold Mapping voor twee verschillende beginpunten. Het gemarkeerde punt levert 0.0628 als geschaald verschil en hoort bij de derde iteratie met beginpunt 45 mm.

Omdat we het algoritme 10 keer doorlopen, kijken we naar het gemiddelde optimum en het gemiddelde geschaalde verschil. De resultaten kunt u vinden in Tabel 1. Tussen de haakjes staat de standaardafwijking vermeld.

Het aantal iteraties van het fijne model voor de Manifold Mapping volgt uit het feit dat we slechts twee steunpunten gebruiken en dat tijdens elke manifold-mapping iteratie het fijne model alleen één keer geëvalueerd wordt. Het gemiddelde aantal evaluaties van het grove model tijdens gehele optimaliseringproces met Manifold Mapping is ongeveer gelijk aan 330 (met 5.06 als standaardafwijking). Hoewel dit getal redelijk groot lijkt, wordt de oplossingstijd aanzienlijk sneller gevonden. Dit komt doordat de evaluatietijd van het grove model slechts een paar

Model	Optimum (m)	Minimum van geschaald verschil	Aantal evaluaties van $\mathbf{f}$
Fijn	0.01444	0.06295	41
Manifold Mapping	0.01441 ($\pm 1.385 \cdot 10^{-5}$)	0.06288 ($\pm 8.642 \cdot 10^{-5}$)	12

Tabel 1: Resultaten voor 1D-probleem verkregen door optimalisatie van het fijne model en de Manifold Mapping.

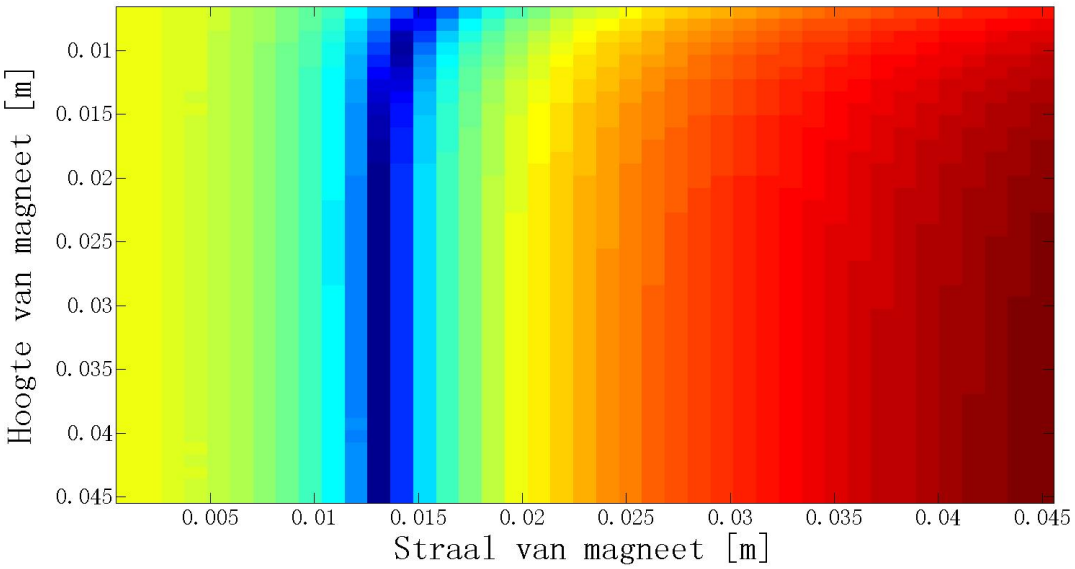
milliseconden is, terwijl de evaluatie van het fijne model ongeveer een minuut duurt.

Het is opvallend dat bij Manifold Mapping het geschaalde verschil in het algemeen iets kleiner is dan 0.06295. (Dit is te verklaren met de nauwkeurigheid waarmee `fminsearchbnd` rekent, die we gelijk nemen aan 10^{-4}.) Het algoritme van Manifold Maping benadert dus zeker het optimum van het fijne model en is aanmerkelijk goedkoper.

6.2 Probleem met twee ontwerpvariabelen

Bij dit probleem zoeken we een combinatie van $x_1 \in [7 \text{ mm}, 45 \text{ mm}]$ en $x_2 \in [1 \text{ mm}, 45 \text{ mm}]$ zodat de Lorentzkracht op de spoel (ongeveer) gelijk wordt aan 12 N voor elke van de negen posities van de spoel.

Het is handig om gelijk op te merken dat voor dit ontwerpprobleem geen unieke optimum bestaat. Er is een verzameling van inputten die dezelfde kracht opleveren en dus ook hetzelfde geschaalde verschil. De situatie voor het fijne model is weergegeven in Figuur 9.



Figuur 9: Logaritme van het geschaalde verschil voor verschillende combinaties van de straal en de hoogte van de magneet. De minima liggen in het donkerblauwe gebied.

We kijken nu naar de 20 iteraties van de Manifold Mapping en vergelijken deze methode met de optimalisatie van het fijne model. Dit doen we weer voor 10 verschillende combinaties van x_1 en x_2. Als steunpunten voor het grove model worden vier punten gebruikt: (16.5 mm, 12 mm),

Model	Minimum van geschaald verschil	Aantal evaluaties van $\mathbf{f}$
Fijn	0.04061	102
	$(\pm 1.587 \cdot 10^{-4})$	(± 8.6)
Manifold Mapping	0.04074	24
	$(\pm 1.614 \cdot 10^{-4})$	n.v.t.

Tabel 2: Resultaten voor 2D-probleem verkregen door optimalisatie van het fijne model en de Manifold Mapping.

(16.5 mm, 34 mm), (35.5 mm, 12 mm) en (35.5 mm, 34 mm). Deze keuze wordt gebaseerd op [5].

In dit geval heeft het geen zin om naar de gemiddelde optima te kijken, omdat er geen unieke minima bestaat. Daarom concentreren we ons voornamelijk op de output die bij het optimum hoort en het aantal evaluaties van het fijne model. De verkregen resultaten staan vermeld in Tabel 2. Voor beide modellen kijken we naar het gemiddelde geschaald verschil en het gemiddelde aantal evaluaties van het fijne model. Tussen de haakjes zijn de standaarduitwijking vermeld.

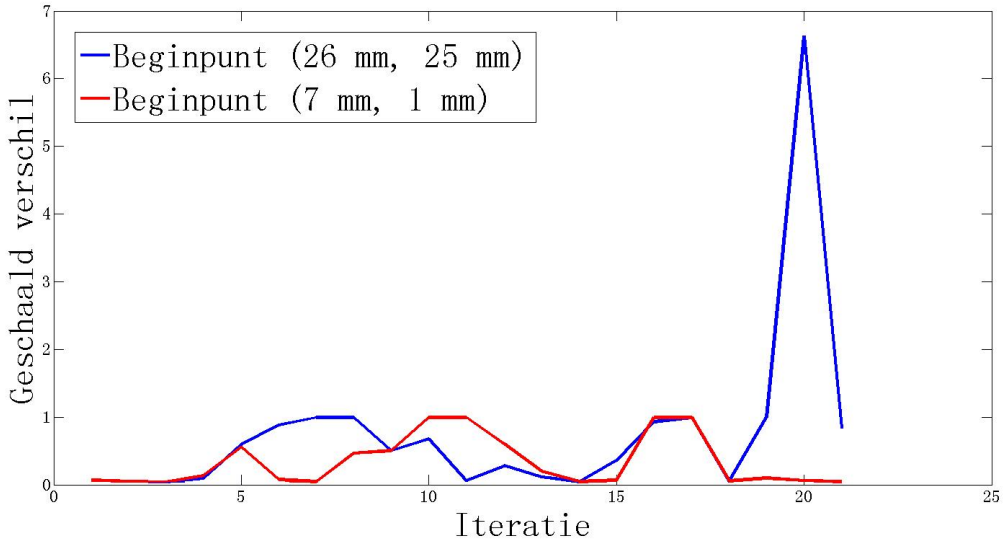
We zien uit de resultaten voor het geschaald verschil voor beide modellen ongeveer hetzelfde is. De Manifold Mapping benadert dus het optimum van het fijne model zeer goed. Daarnaast zijn er aanmerkelijk meer evaluaties van het fijne model nodig voor het vinden van het optimum zonder Manifold Mapping.
Bovendien is het kleinste geschaalde verschil tijdens manifold-mapping optimalisatie voor alle 10 punten binnen de eerste negen iteraties gevonden. Voor elke Manifold Mapping iteratie is het aantal evaluaties van het grove model ongeveer gelijk aan 240. Zelfs als we alle 20 iteraties van de Manifold Mapping meenemen, vinden we minstens vier keer sneller het optimum van het fijne model. Het is dus veel voordeliger om met Manifold Mapping te werken.

Uit de resultaten zien we ook dat het probleem met twee variabelen een oplossing vindt die dichter bij $\mathbf{y}$ ligt, dan de oplossing van het eendimensionaal probleem. Dus naarmate we meer parameters van de voice-coil actuator laten variëren, wordt het gewenste oplossing beter benaderd.

7 Convergentie van Manifold Mapping

7.1 Verschillende aanpakken voor stabilisatie van convergentie

In Hoofdstuk 6.1 hebben we gezien dat de convergentie van de Manifold Mapping niet helemaal stabiel verloopt, maar het verschil tussen twee achtereenvolgende iteraties is gering. Na de eerste iteratie, die in feite geschaald verschil voor $\mathbf{f}(\mathbf{x}_g^*)$ levert, verloopt convergentie van het algoritme redelijk stabiel. Figuur 10 laat zien dat voor het probleem met twee variabelen de convergentie duidelijk minder stabiel is.



Figuur 10: Geschaald verschil voor 21 iteraties van de Manifold Mapping voor 2D-probleem voor twee verschillende beginpunten.

Dit resultaat contrasteert met het feit dat manifold-mapping transformatie zodanig gemaakt is dat de oplossing die deze techniek oplevert altijd naar het optimum van het fijne model convergeert. De oorzaak van zo'n gedrag kan verschillend zijn. De belangrijke factoren, die de convergentie van de Manifold Mapping beïnvloeden, zijn:

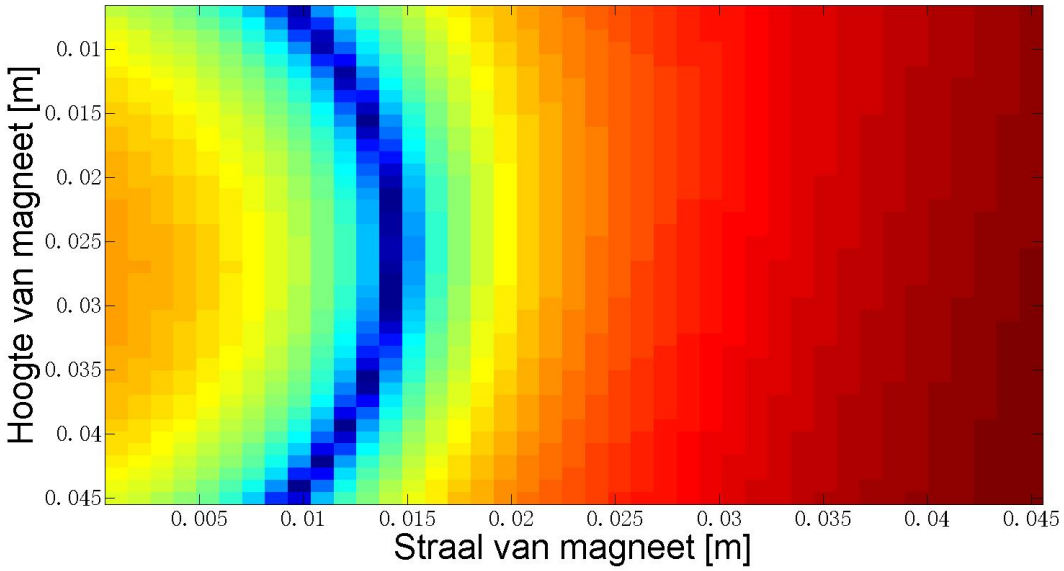
- Output van het fijne en het grove model. (Er is al gezegt dat zowel $\mathbf{f}(\mathbf{x})$ als $\mathbf{g}(\mathbf{x})$ minstens differntieerbaar moeten zijn.)
- Conditie van ΔF en ΔG. (Als het conditiegetal van één van de matrices zeer groot wordt, onstaan er problemen bij de berekeningen van de pseudo-inverse matrices en gaat het algoritme grote stappen nemen.)
- Afstaand tussen $\mathbf{f}(\mathbf{x}_f^*)$ en $\mathbf{y}$.
- Kwaliteit van de benadering die het grove model oplevert.

Het is meestal moeilijk om vast te stellen welke van de genoemde factoren voor instabiliteit zorgt. Dit komt doordat het fijne model normaalgesproken als "black box" wordt gebruikt.

In Hoofdstuk 6.2 hebben we gezien dat het fijne model, dat wij gebruiken, glad is en dat de afstaand tussen de optima van dit model en de gewenste oplossing redelijk klein is. In ons geval is het dus verstandig om of naar het conditiegetallen van ΔF en ΔG te kijken of ervoor te zorgen

dat het grove model een betere benadering oplevert van het fijne model.

In dit hoofdstuk proberen we te analyseren hoe het grove model het beste aangepast kan worden zodanig dat het algoritme stabieler wordt. Dit doen we aan de hand van het tweedimensionaal probleem. In Hoofdstuk 6.2 hebben we gezien hoe $\frac{||\mathbf{f}(\mathbf{x})-\mathbf{y}||}{||\mathbf{y}||}$ eruitziet. Nu bekijken we $\frac{||\mathbf{g}(\mathbf{x})-\mathbf{y}||}{||\mathbf{y}||}$. Het resultaat is weergegeven in Figuur 11.



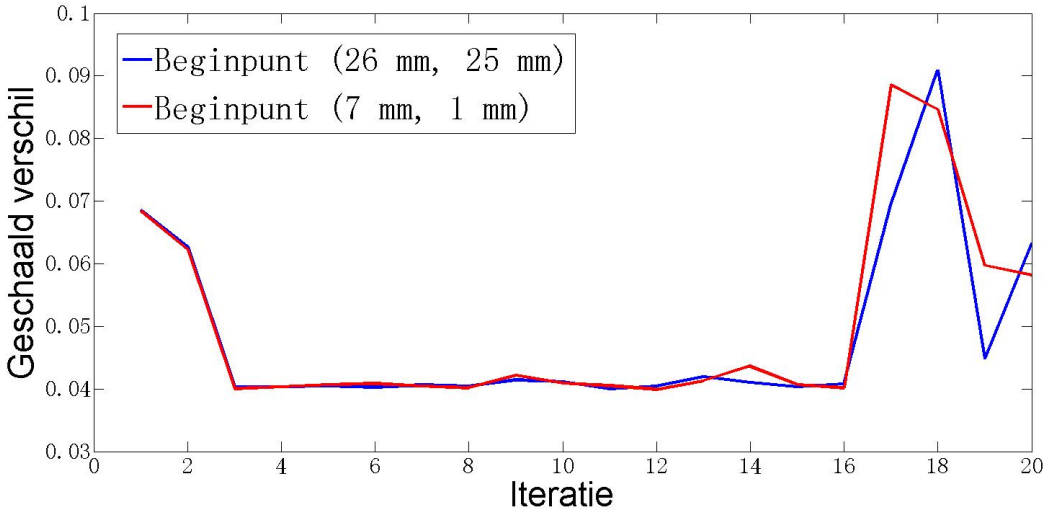
Figuur 11: $\frac{||\mathbf{g}(\mathbf{x})-\mathbf{y}||}{||\mathbf{y}||}$ voor verschillende combinaties van de straal en de hoogte van de magneet. De minima liggen in het donkerblauwe gebied.

Natuurlijk kunnen we oorspronkelijk meer steunpunten gebruiken om de kwaliteit van het grove model te verbeteren. Maar bij elke steunpunt hoort een evaluatie van het fijne model, die we juist willen vermijden. Een andere mogelijkheid is om de punten $\mathbf{x}_k$ met bijbehorende $\mathbf{f}(\mathbf{x}_k)$ die tijdens de manifold-mapping iteratie bepaald worden steeds aan het grove model toe te voegen. We kijken naar de oplossing die met aangepaste grove model gelevert wordt voor dezelfde beginpunten als in Figuur 10. In Figuur 12 zien we een duidelijke verbetering.

We zien dat rond iteratie 18 het geschaalde verschil weer iets groter wordt. In dit geval komt het door de conditie van de matrices ΔF en ΔG. Voor beginoplossingen (26 mm, 25 mm) en (7 mm, 1mm) vindt Manifold Mapping het optimum dat dichter bij $\mathbf{y}$ ligt dan het algoritme zonder aanpassingen van het grove model. Er zijn bovendien minder evaluaties van het grove model nodig. Helaas geldt dit niet voor alle inputten tussen de bovengrens en de ondergrens van mogelijke waarden van x_1 en x_2. In ongeveer vijftig procent van de gevallen die we hebben bekeken werd de matrix Q, waarmee de output van het grove model wordt bepaald, bijna singulier. Hierdoor werd de benadering niet meer nauwkeurig.

Om ervoor te zorgen dat het conditiegetal niet al te groot wordt, gebruiken we een nieuwe aanpak. De punten die tijdens de manifold-mapping iteraties bepaald worden, nemen we nog steeds als steunpunten voor het grove model. Maar nu laten we ook steunpunten weg. Bij iteratie k van de Manifold Mapping wordt het volgende strategie gebruikt:

1. Reken $\mathbf{f}(\mathbf{x}_k)$ uit.



Figuur 12: Geschaald verschil voor 21 iteraties van de Manifold Mapping met aangepaste grove model voor 2D-probleem voor beginpunten (26 mm, 25 mm) en (7 mm, 1mm).

2. Voeg $\mathbf{x}_k$ toe aan de steunpunten van het grove model.
3. Bepaal het geschaalde verschil voor elke steunpunt.
4. Laat het steunpunt met het grootste geschaalde verschil weg.
5. Bereken $\mathbf{g}(\mathbf{x}_k)$ met het vernieuwde grove model. Ga door met de iteratie.

De uitkomsten die met deze aanpak worden gevonden, bespreken we in Hoofdstuk 7.2.

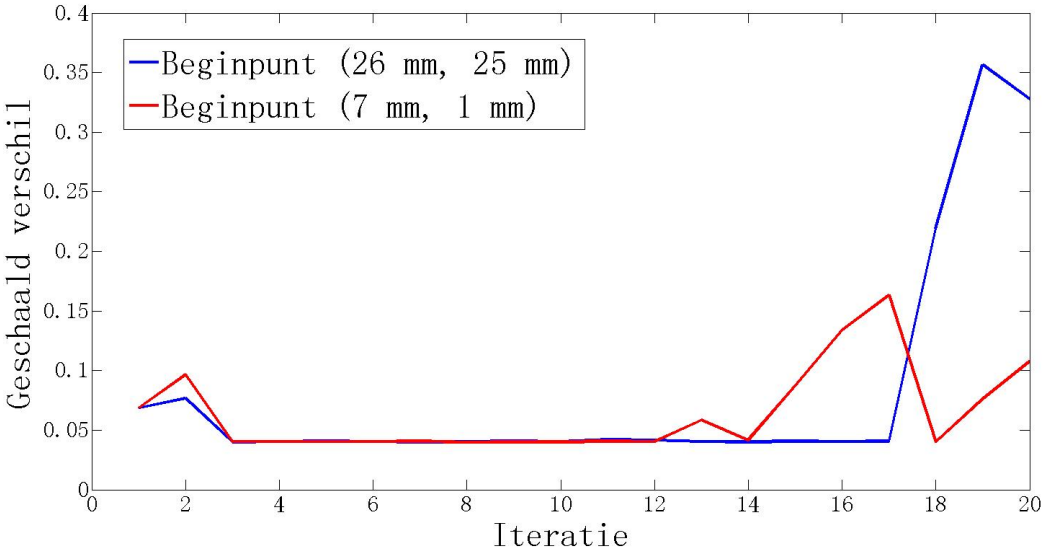
7.2 Vergelijking van resultaten

7.2.1 Probleem met twee onwerpvariabelen

We kijken naar het gestabiliseerde algoritme voor 10 verschillende startpunten. Het aantal iteraties van de Manifold Mapping wordt weer gelijk aan 20 genomen. Slechts voor één van de beginpunten was het geschaalde verschil van een aantal iteraties iets groter dan 6.5. Voor een ander beginpunt was het maximale geschaalde verschil 1.4. Voor alle andere startpunten lag de waarde van het geschaalde verschil altijd onder 1. Een mogelijke oorzaak van deze variabiliteit is de conditie van ΔF of ΔG. Om te voorkomen dat de methode te grote stappen gaat nemen, zou men *Trust-Region* strategie kunnen gebruiken. Voor meer informatie over *Trust-Region* strategie zie [6].

Een voorbeeld van de convergentie van de Manifold Mapping na stabilisatie is te zien in Figuur 13.

Daarnaast hebben we de conditie van Q in de gaten gehouden. Het conditiegetal van Q ten aanzien van de 2-norm werd maximal $2.4 \cdot 10^9$. Dit getal is aanzienlijk groter dan het conditiegetal van het niet aangepaste grove model (114), maar niet groot genoeg om problemen te veroorzaken tijdens het optimaliseringproces.



Figuur 13: Geschaald verschil voor 21 iteraties van de Manifold Mapping na stabilisatie voor 2D-probleem.

Model	Gemiddeld minimum van geschaald verschil	Standaardafwijking van minimum van geschaald verschil
MM voor stabilisatie	0.04074	$1.614 \cdot 10^{-4}$
MM na stabilisatie	0.04038	$7.690 \cdot 10^{-4}$

Tabel 3: Resultaten voor 2D-probleem verkregen door optimalisatie van de Manifold Mapping (MM) voor en na stabilisatie.

Daarnaast vindt gestabiliseerde Manifold Mapping in het algemeen een kleiner minimum voor het geschaalde verschil. Het aantal evaluaties van het grove model tijdens elke iteratie is met factor 2 kleiner geworden, wat de techniek iets goedkoper maakt. In Tabel 3 ziet u de resultaten die we krijgen.

7.2.2 Probleem met vier ontwerpvariabelen

We hebben gezien dat de stabilisatie van de Manifold Mapping voordelig is voor het probleem met twee ontwerpvariabelen. Nu kijken we of het nog steeds zo is als we behalve straal en hoogte van de magneet ook hoogte en straal van de spoel laten variëren. De hoogte van de spoel wordt gekozen uit $X_3 = [1 \text{ mm}, 9 \text{ mm}]$ en de straal uit het interval tussen 1 en 45 mm.

Voor elke ontwerpvariabele wordt het interval tussen de boven- en de ondergrens verdeelt in vier equidistante subintervallen. In alle vier gevallen nemen we slechts twee punten: de bovengrens van de eerste en derde subinterval. Er zijn $2 \times 2 \times 2 \times 2$ mogelijke combinaties van deze punten. Deze combinaties gebruiken we als steunpunten voor het oorspronkelijke grove model. Het aantal iteraties van de Manifold Mapping nemen we gelijk aan 30.

Als startpunten hebben we 10 willekeurige punten genomen. Voor één van die punten werd het conditiegetal te groot en had *MATLAB* moeite met het uitvoeren van het algoritme. Dit punt hebben we vervangen door een ander punt door de hoogte van de spoel te veranderen. (We

Model	Gemiddeld minimum van geschaald verschil	Standaardafwijking van minimum van geschaald verschil
MM voor stabilisatie	0.01837	0.0028
MM na stabilisatie	0.02157	0.0059

Tabel 4: Resultaten voor 4D-probleem verkregen door optimalisatie van de Manifold Mapping (MM) voor en na stabilisatie.

hebben dus gekeken naar 10 punten voor de Manifold Mapping zonder stabilisatie en 11 punten voor de gestabiliseerde techniek.) Het maximale conditiegetal van andere 9 punten en van het nieuwe punt waren van de orde $10^{13}, 10^{15}$ en 10^{17}. Dit betekent de matrix Q bijna singulier wordt tijdens het gestabiliseerde manifold-mapping algoritme. Maar de afstaand tussen de output van de gevonden optima en de gewenste oplossing is met stabilisatie kleiner dan de afstaand die wij zonder stabilisatie vinden. Dit is te zien in Tabel 4.

De singulariteit van matrix Q is te verklaren door het feit dat we redelijk veel steunpunten gebruiken en doordat de punten die wij toevoegen heel dicht bij elkaar kunnen liggen.

We zien dat naarmate we meer ontwerpparameters nemen, komt de Lorentzkracht op de spoel dichter bij 12 N. Dit geldt voor beide technieken die wij vergelijken.
Daarnaast zijn deze methoden aanmerkelijk sneller dan directe optimalisatie van het fijne model die ongeveer 800 evaluaties nodig heeft. Met Manifold Mapping evalueren we het fijne model slechts 46 keer. Bovendien is het minimum van geschaald verschil van manifold-mapping optimalisatie vermoedelijk kleiner, dan de uitkomst (0.02542) die wij bij directe optimalisatie vinden. We kunnen dit niet helemaal zeker zeggen, omdat het fijne model vanwege zijn kosten slechts één keer direct geoptimaliseerd werd.

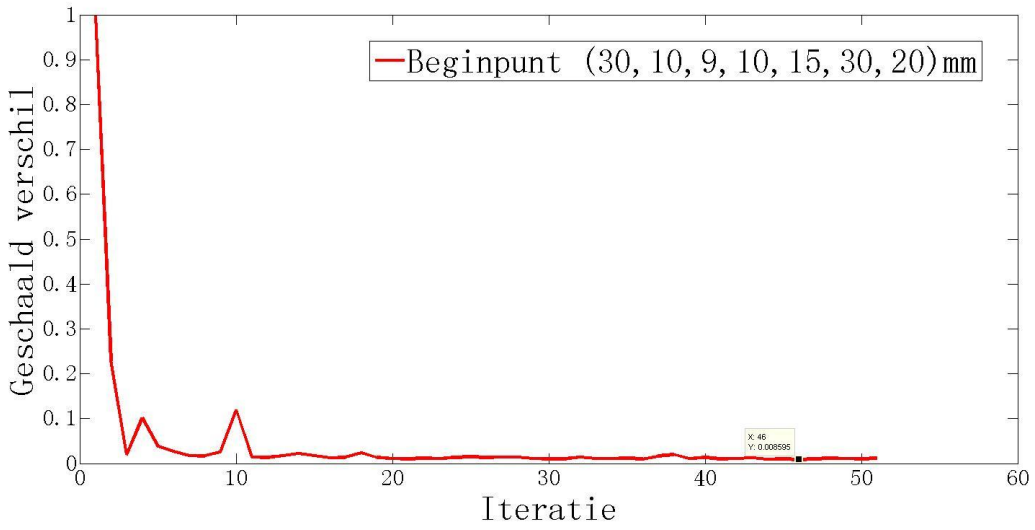
7.2.3 Probleem met zeven otnwerpvariabelen

Bij dit probleem laten we variëren niet alleen de afmetingen van de magneet en de spoel, maar ook de afmetingen van de ferromagnetische kern. Dus eigenlijk kijken we naar de optimale constructie van de gehele voice-coil actuator.

De steunpunten worden net zo gekozen als in Hoofdstuk 7.2.2. In totaal hebben we dus 2^7 steunpunten. We laten Manifold Mapping 50 iteraties doorlopen. Dus in totaal wordt het fijne model 178 keer doorgerekend.

Helaas vindt de Manifold Mapping zonder stabilisatie geen oplossing die beter is dan het optimum van het probleem met 4 variabelen, terwijl de optimalisatie in dit geval bijna 4 keer langer is. Het minimale geschaalde verschil dat met deze techniek bereikt wordt, is ongeveer gelijk aan 0.04 (wat zeer op de oplossing van het tweedimensionale probleem lijkt).

De stabilisatie van de Manifold Mapping heeft in dit geval niet altijd positieve invloed. Voor de meeste waarden van ontwerpvariabelen loopt het algoritme vast, omdat Q (bijna) singulier wordt. Hoewel er toch wel punten bestaan waarvoor de stabilisatie voor een oplossing zorgt met het geschaalde verschil van 0.0098. Zo'n geval is weergegeven in Figuur 14.



Figuur 14: Geschaald verschil voor 50 iteraties van de Manifold Mapping na stabilisatie voor 7D-probleem.

8 Conclusie en discussie

In dit verslag zijn we bezig geweest met het begrijpen van de manifold-mapping techniek en het toepassen van dit algoritme op het ontwerpprobleem van een voice-coil actuator. In Hoofdstuk 4.2 hebben we opgemerkt dat het effect van deze techniek eigenlijk overeenkomt met translatie en rotatie van de output van het grove model zodanig dat dit model hetzelfde optimum oplevert als het fijne model. Vervolgens hebben we gekeken hoe zo'n transformatie ontstaat uit de linearisering van fijne en grove modellen. Uit de constructie van manifold-mapping transformatie kan men direct zien dat het algoritme altijd naar het optimum van het fijne model convergeert.

Het doel van de Manifold Mapping is het reduceren van de rekentijd (en dus ook de kosten) van optimalisatie van de problemen die een nauwkeurige oplossing vereisen. Als voorbeeld van zo'n probleem nemen we het samenstel van een voice-coil actuator. Een voice-coil actuator is een toestel die elektrische signalen in trillingen omzet. Het bestaat uit 3 onderdelen: een spoel, een permanente magneet en een kern waarin de magneet ingebouwd is. We willen de afmetingen van de voice-coil actuator weten, waarbij een magnetische kracht van 12 N op de spoel uitgeoefend wordt.

In Hoofdstuk 5.2 worden de natuurkundige processen geanalyseerd die van belang zijn voor de constructie van het fijne model. Dit wordt gedaan met behulp van Maxwell vergelijkingen. Als resultaat krijgen we een ingewikkelde partiële differentiaalvergelijking die analytisch niet oplosbaar is. Daarom maken we gebruik van een softwarepacket dat een numerieke oplossing vindt van ons probleem. Dit model levert een nauwkeurig resultaat op, maar de rekentijd wordt zeer lang. In Hoofdstuk 5.3 maken we een model dat ons fijn model kan snel benaderen. De Manifold Mapping wordt gebaseerd op deze twee modellen.

In Hoofdstuk 6.1 worden de resultaten besproken die wij krijgen voor eendimensionaal probleem met behulp van de manifold-mapping techniek. Bij dit probleem laten we alleen de straal van de magneet variëren. Wij vergelijken de oplossing die manifold-mapping optimalisatie oplevert

met de oplossing die bepaald wordt met behulp van directe optimalisatie van het fijne model. Het blijkt dat Manifold Mapping niet alleen hetzelfde resultaat vindt, maar ook de zoektocht naar het optimum flink versneld. De resultaten voor 2D-probleem worden besproken in 6.1. In dit geval bestaat er een gehele verzameling van de inputvariabelen waarbij de kracht op de spoel bijna gelijk wordt aan 12 N. Ook hier concluderen we dat het gebruik van Manifold Mapping zeer voordelig is. Bovendien ligt de kracht die bij het optimum hoort dichter bij 12 N dan de minimale kracht die wij voor 1D-probleem vinden.

Het is opvallend dat de convergentie van de manifold-mapping voor het tweedimensionale probleem niet stabiel verloopt. Dit verschijnsel wordt bestudeert in Hoofdstuk 7. Eerst bespreken we allerlei factoren die de convergentie kunnen beïnvloeden. Aan de hand van de gegevens over het fijne model concluderen we dat de instabiliteit twee mogelijke oorzaken kan hebben. Namelijk, de conditie van de matrices ΔF en ΔG (zie Hoofdstuk 4.2 voor de definitie) of de kwaliteit van de benadering die het grove model oplevert. We besluiten naar het grove model te gaan kijken. Er bestaan meerdere aanpakken om de kwaliteit van dit model te verbeteren. Bijvoorbeeld is het mogelijk om de output van het fijne model, die tijdens manifold-mapping iteratie gevonden wordt, toe te voegen aan de punten waarmee het grove model rekent. Deze aanpak maakt de convergentie in 2D-geval aanmerkelijk stabieler. Het feit dat de methode niet helemaal stabiel wordt, kan verklaard worden door conditie van ΔF en ΔG. Als deze matrices te veel op de singuliere matrices gaan lijken, onstaan er problemen met het bepalen van pseudo-inverse matrices en kan de Manifold Mapping te grote stappen gaan nemen. Dit kan men voorkomen met behulp van *Trust-Region* strategie, die beschreven is in [6].
Er moet opgemerkt worden dat de Manifold Mapping met stabilisatie een betere oplossing van het probleem vindt en sneller is.

Vervolgens kijken we ook naar vierdimensionale versie van het voice-coil actuator probleem. In de meeste gevallen verloopt de stabilisatie van de Manifold Mapping heel goed. (Maar soms ontstaan er problemen met het inverteren van de matrix Q (zie 5.3 voor de definitie), waarmee we de output van het grove model bepalen.) De 4D-variant vindt op zijn beurt een kracht op de spoel die dichter bij 12 N zit, dan de minimale kracht van 2D-probleem. Hoewel de rekentijd langer wordt.

Als laatste werd het geval bekeken met 7 ontwerpvariabelen. De stabilisatie van het algoritme voor dit probleem verloopt bij de meeste gevallen niet soepel. Dit is weer het gevolg van de conditie van Q. Wanneer Q toch niet-singulier blijft, verloopt de convergentie stabiel en de wordt het verschil tussen gevonden en gewenste oplossing kleiner dan in 4D-probleem.

Hieruit wordt geconcludeerd dat voor een-, twee- en vierdimensionale versies van het voice-coil actuator probleem zorgt de manifold-mapping techniek voor een nauwkeurige oplossing die binnen relatief korte tijd bepaald wordt. In alle drie gevallen convergeert het algoritme naar het optimum van het fijne model. De optimalisatie met behulp van de Manifold Mapping verloopt aanzienlijk sneller dan de directe optimalisatie van het fijne model.
Voor het zevendimensionale geval levert Manifold Mapping zonder stabilisatie geen verbetering van de 2D-oplossing. Als het algoritme gestabiliseerd wordt, ontstaan er meestal problemen bij de matrices van het grove model. Deze problemen kunnen waarschijnlijk opgelost worden door een andere keuze van de oorspronkelijke steunpunten.

9 Bijlage

9.1 Singuliere Waarden Ontbinding

De Singuliere Waarden Ontbinding wordt in dit verslag gebruikt om de pseudo-inverse van $J_g(\mathbf{x}_f^*)$ te bepalen. In het algemeen werkt deze techniek voor elke singuliere matrix met elementen uit $\mathbb{R}$ en $\mathbb{C}$.
Met behulp van Singuliere Waarden Ontbinding kan $J_g(\mathbf{x}_f^*)$ geschreven worden als $U_g \Sigma_g V_g^T$. Hier is U_g een $m \times m$-matrix met het volgende eigenschap:

$$U_g^T U_g = I.$$

Σ_g is een $m \times n$-matrix met diagonaalelementen groter of gelijk aan nul en alle andere elementen gelijk aan nul. De diagonaalelementen van Σ_g worden de singuliere waarden van $J_g(\mathbf{x}_f^*)$ genoemd. Verder is V_g een $n \times n$-matrix zodanig dat $V_g^T V_g = I$.

De Singuliere Waarden Ontbinding kan als volgt uitgevoerd worden:

1. - Om U_g te vinden, bepaal eerst de eigenwaarden van de $m \times m$-matrix $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*)$. Zorg ervoor dat de eigenwaarden niet-negatief worden. Vind de bijbehorende eigenvectoren.
   - Gebruik de gevonden eigenvectoren om een nieuwe matrix te construeren. De eerste kolom van de nieuwe matrix wordt gevormd door de eigenvector die bij de grootste eigenwaarde hoort, de tweede kolom bestaat uit de eigenvector van de een na grootste eigenwaarde enzovoort. (De volgorde is niet van belang als er dezelfde eigenwaarde meerde keren voorkomt.)
   - De gevonden matrix moet vervolgens omgezet worden naar een orthogonale matrix. Dit levert U_g op.
2. Het vinden van V_g gaat op de manier die beschreven is bij stap 1, maar je gebruikt matrix $J_g^T(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*)$ in plaats van $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*)$.

   Merk op: als $\lambda \neq 0$ een eigenwaarde van $J_g^T(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*)$ is, dan is λ ook een eigenwaarde van $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*)$. Andersom geldt dit ook.
3. - De diagonaalelementen van Σ_g bepaal je door wortel te trekken uit de positieve eigenwaarden van $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*)$ (of $J_g^T(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*)$). Het element in de eerste rij en de eerste kolom van Σ_g moet groter of gelijk zijn aan het element in de tweede rij en de tweede kolom enzovoort.
   - Breid de matrix uit met nullen totdat je m kolommen en n rijen krijgt. Zo vind je Σ_g.
4. Nu is $J_g(\mathbf{x}_f^*)$ gelijk aan $U_g \Sigma_g V_g^T$.

Voor verduidelijking bekijken we de Singuliere Waarden Ontbinding van

$$J_g(\mathbf{x}_f^*) = \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix}.$$

1. $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*) = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}$.
   Aan $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*)\mathbf{x} = \lambda\mathbf{x}$ met $\mathbf{x} \neq 0$ voldoen $\lambda = 0$, $\lambda = 10$ en $\lambda = 12$. De bijbehorende eigenvectoren zijn respectievelijk

   $$[1, 2, -5], [2, -1, 0] \text{ en } [1, 2, 1].$$

   Hieruit volgt dat

   $$\begin{bmatrix} 1 & 2 & 1 \\ 2 & -1 & 2 \\ 1 & 0 & 5 \end{bmatrix}$$

   moet omgezet worden naar de orthogonale matrix om U_g te verkrijgen. Dit kan met behulp van Gram-Schmidtmethode. Na wat rekenwerk vinden we dat

   $$U_g = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix}.$$

2. Hetzelfde doen we voor $J_g^T(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*)$ en vinden

   $$V_g = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}.$$

3. De eigewaarden van $J_g(\mathbf{x}_f^*)J_g^T(\mathbf{x}_f^*)$ zijn 0, 10 en 12. Bij stap 2 vonden we dat 10 en 12 de eigenwaarden zijn van $J_g^T(\mathbf{x}_f^*)J_g(\mathbf{x}_f^*)$. Dit levert ons twee positieve eigenwaarden 10 en 12. Dus

   $$\Sigma_g = \begin{bmatrix} \sqrt{12} & 0 \\ 0 & \sqrt{10} \\ 0 & 0 \end{bmatrix}.$$

4. Hieruit volgt dat

   $$J_g(\mathbf{x}_f^*) = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{30}} \\ \frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{30}} \\ \frac{1}{\sqrt{6}} & 0 & \frac{-5}{\sqrt{30}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 \\ 0 & \sqrt{10} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}.$$

De pseudo-inverse matrix van $J_g(\mathbf{x}_f^*) = U_g\Sigma_g V_g^T$ is gelijk aan

$$J_g^\dagger = (U_g\Sigma_g V_g^T)^\dagger = (V_g^T)^\dagger\Sigma_g^\dagger U_g^\dagger = (V_g^T)^{-1}\Sigma_g^\dagger U_g^{-1} = (V_g^{-1})^{-1}\Sigma_g^\dagger U_g^T = V_g\Sigma_g^\dagger U_g^T.$$

De pseudo-inverse van Σ_g is zeer makkelijk te bepalen. Namelijk, Σ_g wordt getransponeerd en elke diagonaalelement $\Sigma_g^\dagger(i,i)$ wordt gelijk aan $\frac{1}{\Sigma_g(i,i)}$ met $\Sigma_g(i,i) \neq 0$. De nullen blijven onveranderd.

9.2 *MATLAB*-code

9.2.1 1D-model

```
function Fz_coil= fm(x2,D)
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.4 (COMSOL 3.4.0.248, $Date: 2007/10/10 16:07:51 $)

flclear fem
% Constants
fem.const = {'murFe','1e3', ...
  'murmagn','1.09', ...
  'Mpre','978802', ...
  'fill_factor','.7', ...
  'curr_dens','-2.15e7', ...
  'coil_curr_dens','fill_factor*curr_dens', ...
  'murmax','5000', ...
  'murmin','200', ...
  'C','25'};

%Set design parameter
design_params(1) = 1e-3;          %..core base height..
design_params(2) = 1.472e-3;      %..core tip heigt..
design_params(3) = 1e-3;          %..core leg width..
design_params(4) = 8.62e-3;       %..pm height..
design_params(5) = x2;            %..pm radius..
design_params(6) = .9e-3;         %..ag left width..
design_params(7) = .725e-3;       %..ag right width..
design_params(8) = 7.7463e-3;     %..coil height..
design_params(9) = 2.376e-3;      %..coil width..

%Set coil displacement
cl_displ  =  D;

%Define geometry
cr_base_h = design_params(1);
cr_tip_h  = design_params(2);
cr_leg_w  = design_params(3);
pm_h      = design_params(4);
pm_w      = design_params(5);
ag_left_w = design_params(6);
ag_rght_w = design_params(7);
cl_h      = design_params(8);
cl_w      = design_params(9);
ag_h      = pm_h + cr_tip_h;
ag_w      = ag_left_w + ag_rght_w + cl_w;
cr_h      = cr_base_h + pm_h + cr_tip_h;
cr_w      = pm_w + ag_w + cr_leg_w;
cr_no_gap = rect2(0, cr_w, 0, cr_h);
```

```
ag        = rect2(pm_w, pm_w + ag_w, cr_base_h, cr_base_h + ag_h);
cr        = cr_no_gap - ag;
pm        = rect2(0, pm_w, cr_base_h, cr_base_h + pm_h);
cl        = rect2(pm_w + ag_left_w,  pm_w + ag_left_w + cl_w, ...
    cr_h-cl_h, cr_h);
cl        = move(cl,0,cl_displ);
factor    = 5;
as        = rect2(0, factor*cr_w, -factor*cr_h, factor*cr_h);
my_geom   = cr + pm + cl + as;
my_geom   = cl + as;

clear s
s.objs={cr,pm,cl,as};
s.name={'Core','Permanent_Magnet','Coil','Air_Surrounding'};
s.tags={'Core','Permanent_Magnet','Coil','Air_Surrounding'};

fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% Initialize mesh
fem.mesh=meshinit(fem,'hauto',5);

% Application mode 1
clear appl
appl.mode.class = 'AzimuthalCurrents';
appl.mode.type = 'axi';
clear bnd
bnd.type = {'A0','cont','ax'};
bnd.ind = {[2,11,22],[4,6,8,10,12:13:21],[1,3,5,7,9]};
appl.bnd = bnd;
clear equ
equ.magconstrel = {'mur','mur','M','mur'};
equ.M = {{0;0},{0;0},{0;'Mpre'},{0;0}};
equ.mur = {1,'murFe','murmagn',1};
%{1,'(murmax/(1+C*normB^2)+murmin)','murmagn',1};
equ.Jephi = {0,0,0,'coil_curr_dens'};
equ.ind = [1,2,3,2,4];
appl.equ.init= '2e-3';
appl.equ = equ;
appl.shape=1;
fem.appl = appl;
fem.sdim = {'r','z'};
fem.frame = {'ref'};
fem.border = 1;
fem.outform = 'general';

% ODE Settings
clear ode
clear units;
```

```
units.basesystem = 'SI';
ode.units = units;
fem.ode=ode;


% Multiphysics
fem=multiphysics(fem,'outform', 'general');

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femlin(fem);

% % Save current fem structure for restart purposes
% fem0=fem;
%
% %Lorentz force
 Fz_coil_lin = postint(fem,'-2*pi*r*Br*Jephi',...
     'edim', 2, 'dl', 5);


 feminit = fem;



%%Non-linear
clear appl
appl.mode.class = 'AzimuthalCurrents';
appl.mode.type = 'axi';
clear bnd
bnd.type = {'A0','cont','ax'};
bnd.ind = {[2,11,22],[4,6,8,10,12:13:21],[1,3,5,7,9]};
appl.bnd = bnd;
clear equ
equ.magconstrel = {'mur','mur','M','mur'};
equ.M = {{0;0},{0;0},{0;'Mpre'},{0;0}};
equ.mur = {1,'(murmax/(1+C*normB^2)+murmin)','murmagn',1};
%{1,'(murmax/(1+C*normB^2)+murmin)','murmagn',1};
equ.Jephi = {0,0,0,'coil_curr_dens'};
equ.ind = [1,2,3,2,4];
appl.equ.init= '2e-3';
appl.equ = equ;
appl.shape=1;
fem.appl = appl;
fem.sdim = {'r','z'};
fem.frame = {'ref'};
fem.border = 1;
fem.outform = 'general';
```

```
% Multiphysics
fem=multiphysics(fem,'outform', 'general');

% Extend mesh
fem.xmesh=meshextend(fem);

%Solve problem
Maxiter = 200;
    fem.sol = femnlin(fem, 'Maxiter', Maxiter',...
                      'Pname','scale_factor',...
                      'Plist',[1 2 5 10],...
                      'init',feminit.sol);

 %Lorentz force
  Fz_coil = postint(fem,'-2*pi*r*Br*Jephi',...
      'edim', 2, 'dl', 5);

################################################

function[antwoord] = manifold1d(punt)
format long

y=12*ones(9,1);
S=cell(0);
dc=cell(1);
df=cell(1);

vecx=[12 34]*1e-3;
for i=1:length(vecx)
    F(i,:)=lookup_fm_1d(vecx(i));
end

%Ik neem aan dat zowel p_streep als S0 gelijk zijn aan I.
%x0 bepalen.
k=1;
[x(k),fval(k),exitflag(k),output(:,k)]=fminsearchbnd(@(x) ...
    compute_cm_1d(x,y,vecx,F), punt, 1e-3, 45e-3,...
    optimset('TolX',1e-5,'TolFun',1e-5));
number_iter_grof_model(k)=output(:,k).funcCount;

f(:,k)=lookup_fm_1d(x(k));
c(:,k)=lookup_cm_1d(x(k),vecx,F);
resp(:,k)=f(:,k);

%x1 bepalen.
k=k+1;
[x(k),fval(k),exitflag(k),output(:,k)]=fminsearchbnd(@(x)...
```

```
    opt1d_manifold_1d(x,y,c(:,k-1),f(:,k-1),...
    zeros(9,1),zeros(9,1),zeros(9),vecx,F), x(k-1),1e-3, 45e-3,...
    optimset('TolX',1e-5,'TolFun',1e-5));
number_iter_grof_model(k)=output(:,k).funcCount;

f(:,k)=lookup_fm_1d(x(k));
c(:,k)=lookup_cm_1d(x(k),vecx,F);df{1,1}(:,1)=f(:,k-1)-f(:,k);
dc{1,1}(:,1)=c(:,k-1)-c(:,k);
[Uc,Sc,Vc]=svd(dc{1,1});
[Uf,Sf,Vf]=svd(df{1,1}); %deze mag weg!
S{1,length(S)+1}=eye(9);
resp(:,k)=f(:,k);


%Optimale waarde van x zoeken.
while k<=9
    k=k+1;
    [x(k),fval(k),exitflag(k),output(:,k)]=fminsearchbnd(@(x)...
        opt1d_manifold_1d(x,y,c(:,k-1),f(:,k-1),...
        dc{1,k-2},df{1,k-2},Uc,vecx,F), x(k-1),1e-3, 45e-3,...
        optimset('TolX',1e-5,'TolFun',1e-5));
    number_iter_grof_model(k)=output(:,k).funcCount;

    f(:,k)=lookup_fm_1d(x(k));
    c(:,k)=lookup_cm_1d(x(k),vecx,F);
    df{1,length(df)+1}(:,1)=f(:,k-1)-f(:,k);
    dc{1,length(dc)+1}(:,1)=c(:,k-1)-c(:,k);
    [Uc,Sc,Vc]=svd(dc{1,length(dc)});
    [Uf,Sf,Vf]=svd(df{1,length(df)}); %deze mag ook weg, omdat Uf wordt niet gebruikt!
    S{1,length(S)+1}=df{1,length(df)}*pinv(dc{1,length(dc)})+...
        (eye(9)-Uf*Uf')*(eye(9)-Uc*Uc');
    resp(:,k)=f(:,k);
end

for l=1:k
    NN(l)=norm(resp(:,l)-y)/norm(y);
end
% plot(1:k,NN)
% xlabel('Iteratie');
% ylabel('Verschil verkregen en gewenste output');
% hold on
[value_min_NN, it_min_NN]=min(NN)
antwoord=x(it_min_NN)
aantal_iter_MM=length(x)
total_number_iter_fijn_model=aantal_iter_MM+size(vecx,2)
total_number_iter_grof_model=sum(number_iter_grof_model)

##############################################################
```

```
function fm_resp_vec = lookup_fm_1d(pm_w)

  %..Looks up the fine model response for x on input

  cl_displ = [0:0.5:4]*1e-3;
  for i=1:length(cl_displ)
         fm_resp_vec(i,1) = fm(pm_w, cl_displ(i));
  end

##############################################################

function[c]=lookup_cm_1d(x,vecx,F)

K=50;
N=length(vecx);

for i=1:N
    vecn(i)=norm(vecx(i));
end
h=sum(vecn)/(K*N);

for j=1:N
    q(j)=sqrt((norm(x-vecx(j)))^2+h);
end

for k=1:N
    for l=1:N
            Q(k,l)=sqrt((norm(vecx(k)-vecx(l)))^2+h);
    end
end

a=Q\F;
[m,n]=size(a);

for t=1:n
    c(t,1)=q*a(:,t);
end

##############################################################

function cm_costf = compute_cm_1d(pm_w, desired_resp,vecx,F)

  %..Compute the fine model cost function for x on input

  cm_resp_vec = lookup_cm_1d(pm_w,vecx,F);

  cm_costf = norm(cm_resp_vec - desired_resp) / ...
             norm(desired_resp);
```

```
#################################################################

function[Opt]=opt1d_manifold_1d(x,y,c0,f0,dc,df,Uc,vecx,F)

c(:,1)=lookup_cm_1d(x,vecx,F);
Opt=norm((c(:,1)-c0)+(dc*pinv(df)+eye(9)-Uc*Uc')*(f0-y))/norm(y);
```

9.2.2 2D-model met stabilisatie

2D-model zonder stabilisatie ziet er analoog uit.

```
function[value_min_NN] = manifold2d_uitbreiding(px1,px2)
format long

y=12*ones(9,1);
S=cell(0);
OG=[7,1]*1e-3;
BG=[45,45]*1e-3;

X_pm_h=[16.5, 35.5]*1e-3;
X_pm_w=[12, 34]*1e-3;
matx=allcomb(X_pm_h,X_pm_w)';

for i=1:size(matx,2)
    F(i,:)=lookup_fm_2d(matx(1,i),matx(2,i));
end
F


%Ingredienten voor het grove model
K=50;
N=size(matx,2);
for r=1:N
    matn(r)=norm(matx(:,r));
end
h=sum(matn)/(K*N);
for p=1:N
    for t=1:N
            Q(p,t)=sqrt((norm(matx(:,p)-matx(:,t)))^2+h);
    end
end
Cond_Q(1)=cond(Q);


%Ik neem aan dat zowel p_streep als S0 gelijk zijn aan I.
%Vector x0 bepalen.
k=1;
```

```
[x(:,k),fval(k),exitflag(k),output(:,k)]=fminsearchbnd(@(x)...
    compute_cm_2d_uit(x,y,matx,F,h,Q),[px1;px2],OG,BG,...
    optimset('TolX',1e-4,'TolFun',1e-4,'MaxFunEvals',5000));
number_iter_grof_model(k)=output(:,k).funcCount;

f(:,k)=lookup_fm_2d(x(1,k), x(2,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k);
r_F=size(F,1);
F(r_F+1,:)=f(:,k)';

%Een rij van F weglaten
k_matx=size(matx,2);
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[];

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[];

%Ingredienten voor het grove model
K=50;
N=size(matx,2);
for r=1:N
    matn(r)=norm(matx(:,r));
end
h=sum(matn)/(K*N);
for p=1:N
    for t=1:N
            Q(p,t)=sqrt((norm(matx(:,p)-matx(:,t)))^2+h);
    end
end
Cond_Q(k+1)=cond(Q);


c(:,k)=lookup_cm_2d_uit(x(:,k),matx,F,h,Q);
resp(:,k)=f(:,k);


%x1 bepalen.
k=k+1;
[x(:,k),fval(k),exitflag(k),output(:,k)]=fminsearchbnd(@(x) ...
    opt2d_manifold_uit(x,y,c(:,k-1),f(:,k-1),...
    zeros(9,1),zeros(9,1),zeros(9),matx,F,h,Q), x(:,k-1),OG,BG,...
```

```
    optimset('TolX',1e-4,'TolFun',1e-4,'MaxFunEvals',5000));
number_iter_grof_model(k)=output(:,k).funcCount;

f(:,k)=lookup_fm_2d(x(1,k), x(2,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k);
r_F=size(F,1);
F(r_F+1,:)=f(:,k)';

%Een rij van F weglaten
k_matx=size(matx,2);
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[];

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[];

%ingredienten voor het grove model
N=size(matx,2);
for r=1:N
    matn(r)=norm(matx(:,r));
end
h=sum(matn)/(K*N);
for p=1:N
    for t=1:N
            Q(p,t)=sqrt((norm(matx(:,p)-matx(:,t)))^2+h);
    end
end
Cond_Q(k+1)=cond(Q);


c(:,k)=lookup_cm_2d_uit(x(:,k),matx,F,h,Q);
dc(:,1)=c(:,k-1)-c(:,k);
df(:,1)=f(:,k-1)-f(:,k);
[Uc,Sc,Vc]=svd(dc);
S{1,length(S)+1}=eye(9);
resp(:,k)=f(:,k);


% Optimale waarde van x zoeken.
while k<=19
    k=k+1;
    [x(:,k),fval(k),exitflag(k),output(:,k)]=fminsearchbnd(@(x)...
```

```
    opt2d_manifold_uit(x,y,c(:,k-1),f(:,k-1),...
    dc,df,Uc,matx,F,h,Q), x(:,k-1),OG,BG,...
    optimset('TolX',1e-4,'TolFun',1e-4,'MaxFunEvals',5000));
number_iter_grof_model(k)=output(:,k).funcCount;

f(:,k)=lookup_fm_2d(x(1,k), x(2,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k);
r_F=size(F,1);
F(r_F+1,:)=f(:,k)';

%Een rij van F weglaten
k_matx=size(matx,2);
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[];

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[];

%ingredienten voor het grove model
N=size(matx,2);
for r=1:N
    matn(r)=norm(matx(:,r));
end
h=sum(matn)/(K*N);
for p=1:N
    for t=1:N
        Q(p,t)=sqrt((norm(matx(:,p)-matx(:,t)))^2+h);
    end
end
Cond_Q(k+1)=cond(Q);



c(:,k)=lookup_cm_2d_uit(x(:,k),matx,F,h,Q);
clear dc
clear df
for i=1:min(2,k-1)
    dc(:,i)=c(:,k-i)-c(:,k);
    df(:,i)=f(:,k-i)-f(:,k);
end
[Uc,Sc,Vc]=svd(dc);
[Uf,Sf,Vf]=svd(df);
```

```
    S{1,length(S)+1}=df*pinv(dc)+(eye(9)-...
        Uf*Uf')*(eye(9)-Uc*Uc');
    resp(:,k)=f(:,k);
end


for l=1:k
    NN(l)=norm(resp(:,l)-y)/norm(y);
end
% plot(1:k,NN,'r')
% xlabel('Iteratie');
% ylabel('Geschaald verschil');
% hold on
[value_max_cond,it_max_cond]=max(Cond_Q)
[value_min_NN, it_min_NN]=min(NN)
[value_max_NN, it_max_NN]=max(NN)
antwoord(:,1)=x(:,it_min_NN)
aantal_iter_MM=length(x)
total_number_iter_fijn_model=aantal_iter_MM+size(matx,2)
total_number_iter_grof_model=sum(number_iter_grof_model)

############################################################

function fm_resp_vec = lookup_fm_2d(pm_h,pm_w)

  %..Looks up the fine model response for x on input

  cl_displ = [0:0.5:4]*0.001;
  for i=1:length(cl_displ)
         fm_resp_vec(i,1) = fm(pm_h,pm_w, cl_displ(i));
  end

############################################################

function[c]=lookup_cm_2d_uit(x,matx,F,h,Q)


N=size(matx,2);


for j=1:N
    q(j)=sqrt((norm(x-matx(:,j)))^2+h);
end


a=Q\F;
n=size(a,2);

for t=1:n
```

```
    c(t,1)=q*a(:,t);
end

##################################################################

function[Opt]=opt2d_manifold_uit(x,y,c0,f0,dc,df,Uc,matx,F,h,Q)

c(:,1)=lookup_cm_2d_uit(x,matx,F,h,Q);
Opt=norm((c(:,1)-c0)+(dc*pinv(df)+eye(9)-Uc*Uc')*(f0-y))/norm(y);

##################################################################
```

9.2.3 4D-model met stabilisatie

We vermelden alleen de hoofdfunctie. Andere functies staan al vermeld bij 1D en 2D modellen.

```
function[antwoord] = manifold4d_uitbreiding()
clear all
clc
format long

y=12*ones(9,1);
S=cell(0);
OG=[7,1,1,1]*1e-3;
BG=[45,45,9,45]*1e-3;

X_pm_h=[16.5, 35.5]*1e-3;
X_pm_w=[12, 34]*1e-3;
X_coil_h=[3, 7]*1e-3;
X_coil_w=[12, 34]*1e-3;
matx=allcomb(X_pm_h,X_pm_w,X_coil_h,X_coil_w)';

for i=1:size(matx,2)
    F(i,:)=lookup_fm_4d(matx(:,i));
end

%Ik neem aan dat zowel p_streep als S0 gelijk zijn aan I.
%Vector x0 bepalen.
k=1;
x(:,k)=fminsearchbnd(@(x) compute_cm_4d(x,y,matx,F),...
    [26;23;9;23]*1e-3,OG,BG,optimset('TolX',1e-4,'TolFun',1e-4));
f(:,k)=lookup_fm_4d(x(:,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k)
r_F=size(F,1);
```

```
F(r_F+1,:)=f(:,k)'

%Een rij van F weglaten
k_matx=size(matx,2);
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[]

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[]

c(:,k)=lookup_cm_4d(x(:,k),matx,F);
resp(:,k)=f(:,k);
%x1 bepalen.
k=k+1;
x(:,k)=fminsearchbnd(@(x) opt_manifold_4d(x,y,c(:,k-1),f(:,k-1),...
    zeros(9,1),zeros(9,1),zeros(9),matx,F), x(:,k-1),OG,BG,...
    optimset('TolX',1e-4,'TolFun',1e-4));
f(:,k)=lookup_fm_4d(x(:,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k)
r_F=size(F,1);
F(r_F+1,:)=f(:,k)'

%Een rij van F weglaten
k_matx=size(matx,2);
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[]

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[]

c(:,k)=lookup_cm_4d(x(:,k),matx,F);
dc(:,1)=c(:,k-1)-c(:,k);
df(:,1)=f(:,k-1)-f(:,k);
[Uc,Sc,Vc]=svd(dc)
S{1,length(S)+1}=eye(9);
resp(:,k)=f(:,k);

%Optimale waarde van x zoeken.
```

```
while k<=29
    k=k+1
    x(:,k)=fminsearchbnd(@(x) opt_manifold_4d(x,y,c(:,k-1),f(:,k-1),...
        dc,df,Uc,matx,F), x(:,k-1),OG,BG,...
        optimset('TolX',1e-4,'TolFun',1e-4));
    f(:,k)=lookup_fm_4d(x(:,k));

    %Het optimum toevoegen aan matx en bijbehorende output aan F.
    k_matx=size(matx,2);
    matx(:,k_matx+1)=x(:,k)
    r_F=size(F,1);
    F(r_F+1,:)=f(:,k)'

    %Een rij van F weglaten
    k_matx=size(matx,2);
    r_F=size(F,1);
    for j=1:r_F
        normF(j)=norm(y'-F(j,:))/norm(y');
    end
    [v_maxF,i_maxF]=max(normF);
    F(i_maxF,:)=[]

    %Bijbehorende kolom van matx ook weglaten
    matx(:,i_maxF)=[]

    c(:,k)=lookup_cm_4d(x(:,k),matx,F);
    clear dc
    clear df
    for i=1:min(4,k-1)
        dc(:,i)=c(:,k-i)-c(:,k);
        df(:,i)=f(:,k-i)-f(:,k);
    end
    [Uc,Sc,Vc]=svd(dc);
    [Uf,Sf,Vf]=svd(df);
    S{1,length(S)+1}=df*pinv(dc)+(eye(9)-Uf*Uf')*(eye(9)-Uc*Uc');
    resp(:,k)=f(:,k);
end


for l=1:k
    NN(l)=norm(resp(:,l)-y)/norm(y)
end
[value_min_NN, it_min_NN]=min(NN)
antwoord(:,1)=x(:,it_min_NN)
aantal_iter_MM=length(x)
total_number_iter_fijn_model=aantal_iter_MM+size(matx,2)

plot(1:k,NN,'r')
xlabel('Iteratie');
```

```
ylabel('Geschaald verschil');
```

9.2.4 7D-model met stabilisatie

Ook bij dit model wordt alleen de hoofdfunctie vermeld.

```
function[antwoord] = manifold7d_uitbreiding()
clear all
clc
format long

y=12*ones(9,1);
S=cell(0);
OG=[7,1,1,1,1,1,1]*1e-3;
BG=[45,45,9,45,45,45,45]*1e-3;


X_pm_h=[16.5, 35.5]*1e-3;
X_pm_w=[12, 34]*1e-3;
X_coil_h=[3, 7]*1e-3;
X_coil_w=[12, 34]*1e-3;
X_core_b=[12, 34]*1e-3;
X_core_t=[12, 34]*1e-3;
X_core_l=[12, 34]*1e-3;


matx=allcomb(X_pm_h,X_pm_w,X_coil_h,X_coil_w,X_core_b,X_core_t,X_core_l)';

for i=1:size(matx,2)
    F(i,:)=lookup_fm_7d(matx(:,i));
end

%Ik neem aan dat zowel p_streep als S0 gelijk zijn aan I.
%Vector x0 bepalen.
k=1;
x(:,k)=fminsearchbnd(@(x) compute_cm_7d(x,y,matx,F),...
    [30;10;9;10;15;30;20]*1e-3,OG,BG,...
    optimset('TolX',1e-4,'TolFun',1e-4,'MaxFunEvals',5000));
f(:,k)=lookup_fm_7d(x(:,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k)
r_F=size(F,1);
F(r_F+1,:)=f(:,k)'

%Een rij van F weglaten
k_matx=size(matx,2);
```

```
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[]

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[]

c(:,k)=lookup_cm_7d(x(:,k),matx,F);
%x1 bepalen.
k=k+1;
x(:,k)=fminsearchbnd(@(x) opt_manifold_7d(x,y,c(:,k-1),f(:,k-1),...
    zeros(9,1),zeros(9,1),zeros(9),matx,F), x(:,k-1),OG,BG,...
    optimset('TolX',1e-4,'TolFun',1e-4,'MaxFunEvals',5000));
f(:,k)=lookup_fm_7d(x(:,k));

%Het optimum toevoegen aan matx en bijbehorende output aan F.
k_matx=size(matx,2);
matx(:,k_matx+1)=x(:,k)
r_F=size(F,1);
F(r_F+1,:)=f(:,k)'

%Een rij van F weglaten
k_matx=size(matx,2);
r_F=size(F,1);
for j=1:r_F
    normF(j)=norm(y'-F(j,:))/norm(y');
end
[v_maxF,i_maxF]=max(normF);
F(i_maxF,:)=[]

%Bijbehorende kolom van matx ook weglaten
matx(:,i_maxF)=[]

c(:,k)=lookup_cm_7d(x(:,k),matx,F);
dc(:,1)=c(:,k-1)-c(:,k);
df(:,1)=f(:,k-1)-f(:,k);
[Uc,Sc,Vc]=svd(dc);
S{1,length(S)+1}=eye(9);
resp(:,k)=f(:,k);

%Optimale waarde van x zoeken.
while k<=50
    k=k+1;
    x(:,k)=fminsearchbnd(@(x) opt_manifold_7d(x,y,c(:,k-1),f(:,k-1),...
        dc,df,Uc,matx,F), x(:,k-1),OG,BG,...
        optimset('TolX',1e-4,'TolFun',1e-4,'MaxFunEvals',5000));
```

```
    f(:,k)=lookup_fm_7d(x(:,k));

    %Het optimum toevoegen aan matx en bijbehorende output aan F.
    k_matx=size(matx,2);
    matx(:,k_matx+1)=x(:,k)
    r_F=size(F,1);
    F(r_F+1,:)=f(:,k)'

    %Een rij van F weglaten
    k_matx=size(matx,2);
    r_F=size(F,1);
    for j=1:r_F
        normF(j)=norm(y'-F(j,:))/norm(y');
    end
    [v_maxF,i_maxF]=max(normF);
    F(i_maxF,:)=[]

    %Bijbehorende kolom van matx ook weglaten
    matx(:,i_maxF)=[]

    c(:,k)=lookup_cm_7d(x(:,k),matx,F);
    for i=1:min(7,k-1)
        dc(:,i)=c(:,k-i)-c(:,k);
        df(:,i)=f(:,k-i)-f(:,k);
    end
    [Uc,Sc,Vc]=svd(dc);
    [Uf,Sf,Vf]=svd(df);
    S{1,length(S)+1}=df*pinv(dc)+(eye(9)-Uf*Uf')*(eye(9)-Uc*Uc');
    resp(:,k)=f(:,k);
end

for l=1:k
    NN(l)=norm(resp(:,l)-y)/norm(y)
end
[value_min_NN, it_min_NN]=min(NN)
[value_max_NN, it_max_NN]=max(NN)
antwoord(:,1)=x(:,it_min_NN)
aantal_iter_MM=length(x)
total_number_iter_fijn_model=aantal_iter_MM+size(matx,2)

plot(1:k,NN,'r')
xlabel('Iteratie');
ylabel('Geschaald verschil');
```

9.3 fminsearchbnd.m en allcomb.m

Deze functie zijn beschikbaar op *MathWorks*.

```
function [x,fval,exitflag,output]=fminsearchbnd3(fun,x0,LB,UB,options,varargin)
% FMINSEARCHBND: FMINSEARCH, but with bound constraints by transformation
% usage: x=FMINSEARCHBND(fun,x0)
% usage: x=FMINSEARCHBND(fun,x0,LB)
% usage: x=FMINSEARCHBND(fun,x0,LB,UB)
% usage: x=FMINSEARCHBND(fun,x0,LB,UB,options)
% usage: x=FMINSEARCHBND(fun,x0,LB,UB,options,p1,p2,...)
% usage: [x,fval,exitflag,output]=FMINSEARCHBND(fun,x0,...)
%
% arguments:
%  fun, x0, options - see the help for FMINSEARCH
%
%  LB - lower bound vector or array, must be the same size as x0
%
%       If no lower bounds exist for one of the variables, then
%       supply -inf for that variable.
%
%       If no lower bounds at all, then LB may be left empty.
%
%       Variables may be fixed in value by setting the corresponding
%       lower and upper bounds to exactly the same value.
%
%  UB - upper bound vector or array, must be the same size as x0
%
%       If no upper bounds exist for one of the variables, then
%       supply +inf for that variable.
%
%       If no upper bounds at all, then UB may be left empty.
%
%       Variables may be fixed in value by setting the corresponding
%       lower and upper bounds to exactly the same value.
%
% Notes:
%
%  If options is supplied, then TolX will apply to the transformed
%  variables. All other FMINSEARCH parameters should be unaffected.
%
%  Variables which are constrained by both a lower and an upper
%  bound will use a sin transformation. Those constrained by
%  only a lower or an upper bound will use a quadratic
%  transformation, and unconstrained variables will be left alone.
%
%  Variables may be fixed by setting their respective bounds equal.
%  In this case, the problem will be reduced in size for FMINSEARCH.
%
%  The bounds are inclusive inequalities, which admit the
```

```
%  boundary values themselves, but will not permit ANY function
%  evaluations outside the bounds. These constraints are strictly
%  followed.
%
%  If your problem has an EXCLUSIVE (strict) constraint which will
%  not admit evaluation at the bound itself, then you must provide
%  a slightly offset bound. An example of this is a function which
%  contains the log of one of its parameters. If you constrain the
%  variable to have a lower bound of zero, then FMINSEARCHBND may
%  try to evaluate the function exactly at zero.
%
%
% Example usage:
% rosen = @(x) (1-x(1)).^2 + 105*(x(2)-x(1).^2).^2;
%
% fminsearch(rosen,[3 3])     % unconstrained
% ans =
%    1.0000    1.0000
%
% fminsearchbnd(rosen,[3 3],[2 2],[])     % constrained
% ans =
%    2.0000    4.0000
%
% See test_main.m for other examples of use.
%
%
% See also: fminsearch, fminspleas
%
%
% Author: John D'Errico
% E-mail: woodchips@rochester.rr.com
% Release: 4
% Release date: 7/23/06

% size checks
xsize = size(x0);
x0 = x0(:);
n=length(x0);

if (nargin<3) || isempty(LB)
  LB = repmat(-inf,n,1);
else
  LB = LB(:);
end
if (nargin<4) || isempty(UB)
  UB = repmat(inf,n,1);
else
  UB = UB(:);
end
```

```
if (n~=length(LB)) || (n~=length(UB))
  error 'x0 is incompatible in size with either LB or UB.'
end

% set default options if necessary
if (nargin<5) || isempty(options)
  options = optimset('fminsearch');
end

% stuff into a struct to pass around
params.args = varargin;
params.LB = LB;
params.UB = UB;
params.fun = fun;
params.n = n;
% note that the number of parameters may actually vary if
% a user has chosen to fix one or more parameters
params.xsize = xsize;
params.OutputFcn = [];

% 0 --> unconstrained variable
% 1 --> lower bound only
% 2 --> upper bound only
% 3 --> dual finite bounds
% 4 --> fixed variable
params.BoundClass = zeros(n,1);
for i=1:n
  k = isfinite(LB(i)) + 2*isfinite(UB(i));
  params.BoundClass(i) = k;
  if (k==3) && (LB(i)==UB(i))
    params.BoundClass(i) = 4;
  end
end

% transform starting values into their unconstrained
% surrogates. Check for infeasible starting guesses.
x0u = x0;
k=1;
for i = 1:n
  switch params.BoundClass(i)
    case 1
      % lower bound only
      if x0(i)<=LB(i)
        % infeasible starting value. Use bound.
        x0u(k) = 0;
      else
        x0u(k) = sqrt(x0(i) - LB(i));
      end
```

```
      % increment k
      k=k+1;
    case 2
      % upper bound only
      if x0(i)>=UB(i)
        % infeasible starting value. use bound.
        x0u(k) = 0;
      else
        x0u(k) = sqrt(UB(i) - x0(i));
      end

      % increment k
      k=k+1;
    case 3
      % lower and upper bounds
      if x0(i)<=LB(i)
        % infeasible starting value
        x0u(k) = -pi/2;
      elseif x0(i)>=UB(i)
        % infeasible starting value
        x0u(k) = pi/2;
      else
        x0u(k) = 2*(x0(i) - LB(i))/(UB(i)-LB(i)) - 1;
        % shift by 2*pi to avoid problems at zero in fminsearch
        % otherwise, the initial simplex is vanishingly small
        x0u(k) = 2*pi+asin(max(-1,min(1,x0u(k))));
      end

      % increment k
      k=k+1;
    case 0
      % unconstrained variable. x0u(i) is set.
      x0u(k) = x0(i);

      % increment k
      k=k+1;
    case 4
      % fixed variable. drop it before fminsearch sees it.
      % k is not incremented for this variable.
  end

end
% if any of the unknowns were fixed, then we need to shorten
% x0u now.
if k<=n
  x0u(k:n) = [];
end
```

```
% were all the variables fixed?
if isempty(x0u)
  % All variables were fixed. quit immediately, setting the
  % appropriate parameters, then return.

  % undo the variable transformations into the original space
  x = xtransform(x0u,params);

  % final reshape
  x = reshape(x,xsize);

  % stuff fval with the final value
  fval = feval(params.fun,x,params.args{:});

  % fminsearchbnd was not called
  exitflag = 0;

  output.iterations = 0;
  output.funcount = 1;
  output.algorithm = 'fminsearch';
  output.message = 'All variables were held fixed by the applied bounds';

  % return with no call at all to fminsearch
  return
end

% Check for an outputfcn. If there is any, then substitute my
% own wrapper function.
if ~isempty(options.OutputFcn)
  params.OutputFcn = options.OutputFcn;
  options.OutputFcn = @outfun_wrapper;
end

% now we can call fminsearch, but with our own
% intra-objective function.
[xu,fval,exitflag,output] = fminsearch(@intrafun,x0u,options,params);

% undo the variable transformations into the original space
x = xtransform(xu,params);

% final reshape to make sure the result has the proper shape
x = reshape(x,xsize);

% Use a nested function as the OutputFcn wrapper
  function stop = outfun_wrapper(x,varargin);
    % we need to transform x first
    xtrans = xtransform(x,params);

    % then call the user supplied OutputFcn
```

```
    stop = params.OutputFcn(xtrans,varargin{1:(end-1)});

  end

end % mainline end

% ======================================
% ========= begin subfunctions =========
% ======================================
function fval = intrafun(x,params)
% transform variables, then call original function

% transform
xtrans = xtransform(x,params);

% and call fun
fval = feval(params.fun,reshape(xtrans,params.xsize),params.args{:});

end % sub function intrafun end

% ======================================
function xtrans = xtransform(x,params)
% converts unconstrained variables into their original domains

xtrans = zeros(params.xsize);
% k allows some variables to be fixed, thus dropped from the
% optimization.
k=1;
for i = 1:params.n
  switch params.BoundClass(i)
    case 1
      % lower bound only
      xtrans(i) = params.LB(i) + x(k).^2;

      k=k+1;
    case 2
      % upper bound only
      xtrans(i) = params.UB(i) - x(k).^2;

      k=k+1;
    case 3
      % lower and upper bounds
      xtrans(i) = (sin(x(k))+1)/2;
      xtrans(i) = xtrans(i)*(params.UB(i) - params.LB(i)) + params.LB(i);
      % just in case of any floating point problems
      xtrans(i) = max(params.LB(i),min(params.UB(i),xtrans(i)));

      k=k+1;
    case 4
```

```
      % fixed variable, bounds are equal, set it at either bound
      xtrans(i) = params.LB(i);
    case 0
      % unconstrained variable.
      xtrans(i) = x(k);

      k=k+1;
  end
end

end % sub function xtransform end

####################################

function A = allcomb(varargin)
% ALLCOMB - All combinations
%    B = ALLCOMB(A1,A2,A3,...,AN) returns all combinations of the elements
%    in A1, A2, ..., and AN. B is P-by-N matrix is which P is the product
%    of the number of elements of the N inputs.
%    Empty inputs yields an empty matrix B of size 0-by-N. Note that
%    previous versions (1.x) simply ignored empty inputs.
%
%    Example:
%       allcomb([1 3 5],[-3 8],[0 1]) ;
%         1  -3   0
%         1  -3   1
%         1   8   0
%         ...
%         5  -3   1
%         5   8   0
%         5   8   1
%
%    ALLCOMB(A1,..AN,'matlab') causes the first column to change fastest.
%    This is more consistent with matlab indexing. Example:
%    allcomb(1:2,3:4,5:6,'matlab') %->
%      1   3   5
%      2   3   5
%      1   4   5
%      ...
%      2   4   6
%
%    This functionality is also known as the cartesian product.
%
%    See also NCHOOSEK, PERMS,
%    and COMBN (Matlab Central FEX)

% for Matlab R13+
% version 2.1 (feb 2011)
% (c) Jos van der Geest
```

```
% email: jos@jasen.nl

% History
% 1.1 (feb 2006), removed minor bug when entering empty cell arrays;
%     added option to let the first input run fastest (suggestion by JD)
% 1.2 (jan 2010), using ii as an index on the left-hand for the multiple
%     output by NDGRID. Thanks to Jan Simon, for showing this little trick
% 2.0 (dec 2010). Bruno Luong convinced me that an empty input should
% return an empty output.
% 2.1 (feb 2011). A cell as input argument caused the check on the last
%      argument (specifying the order) to crash.

error(nargchk(1,Inf,nargin)) ;

% check for empty inputs
q = ~cellfun('isempty',varargin) ;
if any(~q),
    warning('ALLCOMB:EmptyInput','Empty inputs result in an empty output.') ;
    A = zeros(0,nargin) ;
else

    ni = sum(q) ;

    argn = varargin{end} ;
    ischar(argn);
    if ischar(argn) && (strcmpi(argn,'matlab') || strcmpi(argn,'john')),
        % based on a suggestion by JD on the FEX
        ni = ni-1 ;
        ii = 1:ni ;
        q(end) = 0 ;
    else
        % enter arguments backwards, so last one (AN) is changing fastest
        ii = ni:-1:1 ;
    end

    if ni==0,
        A = [] ;
    else
        args = varargin(q) ;
        if ~all(cellfun('isclass',args,'double')),
            error('All arguments should be arrays of doubles') ;
        end
        if ni==1,
            A = args{1}(:) ;
        else
            % flip using ii if last column is changing fastest
            [A{ii}] = ndgrid(args{ii}) ;
            % concatenate
            A = reshape(cat(ni+1,A{:}),[],ni) ;
```

```
        end
    end
end
```

10 Referenties

Referenties

[1] D. Echeverria, D. Lahaye, P. Hemket:*Space Mapping and Deffect Correction*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands.

[2] J. Lagarias, J. Reeds, M. Wright, P. Wright:*Convergence properies of the Nelder-Mead Simplex method in low dimensions*, Society for Industrial and Applied Mathematics, 1998.

[3] A. Canova, G. Gruosso: *Magnetic Design Optimization and Objective Function Approximation*, IEEE TRANSACTIONS ON MAGNETICS, VOL. 39, NO. 5, SEPTEMBER 2003.

[4] *COMSOL Multiphysics Modeling Guide* Version: November 2008 Comsol 3.5a.

[5] D. Lahaye, A. Canova, G. Grusso, M. Repetto: *Adaptive manifold-mapping using multiquadric interpolation applied to lineair actuator design*, Centrum voor Wiskunde en Informatica (CWI), Report MAS-E0622, SEPTEMBER 2006.

[6] D.Echeverria *Two new variants of the manifold-mapping technique*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, 2007.