CHALMERS | 

GÖTEBORGS UNIVERSITET

Optimering av kylanordning för behandling med hypertermi

Implementation av algoritm för att lösa ett minimeringsproblem begränsat av en partiell differentialekvation

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Erik von Brömssen
Rebecca Henrysson
Emelie Johansson
Olivia Stensöta

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

Optimering av kylanordning för behandling med hypertermi

Implementation av algoritm för att lösa ett minimeringsproblem begränsat av en partiell differentialekvation

Kandidatarbete i matematik inom civilingenjörsprogrammet Automation och mekatronik vid Chalmers

Olivia Stensöta

Kandidatarbete i matematik inom civilingenjörsprogrammet Maskinteknik vid Chalmers

Rebecca Henrysson

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk fysik vid Chalmers

Emelie Johansson

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers

Erik von Brömssen

Handledare:	Maria Roginskaya	Matematiska vetenskaper
	Hana Dobsicek Trefna	Elektroteknik
Examinator:	Ulla Dinger	

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

Förord

Vi vill börja med att tacka våra handledare Maria Roginskaya och Hana Dobsicek Trefna för deras goda vägledning och stöd under hela arbetet. Ett speciellt tack går också till Saül Llacer Navarro, för assistans i beräkningsprogrammet Solidworks samt assistans vid laborationen. Ett tack går även till Massimiliano Zanoli, för hans hjälp med simuleringsdata samt utbildning i CST.

Bidragsrapport

Arbetsfördelningen har i stort sätt fördelats jämnt i gruppen.

Erik von Brömssen har haft huvudansvar för utvecklandet av optimeringsalgoritmen i Matlab.

Rebecca Henrysson och Olivia Stensöta har haft huvudansvar för de simuleringar som skett i Solidworks.

Emelie Johansson har haft huvudansvar för rapporten och 3D-utskriften av gjutningsformen.

Samtliga deltagare har varit med vid de experimentella testerna.

Samtliga deltagare har även hjälpts åt då det behövs.

Alla deltagare har skrivit olika delar av rapporten men den har bearbetats och korrekturlästs av alla.

I gruppens dagbok finns tydligare anteckningar om vad som gjorts under arbetetsgång och även i tidsloggen går det att finna vad enskilda gruppmedlemmar gjort.

Populärvetenskaplig presentation

Kan matematik optimera cancervården?

Med hjälp av matematiska optimeringsmetoder beräknas den mest optimala placeringen av en kylkanal inuti en applikator som används vid cancerbehandling. Applikatorn sänder ut mikrovågsstrålning som värmer upp tumörer och uppvärmningen i sin tur hjälper den huvudsakliga cancerbehandlingen, såsom strålning eller kemoterapi. För att uppvärmningen inte ska skada frisk vävnad, behövs en väl fungerande kylanordning, vars utveckling nu har tagit ett steg framåt.

Det är idag inte helt ovanligt att drabbas av cancer, en sjukdom som forskare världen över kämpar med att utrota. Vid behandling av carcinom, alltså cancer som uppstår som större tumörer, har man sett positiva resultat vid uppvärmning av tumören, så kallad hypertermi. Behandlingen har använts tillsammans med andra läkemedel, eftersom då tumörvävnader värms upp till cirka 40-43 °C bidrar värmen till ett bättre resultat vid strålbehandling, men även effektivare behandling med kemoterapi. Den höga värmen kan också medföra att vävnader genomgår celldöd. De höga temperaturer som behövs för hypertermibehandling kan ge biverkningar i form av värmeblåsor på friska vävnader. För att förhindra blåsornas uppkomst behövs en kylanordning som kan kyla huden samtidigt som tumören värms upp.

Cancer i halsområdet är intressant att behandla med hypertermi då det ofta är svårt att operera tumörer i området och det därför krävs andra behandlingssätt. Ett sätt att behandla denna typ av tumörer är med hjälp av en anordning bestående av nio antenner, som placeras runt halsen. Antennerna skickar ut mikrovågsstrålning, som fokuseras i tumörerna, vilket bidrar till en uppvärmning koncentrerad just i tumörerna. Antennerna är så pass stora att de behöver placeras med ett avstånd på 10 cm från huden om alla nio skall få plats. Eftersom mycket av effekten från antennerna går förlorad, på grund av reflektion, då mikrovågorna passerar från luft till hud, är det fördelaktigt att ha ett annat medium mellan antenner och hud. Ett slime-liknande material, en så kallad *hydrogel*, som består till 99 % av vatten och till 1 % av naturliga polymerer, placeras mellan antenner och hud. Denna hydrogel används, förutom som medium för mikrovågorna, för att kyla de friska vävnaderna. Hydrogelen ger dock inte tillräcklig kylning själv, därför placeras kylkanaler av plast inuti hydrogelen. Tanken är att kallt vatten ska flöda genom kylkanalerna under behandlingen.

För att få en så effektiv kylning som möjligt användes ett beräkningsprogram för att optimera hur kylkanalerna skall placeras och formas. Optimeringsalgoritmen utgick från att cirka tio punkter placerades ut på olika höjder, därefter drogs en linje mellan dessa punkter för att få kylkanalens form. Algoritmen placerade ut punkterna med målet att minimera medeltemperaturen på hela huden.

Det är alltid viktigt att kontrollera beräkningar på olika sätt för att styrka resultaten. Detta gjordes genom att den beräknade kylkanalen målades upp i ett rit- och simuleringsprogram för att testa ifall temperaturangivelsen i optimeringsalgoritmen stämde. Beräkningarna kontrollerades även genom ett test i verklig miljö där skillnaden på en helt rak och en böjd kylkanal testades.

Formen som erhölls från algoritmen var för svår för att kunna sättas upp under laborationen, därför fick den förenklas. Efter förenklingen spändes en kylkanal upp i en gjutningsform utefter den förenklade formen. Därefter hälldes hydrogelen i och allting fick stå i kylskåp över natten för att stelna, för att morgonen efter börja med laborationen. Även hydrogelen med den raka kylkanalen gjordes och fick stå i kylskåp.

Denna laboration gjordes inte med kylning, istället värmdes hydrogelen upp för att lättare kontrollera att både hydrogelen med den raka kylkanalen och hydrogelen med den böjda kylkanalen hade samma starttemperatur. Då hydrogelerna placerats i ett kylskåp över natten, vilket innebar att de båda hade samma temperatur som kylskåpet.

Samtliga tester, såväl beräkningar som simuleringar och laborationen, gav goda resultat; det som beräknades i beräkningsprogrammet stämde överens med verifieringen i simuleringsprogrammet.

De små skillnaderna som uppstod kan bero på att simuleringsprogrammet använder fler materialkoefficienter än beräkningsprogrammet. Resultatet av experimentet (som tidigare nämnt var en uppvärmning) var att en böjd kylkanal gav upphov till en inhomogen uppvärmning medan en rak kylkanal gav upphov till en homogen uppvärmning.

Algoritmen fungerar bra — men det finns rum för förbättringar. Till exempel kan man studera hur kylningen förändras under en behandlingstid, då den omgivande temperaturen stiger under en behandling. Med dessa vidareutvecklingar skulle hypertermibehandling innefattandes en hydrogel med kylanordning vara ännu ett steg närmare sjukhusens behandlingsrum och fler liv skulle kunna räddas.

Sammandrag

En optimeringsalgoritm för utformningen av en kylanordning, gjord för behandling av cancer med hypertermi, utvecklades. Ett optimeringsproblem begränsat av en partiell differentialekvation — den tidsoberoende värmeledningsekvationen $-\Delta u = f$ — ställdes upp över variabler som beskriver geometrin av två kylkanaler. Värmeledningsekvationen löstes i en domän Ω som representerar en del av en hypertermiapplikator. Målfunktionalen att minimera är medeltemperaturen på en patients hud, $\int_{\Gamma_1} u\, dS$, där Γ_1 är den del av Ω som representerar patientens hud. Denna algoritm implementerades med en gradient projection-metod och en finit differenslösare. Algoritmen verifierades även, både numeriskt och experimentellt med goda resultat. Resultatet påvisar en fungerande optimeringsalgoritm, dock under begränsningar. Med de förenklingar som gjorts har vi hittat en metod att optimera kylkanaler för en förbättrad kylning. Algoritmen behöver dock vidareutvecklas för att bättre spegla verkligheten och möjliga behandlingsfall bör undersökas. Med en fullt fungerande optimeringsalgoritm utan begränsningar kan behandling av hypertermi användas med minskad risk för brännskador.

Abstract

An optimization algorithm for a cooling system, created for cancer treatment with hyperthermia was developed. An optimization problem restricted by a partial differential equation — the time independent heat equation $-\Delta u = f$ — was formulated over the variables describing the geometry of two cooling channels. The heat equation was solved in a domain Ω that represents part of the hyperthermia applicator. The objective functional to minimize is the average temperature of the patients skin, $\int_{\Gamma_1} u\, dS$ where Γ_1 is the part of Ω that represents the patients skin. This algorithm was implemented with a gradient projection method and a finite difference solver. The algorithm was also verified, both numerically and experimentally with good results. The results demonstrate a functioning optimization algorithm, but with limitations. With the simplifications made we found a method to optimize cooling channels for an improved cooling. However, the algorithm needs to be further developed to better reflect reality and possible treatment cases should be investigated. With a fully functioning optimization algorithm without limitations, hyperthermia treatment can be used with reduced risk for burns.

Innehåll

1 Inledning **1**
- 1.1 Syfte . . . 2
- 1.2 Avgränsningar . . . 2
- 1.3 Samhälleliga och etiska aspekter . . . 2
- 1.4 Rapportens utformning . . . 3

2 Matematisk modell **3**
- 2.1 Förenklad modell . . . 3
  - 2.1.1 Geometri . . . 3
  - 2.1.2 Målfunktional . . . 3
  - 2.1.3 Bivillkor . . . 4
  - 2.1.4 Optimeringsalgoritm . . . 4
  - 2.1.5 Finit differenslösare . . . 4
- 2.2 Avancerad modell . . . 6
- 2.3 Parametrar för optimering . . . 6

3 Verifiering av algoritm **6**
- 3.1 Numerisk verifikation . . . 6
  - 3.1.1 Förenklad modell . . . 7
  - 3.1.2 Avancerad modell . . . 7
- 3.2 Experimentell verifikation . . . 8
  - 3.2.1 Förberedelser . . . 8
  - 3.2.2 Förväntade resultat . . . 10
  - 3.2.3 Utförande . . . 11

4 Resultat **11**
- 4.1 Numerisk verifikation - Förenklad modell . . . 11
- 4.2 Numerisk verifikation - Avancerad modell . . . 13
- 4.3 Experimentell verifikation . . . 15

5 Diskussion och slutsats **16**
- 5.1 Optimeringsalgoritm . . . 16
- 5.2 Verifiering av optimeringsalgoritm . . . 17
- 5.3 Samhälleliga och etiska aspekter . . . 17
- 5.4 Vidareutveckling och rekommendationer . . . 17
- 5.5 Slutsats och framtidsutsikt . . . 18

Referenser **19**

Appendix **20**

A Matlab-kod avancerad modell **20**
- A.1 Huvudskript . . . 20
- A.2 Gradient projection-algoritm . . . 21
- A.3 Finit differenslösare . . . 23
- A.4 Övriga funktioner . . . 27

B Matlab-kod enkel modell **28**
- B.1 Huvudskript . . . 28
- B.2 Gradient projection-algoritm . . . 29
- B.3 Finit differenslösare . . . 31
- B.4 Övriga funktioner . . . 35

C Matlab-kod laboration **36**
- C.1 Huvudskript . . . 36
- C.2 Gradient projection-algoritm . . . 37

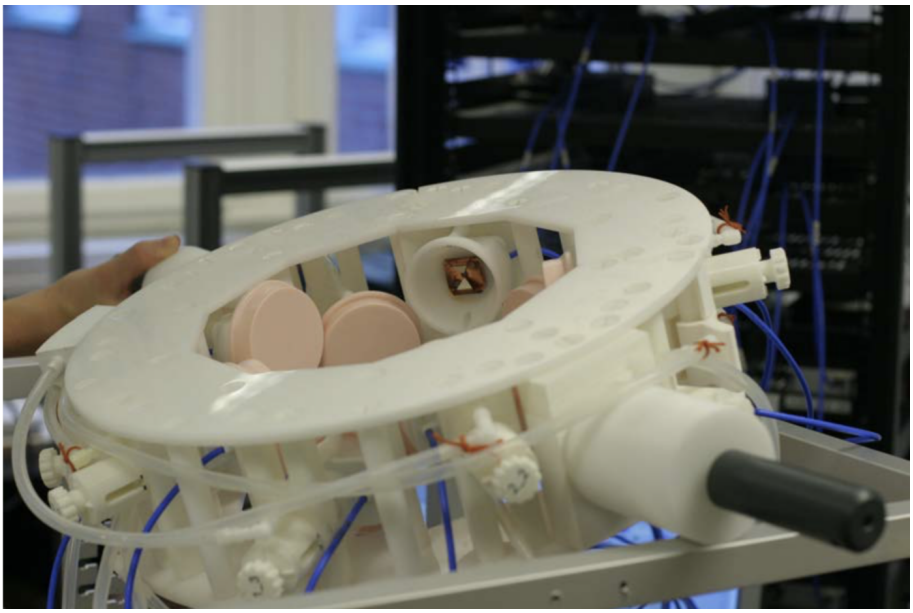
C.3 Finit differenslösare 39
C.4 Övriga funktioner 41

D Gemensamma funktioner **42**

1 Inledning

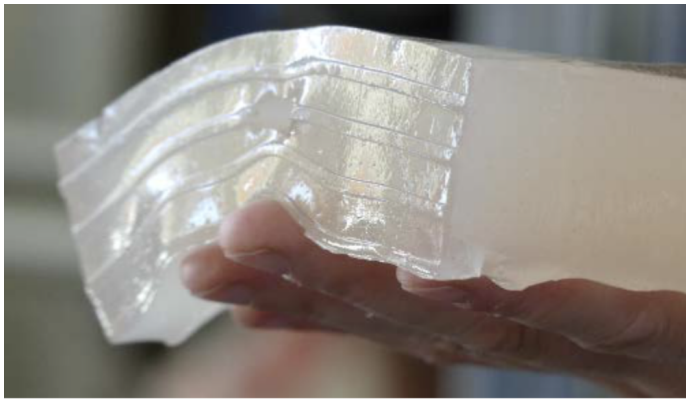
Begreppet hypertermi innebär extern uppvärmning av vävnad [1]. Metoden har ett flertal användningsområden, ett av dessa är inom cancerbehandling [2]. Behandlingen används som komplement till andra behandlingsmetoder, för då vävnader värms upp till $40-43\,^{\circ}\mathrm{C}$ ökar blodgenomströmningen och känsligheten för behandling med strålning och kemoterapi [1]. Vid så höga temperaturer kan tumörvävnader även genomgå celldöd, medan friska vävnader kan få blåsor då det uppstår oönskade *hotspots*, alltså områden där temperaturen överskrider $40-43\,^{\circ}\mathrm{C}$. Det är därför viktigt att rikta uppvärmningen så att endast cancercellerna påverkas och dessa hotspots undviks.

Microwave hyperthermia for cancer treatment är ett projekt som utförs på Chalmers [2]. En del av projektet innefattar utveckling av en applikator som placeras runt halsen på patienten och genom hypertermi behandlar tumörer inom halsområdet. Anordningen har nio antenner som sänder ut mikrovågor, se figur 1. På grund av den höga effekten behöver antennerna kylas, vilket görs med en kylanordning bestående av cirkulerande avjoniserat vatten [3]. Antennerna är fokuserade så att det uppstår maximal intensitet i tumörerna, men den närliggande vävnaden kommer oundvikligt att värmas upp.



Figur 1: En prototyp för applikatorn, utvecklad på Chalmers [4] (publicerad med tillstånd av Hana Dobsicek Trefna).

Då det är stor skillnad i permittiviteten på luft och mänsklig vävnad [5] absorberas en stor effekt i huden vid materialövergången. För att hindra att huden skadas av uppvärmningen från den absorberade effekten placeras i nuläget vattenfyllda påsar mellan antennerna och huden [6], då vatten har en permittivitet närmare mänsklig vävnad. Påsarna kan inte anpassas helt efter applikatorn vilket medför att det uppstår luftfickor. En lösning på problemet utvecklas på Chalmers [2]; en så kallad hydrogel placeras mellan antennerna och patientens hud. Hydrogelen består av 99 % vatten och 1 % hydrofila polymerer, se figur 2. Denna hydrogel kan bättre formas efter applikatorn till skillnad från påsarna, vilket innebär att luftfickor kan undvikas.



Figur 2: Hydrogelen [4] (publicerad med tillstånd av Hana Dobsicek Trefna).

Ett av hydrogelens syften är att kyla huden under behandlingen. Hydrogelen kan däremot inte på grund av dess låga värmeledningsförmåga [5] leda bort värme från patientens hud tillräckligt snabbt för att undvika brännskador på huden. Ett kylsystem till hydrogelen behöver därför introduceras. Tidigare har försök utförts med hål i hydrogelen där vatten flödat, vilket gav hög kyleffekt med minimal inverkan på det elektromagnetiska fältet (EM-fältet), däremot fanns problem med att fästa in- och utflöde av vattnet i hydrogelen [6]. En annan metod är att istället använda kylkanaler av plast som vatten flödar genom. Det är enklare att dra kylkanalen genom hela hydrogelen; då uppstår inte problemet med läckage vid hydrogelens öppning. Dock har kylkanalerna en viss effekt på EM-fältet då plasten kan påverka utspridningen av EM-fältet på oönskat sätt.

1.1 Syfte

Syftet med arbetet är att skapa en optimeringsalgoritm för kylning av huden under hypertermibehandling i halsområdet. De resulterande geometrierna ska vara tillräckligt effektiva för att undvika att den friska vävnaden skadas.

1.2 Avgränsningar

För att kunna nå syftet under den tid som avsatts för projektet har vissa avgränsningar gjorts. Dessa avgränsningar presenteras nedan.

Vi utgår från en förenklad geometri gentemot den verkliga applikatorn: hydrogelen förenklas till en ihålig cylinder av homogent material; antennernas egna kylsystem antas vara fullt fungerande med en konstant temperatur på 5 °C — hydrogelen modelleras därför med en konstant temperatur, 5 °C, på mantelytan. Vi tar inte heller hänsyn till hur utspridningen av EM-fältet påverkas av kylkanalerna.

Att kunna variera formen på kylankanalen godtyckligt under optimeringen vore optimalt, men då detta leder till en för komplex modell och möjligen en orealistisk form på kylkanalen valde vi att modellera kylkanalen som en styckvis linjär kurva mellan ett fåtal nodpunkter. Arbetet begränsas även till att enbart studera hur maximalt två kylkanaler påverkar temperaturen.

1.3 Samhälleliga och etiska aspekter

Kylkanalerna skapas av nylon 6, även kallat *perlon*, vilket är en slags konstfiber, närmare bestämt en polyamid [3]. När kläder av detta material tvättas lossnar mikroplaster vilket är skadligt för naturen på olika sätt; dels genom att bidra till nedskräpning, men mikroplasterna kan även hamna i dricksvatten och i havet där fiskar misstar detta för mat [7]. På så sätt riskerar alla djur att få i sig dessa farliga plaster. Det finns anledning att tro att mikroplaster kan lossna även när vatten

flödar genom kylkanalerna i vår hydrogel. Hydrogelen, däremot, är gjord av naturliga polymerer (LBG, xanthan samt agar [2]) och därmed nedbrytningsbar.

1.4 Rapportens utformning

Resterande sidor av rapporten ordnas som följande; först presenteras framtagandet av den matematiska modellen, hur denna utvecklats till en mer avancerad modell samt de parametrar som använts under optimeringen. Därefter följer ett avsnitt om hur den matematiska modellen verifierats i ett simuleringsprogram samt genom experiment i verklig miljö. Rapportens sista delar redovisar de resultat som åstadkommits samt diskussion och slutsats. Rapporten avslutas med rekommendationer för kommande studier inom ämnet.

2 Matematisk modell

Projektets kommande kapitel beskriver tillvägagångssättet för utvecklandet av en optimeringsalgoritm för ett kylsystem. Utvecklingen utfördes i olika steg; initialt utvecklades en förenklad modell som innehöll flertalet avgränsningar och därefter utvecklades en mer avancerad modell som tog hänsyn till bland annat antennernas inverkan.

I båda modellerna som beskrivs nedan används cylindriska koordinater (r,θ,z), där r är punktens avstånd från origo projicerat på xy-planet och θ är vinkeln mellan den positiva x-axeln och projektionen av punktens ortsvektor ner på xy-planet. Avbildningen $x = r\cos\theta$, $y = r\sin\theta$ ger då omvandlingen från cylindriska till kartesiska koordinater. Om värden anges på dessa koordinater anges alltid r och z i centimeter och θ i radianer.

2.1 Förenklad modell

Projektets första steg var att konstruera en enkel modell med ett antal förenklingar, detta för att bekräfta att vår tänkta lösning var rimlig och implementerbar. Förenklingarna innebar en optimering baserad på ett homogent värmeflöde från huden, att det tidsoberoende jämviktsläget betraktades och att endast en kylkanal användes.

2.1.1 Geometri

Hydrogelen modelleras som en ihålig cylinder med innerradien 10 cm, ytterradien 20 cm och höjden 10 cm, valda godtyckligt. Kylkanalen modelleras som en styckvis linjär kurva, interpolerad mellan n stycken noder. Låt $\mathcal{I}$ vara mängden av alla noder och låt nod $i \in \mathcal{I}$ ha koordinater (r_i, θ_i, z_i). Vi låter alla vinklar θ_i vara likformigt placerade på intervallet $[1/2,\, 2\pi - 1/2]$ och ställer upp ett optimeringsproblem över variablerna r_i och z_i.

2.1.2 Målfunktional

Målfunktionalen sattes till medeltemperaturen på hydrogelens insida, alltså ytan på patientens tänkta hud. Denna målfunktional beräknas genom att lösa den tidsoberoende värmeledningsekvationen $\Delta u = 0$, där $u = u(r,\theta,z)$ betecknar temperatur i grader Celsius, med hydrogelen som domän.

Värmeledningsekvationens randvärden var som följande: konstant 20 °C (cirka rumstemperatur) på hydrogelens ovan- och undersida, 5 °C (se avsnitt 1.2) på utsidan och ett värmeflöde på 100 $\mathrm{W/m^2}$ (valdes godtyckligt) i riktning med sidans normal in mot hydrogelen på insidan. Det sistnämnda randvillkoret syftade till att simulera hur patientens hals värms upp och ger upphov till ett värmeflöde in till hydrogelen. Kylkanalens inre volym inkluderades inte i domänen till ekvationen; istället

sattes ett konstant randvärde på 5 °C vid kanalernas yta, för att undvika en beräkningsintensiv simulering av vattenflöden eftersom detta endast är en enkel modell.

Målfunktionalen att minimera är alltså

$$\frac{1}{|\Gamma_1|}\int_{\Gamma_1} u\,dS$$

där $\Gamma_1 = \{(r,\theta,z) : r = 0{,}1,\ 0 \leq \theta < 2\pi,\ 0 \leq z \leq 0{,}1\}$ betecknar den inre sidan av hydrogelen, $|\Gamma_1|$ dess area och u är lösningen till den tidsoberoende värmeledningsekvationen nämnd ovan.

2.1.3 Bivillkor

Bivillkor i problemet var endast att varje nods radie r_i och höjd h_i var begränsad; $r_{min} \leq r_i \leq r_{max}$ och $h_{min} \leq h_i \leq h_{max}$. Dessa konstanters värden valdes till $h_{min} = r_{min} = 2\,\text{cm}$ och $h_{max} = r_{max} = 8\,\text{cm}$, för att ha utrymme mellan kylkanalen och hydrogelens kant. Sammanfattningsvis blev optimeringsproblemet att

$$\begin{aligned} \text{minimera} \quad & \frac{1}{|\Gamma_1|}\int_{\Gamma_1} u\,dS \\ \text{under} \quad & r_{min} \leq r_i \leq r_{max} \quad i \in \mathcal{I}, \\ & h_{min} \leq h_i \leq h_{max} \quad i \in \mathcal{I}, \end{aligned}$$

där $\mathcal{I}$ är mängden av index för noderna på kylkanalen, u är lösningen till den tidsoberoende värmeledningsekvationen och Γ_1 är randen definierad ovan.

2.1.4 Optimeringsalgoritm

Vi implementerade en optimeringsalgoritm från grunden istället för att använda en färdig funktion i beräkningsprogrammet som användes, i det fallet Matlab; detta på grund av att varje approximativ gradient av målfunktionalen var beräkningsintensiv och att dessa färdiga, mer komplexa algoritmer skulle ta lång tid att exekvera. Vi implementerade då en *gradient projection*-metod. Denna metod utgår från en *gradient descent*-metod (eller *steepest descent*) som, givet en startpunkt $\boldsymbol{x}^0$, itererar via $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^k \nabla f(\boldsymbol{x}^k)$ med *steglängd* α^k för att försöka hitta en minimipunkt till f. En gradient projection-metod har även tillägget att den aktuella punkten projiceras in i den tillåtna mängden X, bestämd av problemets bivillkor, via

$$\boldsymbol{x}^{k+1} = \text{Proj}_X[\boldsymbol{x}^k - \alpha^k \nabla f(\boldsymbol{x}^k)],$$

där Proj_X är en operator som projicerar en punkt i $\mathbb{R}^n$ in till mängden $X \subset \mathbb{R}^n$. Valet av steglängd kan studeras noggrant, men i detta fall används steglängden $\alpha^k = 0{,}005/(k+1)$. Steglängden konvergerar mot noll för att minska risken att algoritmen alltid tar för långa steg för att konvergera. Just $1/(k+1)$ valdes med tanke på att den harmoniska serien är divergent: följder som $1/(k+1)^p$ med $p > 1$ hade dock gått snabbt mot noll och riskerat att aldrig nå en optimal punkt. Faktorn 0,005 ansågs lämplig efter några tester.

Optimeringsalgoritmen som implementerades var som följande: givet en vektor $\boldsymbol{c}^k$ innehållandes kylkanalens noders radier följt av höjder, beräkna målfunktionalsvärdet $f(\boldsymbol{c}^k)$ och approximera dess gradient $\nabla f(\boldsymbol{c}^k)$. Om $|\nabla f(\boldsymbol{c}^k)|$ är mindre än en tolerans TOL_g, avsluta. Om inte, projicera $\boldsymbol{c}^k - \nabla f(\boldsymbol{c}^k)$ in i den tillåtna mängden definierad av bivillkoren ovan, kalla denna punkt $\boldsymbol{c}^{k+1}$. Om $|\boldsymbol{c}^k - \boldsymbol{c}^{k+1}|$ är mindre än en tolerans TOL_p, avsluta. Om inte, iterera med $\boldsymbol{c}^{k+1}$ som den nya aktuella punkten.

2.1.5 Finit differenslösare

Målfunktionalen krävde som tidigare nämnt en lösning av värmeledningsekvationen; detta gjordes med en finit differenslösare. Denna lösare implementerades också från grunden (specifikt byggd för

vår ekvation och domän) då en del av domänens geometri — kylkanalen — snabbt skulle kunna uppdateras vilket inte verkade möjligt med tillräcklig enkelhet i redan existerande verktyg.

Cylindriska koordinater lämpade sig väl till geometrin av domänen; en partition av denna skapades med ett likformigt rutnät i cylindriska koordinater bestående av noderna i $\mathcal{T}_h = \{(i,j,k) : i = 0,\ldots,N_r-1,\ j=0,\ldots,N_\theta-1,\ k=0,\ldots,N_z-1\}$ där de positiva heltalen N_r, N_θ, N_z är antalet punkter längs varje dimension. Med domänens dimensioner (i cm) i åtanke låter vi $h_r = 10/N_r$, $h_\theta = 2\pi/N_\theta$ och $h_z = 10/N_z$. Vi söker en approximativ lösning $\hat{u}$ till värmeledningsekvationen definierad på $\mathcal{T}_h$ och ser att den cylindriska Laplacianen diskretiserad med centrala differenskvoter blir

$$\begin{aligned}\Delta\hat{u}_{i,j,k} = &\frac{\hat{u}_{i+1,j,k} - 2\hat{u}_{i,j,k} + \hat{u}_{i-1,j,k}}{h_r^2} + \frac{\hat{u}_{i+1,j,k} - \hat{u}_{i-1,j,k}}{2rh_r} \\ &+ \frac{\hat{u}_{i,j+1,k} - 2\hat{u}_{i,j,k} + \hat{u}_{i,j-1,k}}{r^2h_\theta^2} + \frac{\hat{u}_{i,j,k+1} - 2\hat{u}_{i,j,k} + \hat{u}_{i,j,k-1}}{h_z^2},\end{aligned} \tag{1}$$

där r är radien i meter som motsvarar den diskreta punkten (i,j,k).

Vi låter $\mathcal{T}_h$ vara sådan att $(i,j,k) = (0,j,k)$ motsvarar Γ_1; i dessa punkter gäller Neumannvillkor, så $\hat{u}_{0,j,k}$ måste beräknas, men ekvation (1) kan inte användas då $\hat{u}_{-1,j,k}$ aldrig är definierad. Givet att randvillkoret är ett värmeflöde q i positiv radiell riktning (1,0,0) får vi via Fouriers lag ($\boldsymbol{q} = -k\nabla u$) och en central approximation av förstaderivatan att

$$\frac{\hat{u}_{i+1,j,k} - \hat{u}_{i-1,j,k}}{2h_r} = -\frac{1}{k}q$$

där k är värmeledningsförmågan i hydrogelen. Med detta uttrycks $\hat{u}_{i-1,j,k}$ i termer av q och $\hat{u}_{i+1,j,k}$ vilket möjliggör att beräkna $\hat{u}_{0,j,k}$ genom ekvation (1). Notera att resterande randvärden är konstanta och ej behöver beräknas (de punkter som ligger på kylkanalen är inkluderade).

En enkel finit differenslösare baserad på en *successive over-relaxation*-metod med *relaxation*-faktor [8] $\omega = 1{,}6$ implementerades i beräkningsprogrammet; värdet på ω valdes efter empiriska tester. Nedan följer pseudokod som beskriver den finita differenslösaren:

$\mathcal{I} \leftarrow \{0,\ldots,N_r-1\}$
$\mathcal{J} \leftarrow \{0,\ldots,N_\theta-1\}$
$\mathcal{K} \leftarrow \{0,\ldots,N_z-1\}$
Skapa $\hat{\boldsymbol{u}} = (\hat{u}_{ijk})$ med alla element lika med noll
Ge randvärden till randpunkter och punkter på kylkanalen
Definiera hjälpfunktionen $a(r) := -2/h_r^2 - 2/(r^2h_t^2) - 2/h_z^2$
när $\max|\hat{\boldsymbol{u}}| > \text{TOL}$ **gör**
  för alla $i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K}$ **gör**
    om (i,j,k) ligger på en rand eller på kylkanalen **då**
      Skippa denna iteration
    avsluta om
    Låt r vara radien i meter som motsvarar punkten (i,j,k)
    $\hat{u}_{ijk} \leftarrow (1-\omega)\hat{u}_{ijk} + \omega/a(r) \cdot \Big($
      $- \big(-1/(2rh_r) + 1/h_r^2\big) \cdot \hat{u}_{i-1,j,k}$
      $- \big(1/(2rh_r) + 1/h_r^2\big) \cdot \hat{u}_{i+1,j,k}$
      $- 1/(r^2h_t^2) \cdot \hat{u}_{i,j-1,k}$
      $- 1/(r^2h_t^2) \cdot \hat{u}_{i,j+1,k}$
      $- 1/h_z^2 \cdot \hat{u}_{i,j,k-1}$
      $- 1/h_z^2 \cdot \hat{u}_{i,j,k+1}\Big)$
  avsluta för
avsluta när

Notera att randen med Neumannvillkor behandlas på liknande sätt, som beskrivet ovan.

2.2 Avancerad modell

Projektets andra steg var att konstruera en mer avancerad modell. Till skillnad från den enkla modellen tog denna modell även hänsyn till halsens insida; värmeledningsekvationens domän utökades till både hydrogel och hals, dessutom infördes en värmekälla (det vill säga en inhomogen term $f(r,\theta,z)$ i värmeledningsekvationen) i halsen. Syftet med detta var att kunna importera simulerad effektupptagning och använda denna som värmekälla. På så sätt möjliggjordes en optimering av kylkanalernas placering med hänsyn till en mer verklig värmefördelning (istället för det tidigare homogena värmeflödet). I denna modell fanns även två kylkanaler istället för en, och randvillkoret på dessa kanaler byttes ut från Dirichletvillkor till Neumannvillkor med ett positivt flöde längs normalen som pekar ut från hydrogelen; detta gjordes för att approximera hur flödande kallt vatten tar upp värme från hydrogelen.

I denna modell låter vi $\mathcal{I}$ beteckna mängden av alla index till alla noder. Notera att båda kylakanaler består av lika många noder, varför det finns två noder för varje index $i \in \mathcal{I}$. Till optimeringsproblemet lades de nya bivillkoren $h_i^1 \leq h_i^2 - d \; \forall i \in \mathcal{I}$, där h_i^1 är höjden av nod nummer $i \in \mathcal{I}$ på den nedre kylkanalen, h_i^2 är höjden av nod nummer i på den övre kylkanalen och d är den minsta tillåtna höjdskillnaden mellan två noder på samma vertikala linje. Notera att båda kylkanaler består av lika många noder (nämligen $|\mathcal{I}|$) placerade vid samma θ-koordinater. Dessa bivillkor säkerställer att kylkanalerna inte korsar varandra, under antagandet att de ligger på samma radie. Det nya optimeringsproblemet blev då att

$$\begin{aligned}
\text{minimera} \quad & \frac{1}{|\Gamma_1|} \int_{\Gamma_1} u \, dS & & & \text{(2a)} \\
\text{under} \quad & r_{min} \leq r_i^1, r_i^2 \leq r_{max} & & i \in \mathcal{I}, & \text{(2b)} \\
& h_{min} \leq h_i^1, h_i^2 \leq h_{max} & & i \in \mathcal{I}, & \text{(2c)} \\
& h_i^1 \leq h_i^2 - d & & i \in \mathcal{I}, & \text{(2d)}
\end{aligned}$$

där $\mathcal{I}$ och u är som i den enkla modellen och Γ_1 är planet som motsvarar patientens hud. Optimeringsalgoritmen uppdaterades för att projicera in i den nya tillåtna mängden.

2.3 Parametrar för optimering

Följande parametrar används genomgående i resten av rapporten; optimeringen görs med tio punkter för den enkla modellen, åtta punkter för den avancerade modellen och tolv punkter i laborationen. Vi låter (N_r, N_θ, N_z) vara $(10, 40, 20)$, $(10, 20, 10)$ och $(12, 30, 10)$ i den enkla modellen, laborationen och den avancerade modellen respektive. För optimering med den avancerade modellen användes även ett effektfält simulerat i simuleringsmjukvaran CST som den inhomogena termen f i värmeledningsekvationen.

3 Verifiering av algoritm

För att säkerställa att optimeringsalgoritmen i ovanstående kapitel fungerar verifierades de optimerade geometrierna, vilket beskrivs i detta kapitel. Verifieringen skedde initialt i en simuleringsmiljö genom rit- och simuleringsprogrammet Solidworks. Därefter utfördes även en experimentell verifikation i verklig miljö med en fysisk modell av kylsystemet efter optimering.

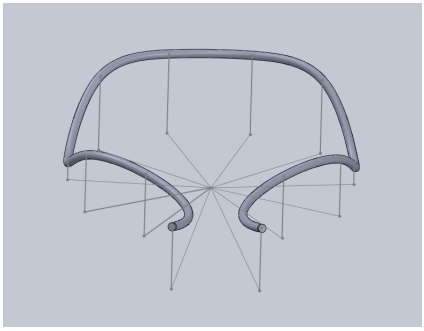
3.1 Numerisk verifikation

Verifieringen av optimeringsalgoritmen utfördes både för den förenklade modellen och för den avancerade.

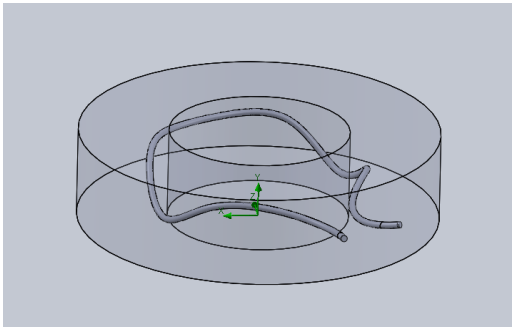
3.1.1 Förenklad modell

För att verifiera implementeringen av den enkla modellen behövde 3D-modellen i simuleringsprogrammet efterlikna den i föregående kapitel. En kylkanal skapades därför utifrån den optimala lösningen framtagen med hjälp av optimeringsmetoden i kapitel 2.1.

Kylkanalen har diametern 0,5 cm och väggtjockleken 0,1 cm. Avståndet från halsens mittpunkt till kylkanalen är 12 cm, se figur 3. Hydrogelen modellerades, likt tidigare, som en ihålig cylinder med innerradie 10 cm, ytterradie 20 cm och höjd 10 cm. Kylkanalerna placerades sedan i hydrogelen så att de båda cirklarnas centrum sammanfaller, se figur 3.



(a) 3D-modell av kylkanalen utifrån de optimerade noderna.



(b) 3D-modell av kylkanalens position inuti hydrogelen.

Figur 3: 3D-modell av resultaten från den enkla modellen; kylkanalen samt hydrogelen och kylkanalen.

Efter att 3D-modellen skapats analyserades denna. De parametrar som valdes vid simuleringen kan ses i tabell 1.

Tabell 1: Parametrar vid simulering.

Parameter	**Värde**
Hydrogel: temperatur ovan- och undersida	20 °C
Hydrogel: temperatur yttre mantelytan	5 °C
Konstant temperatur på kylkanal	5 °C
Värmekälla	$100\,\mathrm{W/m^2}$
Värmeöverföringskoefficient	$10\,\mathrm{W/\left(m^2 \cdot K\right)}$
Omgivande lufttryck	101 325 Pa

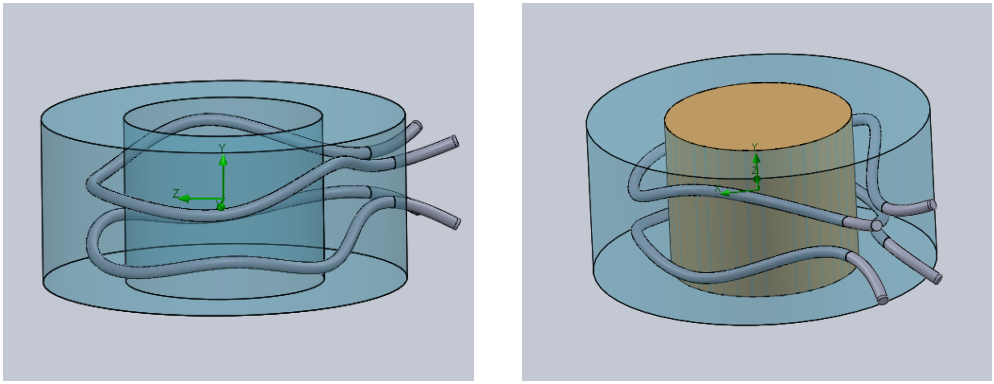
Randvillkoren för hydrogelen behölls från modellen i beräkningsprogrammet. Som tidigare modellerades kylkanalen med en konstant temperatur (5 °C) och värmekällan som motsvarar huden valdes till $100\,\mathrm{W/m^2}$.

I tidigare studie har materialet nylon 6 används till kylkanalerna [3]. Därför valdes nylon 6 även i denna studie. I samma studie modellerades hydrogelen som vatten, vilket även gjordes i denna studie.

3.1.2 Avancerad modell

Även den avancerade modellen verifierades i simuleringsprogrammet. På samma sätt som i avsnitt 3.1.1 skapades modellen för att efterlikna den i avsnitt 2.2. Tillskillnad från föregående avsnitt modellerades nu kylkanalerna med ytterdiametern 0,5 cm och innerdiametern 0,3 cm. Avståndet från halsens mittpunkt till kylkanalerna sattes till 8 cm och hydrogelen modellerades som en ihålig cylinder med innerradie 6 cm, ytterradie 11 cm och höjd 10 cm, se figur 4a.

För den avancerade modellen skapas även en ”hals” i form av en homogen cylinder med radie 6 cm och höjd 10 cm, se figur 4b. Likt den avancerade modellen i kapitel 2.2 användes filen för absorberad effekt som värmekälla i halsen.



(a) 3D-modell av kylkanaler utifrån de optimerade noderna tillsammans med hydrogelen.

(b) 3D-modell av kylkanaler och hydrogel, nu tillsammans med hals.

Figur 4: 3D-modell av kylkanaler, hydrogel och hals, modellerade för att efterlikna optimeringen från den avancerade modellen.

Därefter utfördes en simulering för att verifiera resultaten för den avancerade modellen. I tabell 2 redovisas de parametrar som användes. Massflödet i båda kylkanalerna hade samma riktning.

Tabell 2: Parametrar vid simulering.

Parameter	**Värde**
Hydrogel: temperatur ovan- och undersida	20 °C
Hydrogel: temperatur yttre mantelytan	5 °C
Konstant kyleffekt	-200 $\mathrm{W/m^2}$
Värmekälla	absorberad effekt från simulering
Vattentemperatur	
Värmeöverföringskoefficient	$10\,\mathrm{W/\left(m^2 \cdot K\right)}$
Omgivande lufttryck	101 325 Pa
Halsens under- och översida	37 °C

3.2 Experimentell verifikation

Beroende på vart tumören är placerad i halsen kan det vara viktigt att kunna kyla vissa områden mer än andra under en cancerbehandling. För att verifiera att optimeringsalgoritmen kan uppnå en sådan inhomogen kylning, utfördes ett experiment i verklig miljö. Experimentet baserades på en optimering av den avancerade modellen. I experimentet används kylanordning med en hydrogel, kylkanal, vattenpump och mätutrustning.

3.2.1 Förberedelser

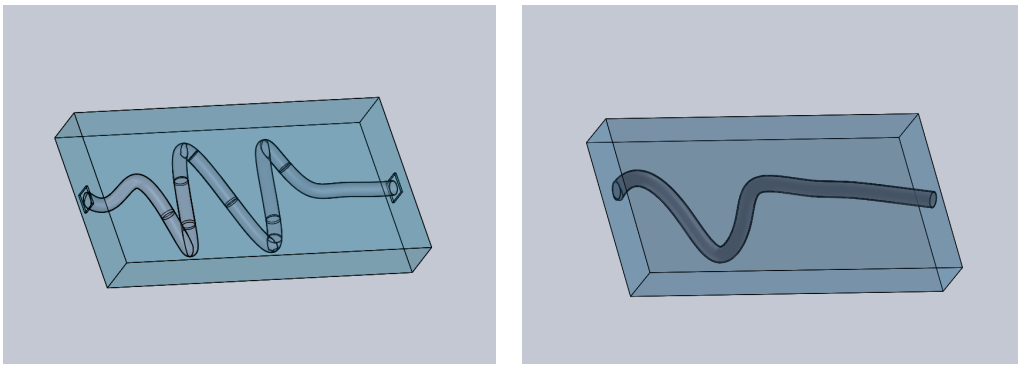
Initialt modifierades den avancerade modellen för att appliceras på ett enkelt test. Optimeringsmodellen använde nu endast en kylkanal, ingen värmekälla och målfunktionen var nu medeltemperatur på en liten kvadratisk del av hydrogelens yta istället för på hela ytan. Som kontroll användes även en rak kylkanal som motsvarar en jämn kylning över hydrogelens ovansidan.

Utifrån de optimerade noderna tillverkades fysiska 3D-modeller bestående av hydrogel och kylkanaler i plast. Hydrogelen göts kring kylkanalen; kylkanalen som användes valdes godtyckligt och

hade ytterdiametern 1,5 cm och innerdiametern 1,3 cm. Under gjutningen användes en rektangulär behållare med bredd 15,7 cm, höjd 4,85 cm och längd 33,7 cm. Behållaren saknade önskvärd design på kortsidorna; därför gjordes dessa i ritprogrammet och 3D-printades i materialet resin i en Form 2-skrivare, se figur 5.

Figur 5: De två kortsidor som ritades och skrevs ut för att användas i experimentet.

För att uppnå önskad form på kylkanalerna spändes plastkanalen upp med tunna trådar. Här uppstod dock ett praktiskt problem med geometrin för den inhomogena kylningen. Formen var för komplex för att det praktiskt skulle vara möjligt att kunna inrätta kanalen i formen för testet. Därför gjordes en förenkling av geometrin, se figur 6, som medförde mindre och mjukare kurvor. Därefter fylldes kylkanalerna med socker, för att undvika att kylkanalerna kollapsade på grund av hydrogelens tyngd. Sockret spolades sedan ut med vatten efter att hydrogelen stelnat.



(a) Geometrin från optimeringen med inhomogen kylning.

(b) Den förenklade varianten av geometrin med inhomogen kylning.

Figur 6: Den komplexa geometrin för experimentet samt den förenklade geometrin.

Hydrogelen blandades utefter följande recept:

- 2970 g vatten,
- 10,5 g LBG,
- 10,5 g xanthan, samt
- 9,0 g agar.

Vattnet värmdes upp till ungefär 80 °C varefter LBG och xanthan tillsattes under omrörning. När blandningen nått 90 °C rördes även agar i. Efter att allt löst upp sig tillsattes den mängd vatten som avdunstat, för att få rätt proportioner. Blandningen hälldes därefter i formen där kylkanalen redan var fastspänd och ställdes sedan in i kylskåp för att stelna.

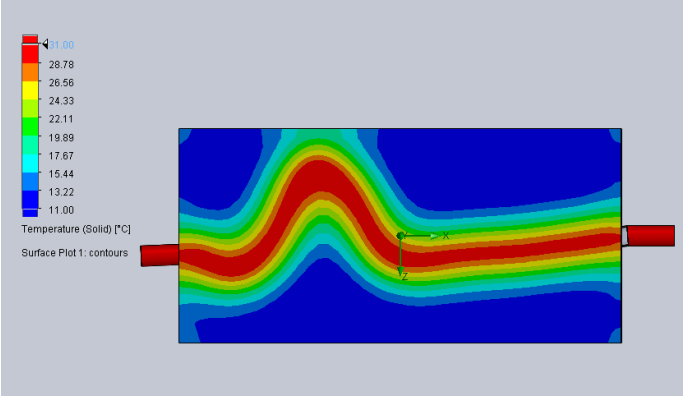
Då hydrogelerna förvarats i kylskåp på grund av stelningen bedömdes det fördelaktigt att inte värma upp hydrogelerna till rumstemperatur för att därefter kyla ner dem; istället utfördes testerna genom uppvärmning av redan kylda hydrogeler med en vattentemperatur på 40 °C.

3.2.2 Förväntade resultat

För att verifiera resultaten från experimenten utfördes tester i simuleringsprogrammet. Simuleringarna kördes, till skillnad från tidigare, över en tid på 40 minuter istället för i tidsoberoende förhållanden. Parametrar för modellen visas i tabell 3 och resultaten återfinns i figur 7 och figur 8.

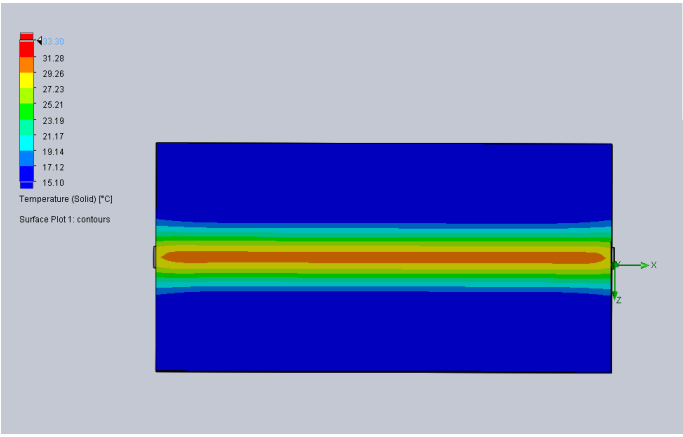
Tabell 3: Parametrar vid simulering.

Parameter	Värde
Hydrogel: initial temperatur	5 °C
Vattentemperatur vid inflöde	40 °C
Värmeöverföringskoefficient	$10\,\mathrm{W}/\left(\mathrm{m}^2 \cdot \mathrm{K}\right)$
Omgivande lufttryck	101 325 Pa



Figur 7: Det förväntade resultatet för kylkanalen med inhomogen kylning.

Medeltemperaturen på ytan för hydrogelen då kylkanalerna är komplext formade för en inhhomogen kylning är, enligt simuleringar, 16,3 °C.

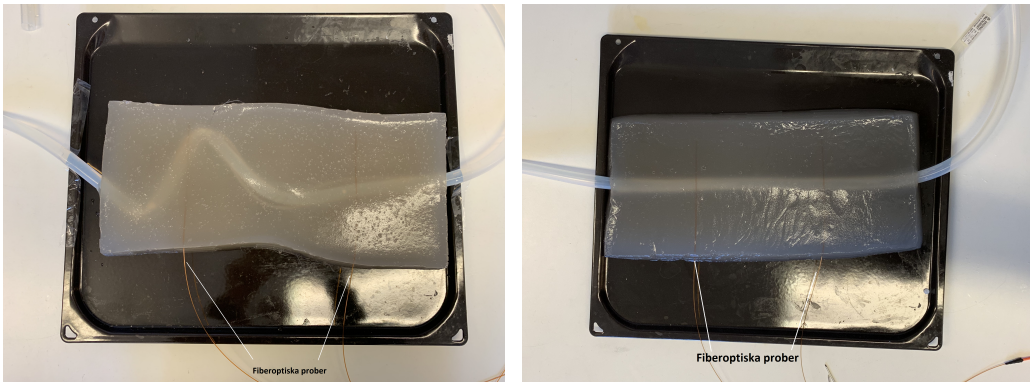


Figur 8: Det förväntade resultatet för den raka kylkanalen.

Hydrogelens yta har i fallet med rak kylkanal medeltemperaturen 16,14 °C, enligt simuleringar.

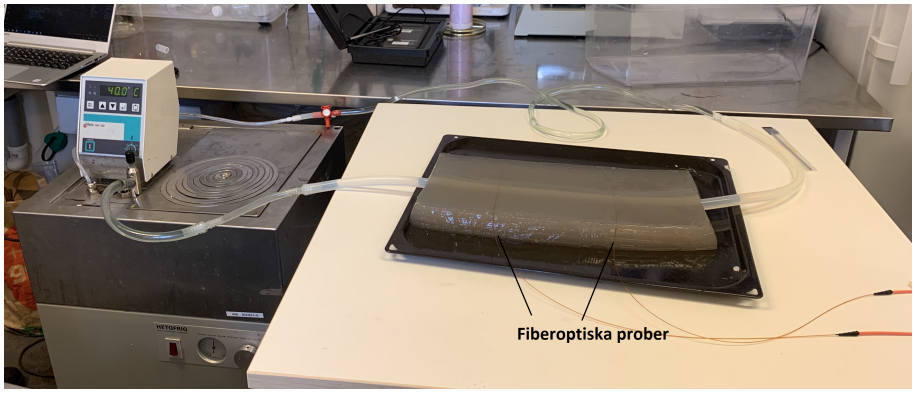
3.2.3 Utförande

Båda testerna utfördes på samma sätt; den gjutna hydrogelen, som förvarats i kylskåp, placerades på en metallskiva. Ett slutet flöde skapades genom att koppla ihop en termostat och pump med kylkanalen. För att mäta temperaturutbredningen under experimentet fördes fiberoptiska prober med värmesensorer in i hydrogelen; ungefär 0,5 cm från hydrogelens översida, se figur 9. Testernas uppställning kan ses i figur 10. Innan och efter testerna användes även en värmekamera för att kontrollera temperaturen.



(a) Geometrin för inhomogen kylning. (b) Geometrin för jämn kylning.

Figur 9: De båda kylkanalerna som skulle testas.



Figur 10: Uppställningen för testet med en pump och termostat till vänster, prober i hydrogelen samt ett slutet kretslopp för vattnet.

Testerna genomfördes under 40 minuter med maximalt vattenflöde.

4 Resultat

I det här kapitlet redovisas resultaten från optimeringsalgoritmen. Vidare presenteras resultatet för verifieringen av optimeringsalgoritmen; både numeriskt och experimentellt.

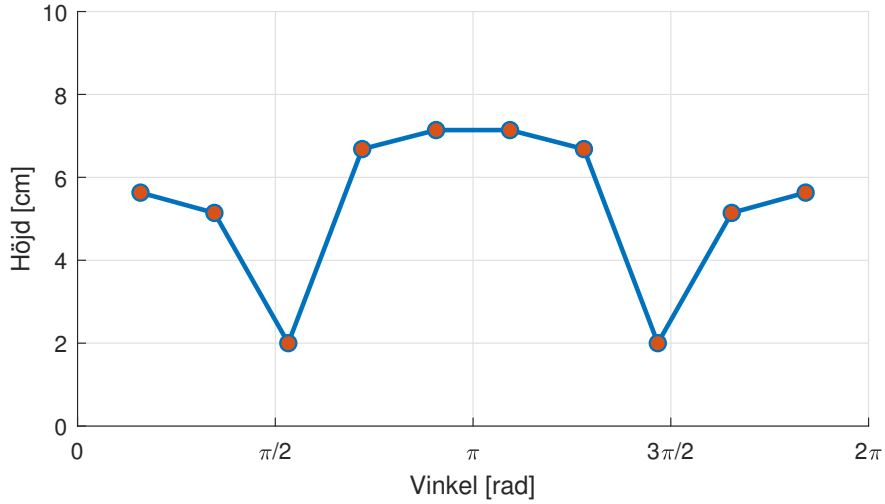
4.1 Numerisk verifikation - Förenklad modell

Optimeringsalgoritmen resulterade i positioner för tio noder, vilka redovisas i tabell 4, alla med konstant radie på 12 cm (avståndet från halsens mittpunkt). Kylkanalens placering utifrån dessa

noder åskådliggörs i figur 11.

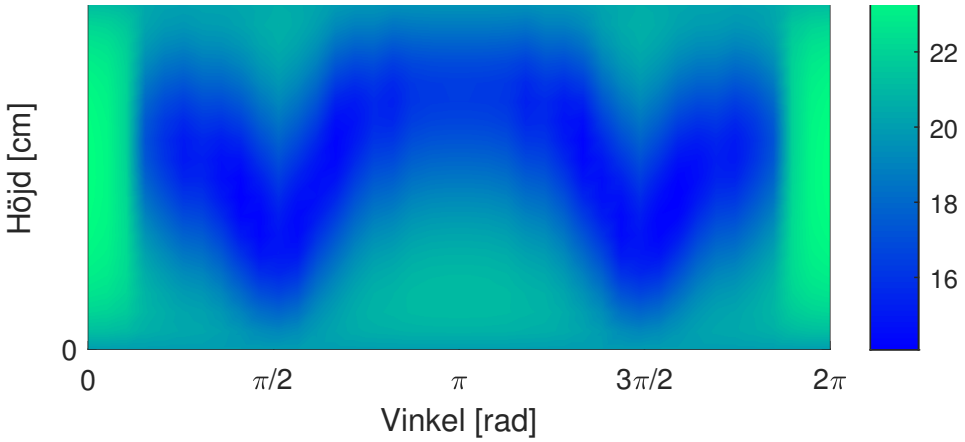
Tabell 4: Höjd [m] från botten och vinkel [rad] för de tio noderna.

Nod	1	2	3	4	5	6	7	8	9	10
Höjd från botten [m]	0,0563	0,0514	0,0200	0,0668	0,0714	0,0714	0,0668	0,0200	0,0514	0,0563
Vinkel [rad]	0,5000	1,0870	1,6740	2,2611	2,8481	3,4351	4,0221	4,6091	5,1962	5,7832



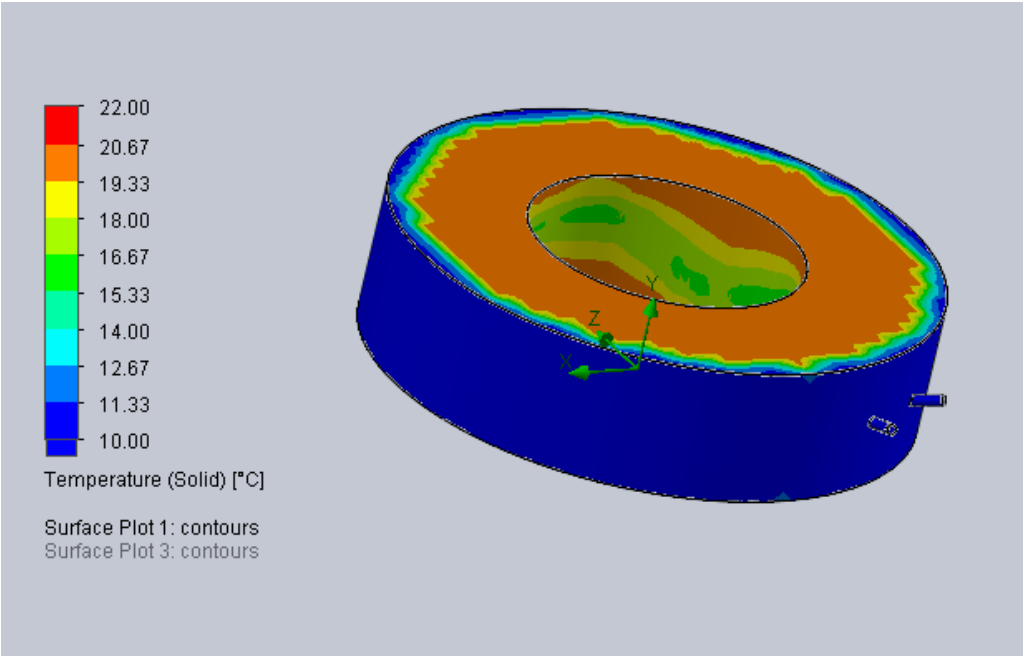
Figur 11: Kylkanalens placering optimerad enligt den enkla modellen. Varje nod på kurvan ligger på avståndet 2 cm från patientens hud i radiell led.

Kylningen som erhölls med kylkanalen inuti hydrogelen visas i figur 12. Huden är 10 cm från halsens centrum, där överstiger temperaturen inte 23,73 °C och medeltemperaturen är 14,64 °C.



Figur 12: Temperaturen som kylkanalen gav upphov till på ytan mot huden. De varma ytorna till vänster och höger uppstår då kylkanalen inte sträcker sig hela vägen ut till kanterna.

De resulterande noderna från optimeringsalgoritmen användes sedan för att modellera och simulera kylkanaler i simuleringsprogrammet, resultaten kan ses i figur 13.



Figur 13: Värmeutveckling vid stationärt tillstånd i hydrogelen.

I figur 13 ses att insidan av hydrogelen kyls ner, särskilt längs med kylkanalen. Insidan, som är mot patientens hud överstiger inte 21,45 °C och har medeltemperaturen 18,70 °C.

4.2 Numerisk verifikation - Avancerad modell

Optimeringsalgoritmen resulterade i positioner för åtta noder på vardera kylkanal, vilka redovisas i tabell 5 och tabell 6, alla med konstant radie på 8 cm (avståndet från halsens mittpunkt). Kylkanalernas placering utifrån dessa noder åskådliggörs i figur 14.

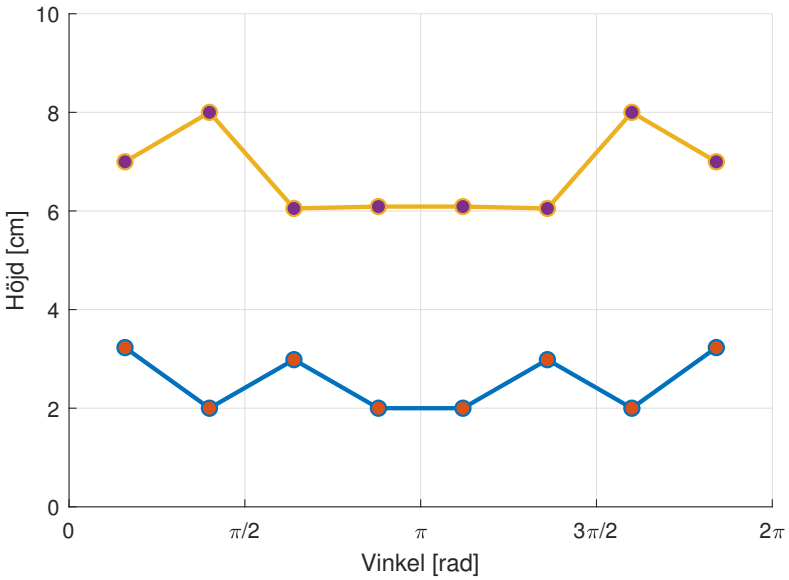
Tabell 5: Höjd [m] från botten för kanal 1.

Nod	1	2	3	4	5	6	7	8
Höjd från botten [m]	0,0323	0,0200	0,0298	0,0200	0,0200	0,0298	0,0200	0,0323

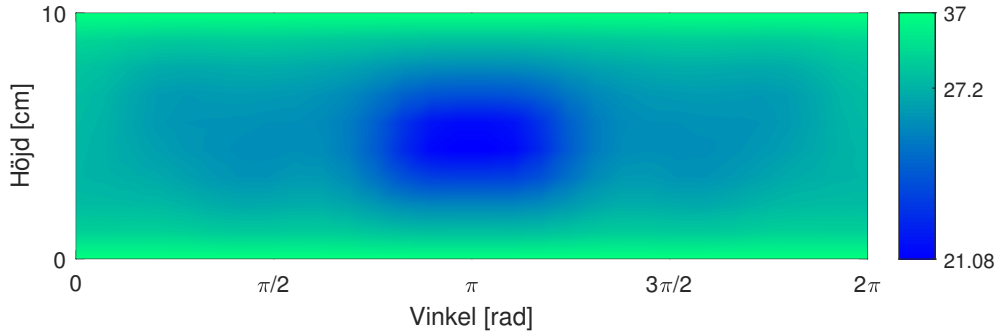
Tabell 6: Höjd [m] från botten för kanal 2.

Nod	1	2	3	4	5	6	7	8
Höjd från botten [m]	0,0700	0,0800	0,0605	0,0609	0,0609	0,0605	0,0800	0,0700

Kylningen som erhölls med kylkanalerna visas i figur 15. Huden är 6 cm från halsens centrum och på huden överstiger inte temperaturen 37 °C, vilket kan ses i figuren. Denna maximala temperatur uppstår från Dirichletvillkoret på ovan- och undersidan. Medeltemperaturen på huden är 28,50 °C och minimumtemperaturen är 21,08 °C.

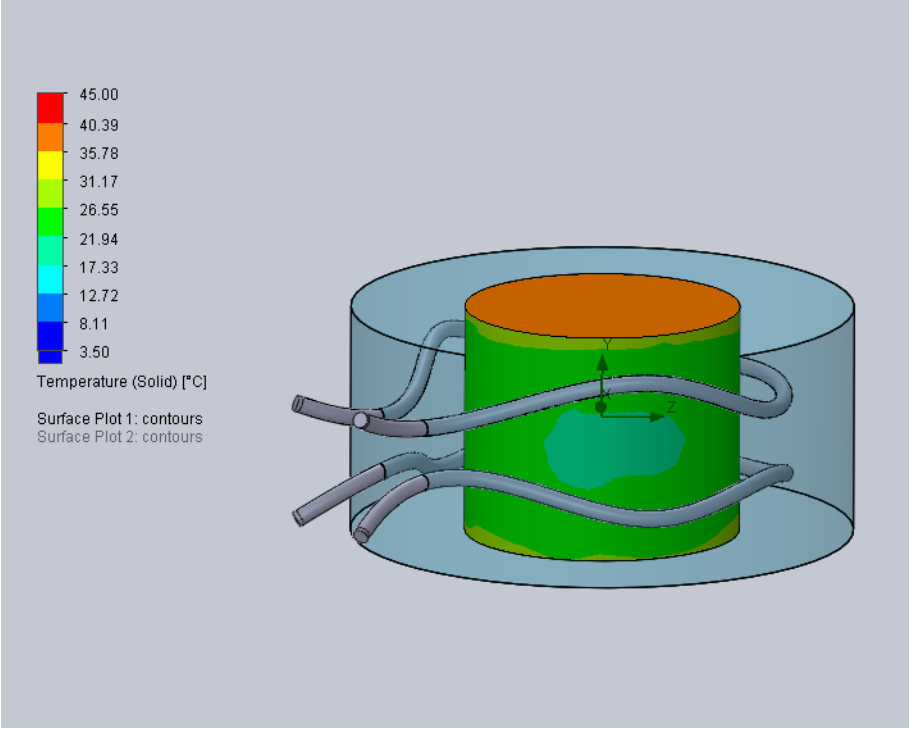


Figur 14: Kylkanalernas placering för den avancerade modellen. Varje nod på kurvan ligger på avståndet 2 cm från patientens hud i radiell led.



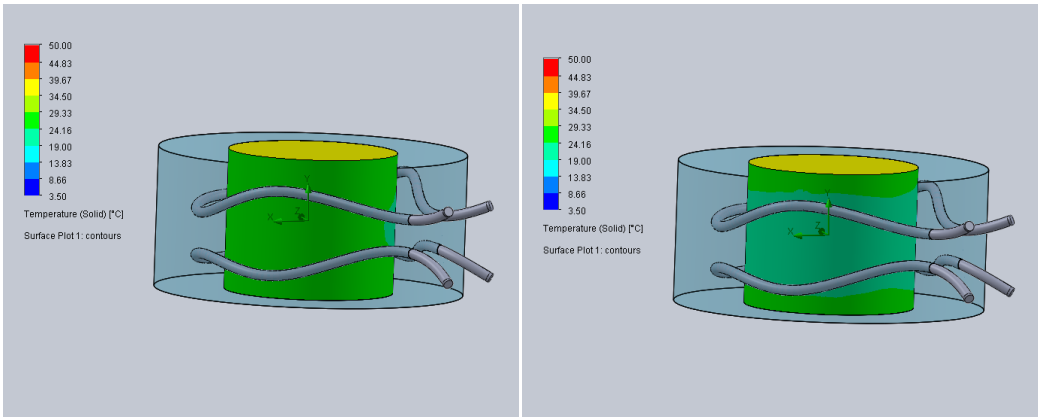
Figur 15: Temperaturen som kylkanalerna gav upphov till på hudens yta.

De resulterande noderna från optimeringsalgoritmen användes för att modellera och simulera kylkanalerna i simuleringsprogrammet. Initialt gjordes en simulering då kanalerna hade ett konstant Neumannvillkor, liksom tidigare optimeringsberäkningar, se figur 16. Patientens hud överstiger inte 39,18 °C och har medeltemperaturen 31,84 °C.



Figur 16: Värmeutveckling för hela modell vid stationärt tillstånd.

Simuleringen utfördes även med ett vattenflöde, till skillnad från konstant kyleffekt (Neumannvillkor), för att se om modellen ger en märkbar kylning, se figur 17. Utan vattenflödet fick huden en medeltemperatur, 32,92 °C och maxtemperatur, 40,28 °C, se figur 17b. Simuleringen med ett flöde på 0,8 kg/s resulterade i en medeltemperatur, 30,70 °C och maxtemperatur, 38,10 °C i huden, se figur 17a.



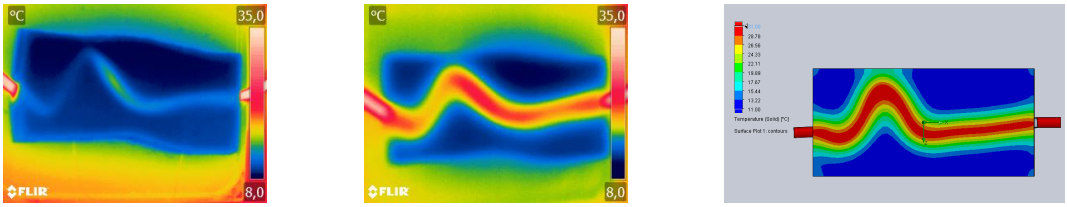
(a) Värmeutveckling i hals utan vattenflöde i kylkanalerna.

(b) Värmeutveckling i hals med vattenflöde i kylkanalerna.

Figur 17: Värmeutveckling för hals vid stationärt tillstånd.

4.3 Experimentell verifikation

I figur 18 och 19 presenteras före- och efterbilder, tagna med värmekamera, på de båda hydrogelerna vid experimentet. I dessa figurer syns att temperaturen ökat i kylkanalerna samt i kylkanalernas omgivning. I figur 18b syns att det blev en inhomogen uppvärmning med den böjda kylkanalen.

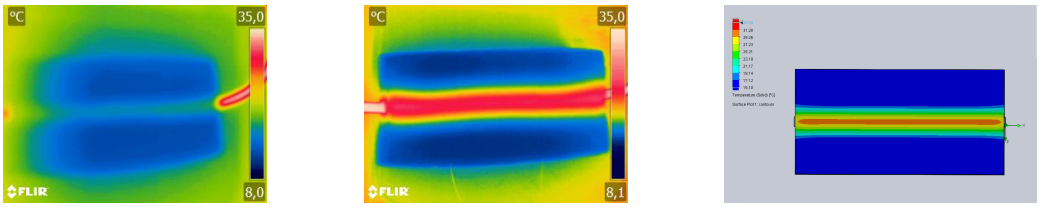


(a) Just när vattnet börjat flöda igenom.

(b) Efter att vattnet flödat igenom i 40 minuter.

(c) Det förväntade resultatet från simuleringsprogrammet.

Figur 18: Den komplext formade kylkanalen.



(a) Just när vattnet börjat flöda igenom.

(b) Efter att vattnet flödat igenom i 40 minuter.

(c) Det förväntade resultatet från simuleringsprogrammet.

Figur 19: Den rakt formade kylkanalen.

Att temperaturen ökat efter att vattnet strömmat igenom kylkanalerna stöds av datan som erhölls av proberna. Dessvärre sparades endast datan för de sista tio minuterna, något som gjorde att det inte gick att studera värmeutvecklingen.

5 Diskussion och slutsats

Projektet har resulterat i en optimeringsalgoritm för kylning av hydrogelen. Optimeringsalgoritmen har verifierats i flera steg där resultaten pekar på att algoritmen fungerar med avseende på projektets omfattning.

5.1 Optimeringsalgoritm

En självklar fråga kring ett minimeringsproblem är huruvida målfunktionalen är konvex eller inte. Den diskreta versionen av målfunktionalen i den enkla modellen är inte konvex i sin tillåtna mängd; om $\boldsymbol{c}_r = (c_r^i)$ betecknar vektorn av alla noders radier (i cm) på kylkanalen så ger $\boldsymbol{c}_r = (2, 8)$ ett målfunktionsvärde på 20,34 °C, $\boldsymbol{c}_r = (8, 2)$ ger 20,28 °C medan $\boldsymbol{c}_r = (5, 5)$ ger ett målfunktionsvärde på 20,45 °C. Men $(5, 5)$ är en linjärkombination av $(2, 8)$ och $(8, 2)$, så alltså är detta ett motexempel mot att målfunktionen är konvex. Detta argument kan trivialt utökas till ett godtyckligt antal noder.

Den diskreta versionen av målfunktionalen i den avancerade modellen är inte konvex i $\boldsymbol{R}^{4|\mathcal{I}|}$ där $|\mathcal{I}|$ är antalet noder på varje kylkanal; ett liknande argument som för den enkla modellen visar detta. Detta argument gäller även under alla bivillkor förutom 2d — när vi tar hänsyn till detta lyckas vi inte hitta ett motexempel mot konvexitet.

Huruvida målfunktionen i den avancerade modellen är konvex eller inte är alltså fortfarande en öppen fråga, som kan och borde undersökas närmre. Ifall den inte är konvex, bör metoder för att försöka hitta ett globalt optimum undersökas och läggas till i vår metod.

Inget bivillkor fanns för att ta hänsyn till den totala längden av kylkanalerna eller deras krökning. Att låta kylkanalerna vara styckvis linjära medför problemet att ett stort antal noder skulle kunna

skapa väldigt små vinklar i hörnen i noderna, vilket är orimligt att använda som en fysisk kylkanal på grund av de verkliga kylkanalernas begränsade böjningsförmåga. Istället vore det lämpligt att dels begränsa kylkanalernas totala längd, dels låta kurvorna vara tillräckligt glatta.

Den utvecklade optimeringsalgoritmen är väldigt beräkningsintensiv och vi har i alla tester använt en väldigt låg upplösning i rutnätet för de finita differenserna och stora toleranser (runt 0,01) för att bedöma konvergens hos både den finita differenslösaren och gradient projection-algoritmen. Lägre toleranser och högre upplösning är önskvärt för en såpass viktig tillämpning, om man vill garantera med stor säkerhet att den punkt som nås är tillräckligt optimal, men detta skulle kräva mer effektiva numeriska metoder. För den finita differenslösaren skulle feluppskattningar och möjligtvis en adaptiv algoritm vara till hjälp för att säkerställa konvergens av lösningen.

5.2 Verifiering av optimeringsalgoritm

Verifieringen av den enkla modellen tyder på att optimeringsalgoritmen fungerar. Det skiljer några få grader mellan resultaten i beräkningsprogrammet och resultatet från simuleringen. Dessa skillnader kan bero på att simuleringsprogrammet använder fler materialkoefficienter än beräkningsprogrammet.

Verifieringen av den avancerade modellen indikerar också att optimeringsalgoritmen fungerar väl. Temperaturskillnaden mellan beräkningen och verifieringen kan även här delvis bero på materialkoefficienter som angivits under simuleringar. I det beräknade resultatet av optimeringsalgoritmen är det svårt att utläsa en omkrets på kylkanalerna. Storleken av kylkanalerna i simuleringsprogrammet är därför enbart uppskattat till ett lämpligt värde. Detta medför att kyleffekten mellan beräkningar och simuleringar varierar eftersom kyleffekten är direkt kopplad till kanalernas yta.

Syftet med experimentet var att kunna verifiera om optimeringsalgoritmen kunde fylla funktionen att kyla specifika områden. I och med att det inte var praktiskt möjligt att implementera kylkanalen för den inhomogena kylningen är det svårt att dra slutsatsen om experimentet fungerat. Resultatet från värmekameran tyder dock på att värmeutvecklingen sker på en större area med optimeringsalgoriten, till skillnad från en rak kanal.

Datan från de fiberoptiska proberna som användes under experimentet sparades inte i sin helhet och kunde därför inte analyseras. Det medförde att det var svårt att se värmeutvecklingen under hela behandlingstiden då bilder med värmekamera enbart togs i början och slutet av testerna.

Gjutningen av kanalen till experimentet blev inte helt optimal. Kanalen lossnade lite från hydrogelen och bildade luftfickor inuti hydrogelen, vilket påverkar kylningen negativt.

5.3 Samhälleliga och etiska aspekter

För att undvika utsläppet av mikroplaster i avloppet, studerades kylkanalerna noga innan de fördes in i formen då trasiga kylkanaler lättare släpper ifrån sig mikroplaster. Efter experimentet togs hydrogel och kylkanaler om hand. Delarna försökte återanvändas i största möjliga mån, och det som blev över sopsorterades i den grad det gick. Slutligen slängdes det som återstod i brännbart, för att undvika spridning av mikroplaster i avloppet. Det vill dock nämnas att denna typ av mikroplaster används i betydligt högre utsträckning inom industrin.

5.4 Vidareutveckling och rekommendationer

Resultaten från beräkningar och simuleringar visar på att de maximala temperaturena i halsen inte överstiger 42 °C, vilket tyder på att den matematiska optimeringensalgoritmen skiljer sig en del från verkliga behandlingsförhållanden, då temperaturen med det givna effektfältet borde uppnå 43-45 °C. Detta kan delvis bero på att optimeringsalgoritmen beräknar maximala temperaturen utifrån ett jämviktsläge där temperaturen inte ändras med tid. Detta innebär att optimeringsalgoritmen inte nödvändigtvis beräknar den maximala temperaturen som skulle kunna uppstå under

hela behandlingstiden. Detta medför att en utveckling av algoritmen borde göras baserad på en tidsberoende värmeledning.

Att temperaturen skiljer mellan algoritmen och en verklig behandling beror även på de förenklade randvillkoren på hydrogelen. Dessa randvillkor borde analyseras noggrannare för mer precisa villkor.

I optimeringsalgoritmen för den avancerade modellen användes absorberad effekt i halsen som värmekälla. I verkligheten tar även hydrogelen upp värme, vilket borde inkluderas i värmekällan som används för optimeringen.

I en utveckling av optimeringsalgoritmen är det intressant att analysera huruvida EM-fältet skulle påverkas av den implementering av kylsystemet som gjorts, till exempel genom att EM-fältet inte får interferens där det är önskat. Att utföra en EM-fältssimulering i varje steg av optimeringsalgoritmen skulle formellt göra metoden väldigt beräkningsintensiv, så numeriska metoder och lågintensiva approximationer är av stort intresse för framtida utveckling.

5.5 Slutsats och framtidsutsikt

Med en vidareutvecklad optimeringsalgoritm finns goda förutsättningar för att algoritmen kan implementeras i cancerbehandling med halsapplikatorn. Halsapplikatorn kan i sin tur öka effektiviteten på cancerbehandling, vilket medför att man kan rädda fler liv. Om halsapplikatorn utvecklas väl efter givna rekommendationer är målet att minska dagens behandlingsdoser av strålning och kemoterapi, då hypertermibehandling ökar dessa behandlingsmetoders effektivitet. Om mängden strålning och kemoterapi minskar, minskar även de negativa biverkningarna av dessa.

Referenser

[1] N. Cihoric, A. Tsikkinis, G. van Rhoon, H. Crezee, D. M. Aebersold, S. Bodis, M. Beck, J. Nadobny, V. Budach, P. Wust och P. Ghadjar, "Hyperthermia-related clinical trials on cancer treatment within the clinicaltrials.gov registry", *International Journal of Hyperthermia*, årg. 31, nr 6, s. 609–614, 2015. URL: https://doi.org/10.3109/02656736.2015.1040471.

[2] H. D. Trefna och A. Ström, "Hydrogels as a water bolus during hyperthermia treatment", Chalmers tekniska högskola, 2019, Accepterat manuskript.

[3] B. Elling, "Bringing an h&n microwave hyperthermia applicator from laboratory to clinics", ETH och Chalmers tekniska högskola, 2018.

[4] H. D. Trefna, *Microwave hyperthermia: A quick intro*, Personlig kommunikation, 2019.

[5] C. Nordling och J. Österman, *Physics Handbook for Science and Engineering*. Studentlitteratur, 2006.

[6] L. Ekman, S. Hannoun, B. Lönn och T. Wegnelius, "Hydrogeler som vattenbolus för mikrovågshypertermi", Chalmers tekniska högskola, 2016.

[7] *Mikroplaster – källor och uppströms arbete samt möjligheter till rening vid kommunala reningsverk*, 2016. URL: http://www.svensktvatten.se/globalassets/avlopp-och-miljo/uppstromsarbete-och-kretslopp/mikroplaster-i-miljon/mikroplaster-kallor-uppstromsarbete-och-reningsteknik-vid-kommunala-reningsverk.pdf.

[8] N. Black och S. Moore, *Successive overrelaxation method.* URL: http://mathworld.wolfram.com/SuccessiveOverrelaxationMethod.html.

A Matlab-kod avancerad modell

A.1 Huvudskript

```
% ------ INPUT PARAMETERS ------
% Note: all physical quantities are expressed in SI units.
% Domain dimensions:
r0 = 0.06;         % radius of neck
rLen = r0 + 0.05;  % radius of neck + hydrogel
% r0 = 0.1;
% rLen = 0.2;
zLen = 0.1;        % height of neck and hydrogel
% Resolution of the grid for the discrete heat equation:
Nr = 12;  % Number of points in radius direction
Nt = 30;  % Number of points in angular direction
Nz = 10;  % Number of points in height direction
% Amount of heat flux (W/m^2) on tube boundaries:
fluxTube = -200;
% Optimization constraints:
minTubeDistance = 0.01;  % Minimum distance between the two tubes in
                         % height direction. Needed if both tubes lie on
                         % the same radius to prevent them from crossing.
minR = r0 + 0.02;    % Minimum radius (from center) of tube nodes
maxR = rLen - 0.02;  % Maximum radius (from center) of tube nodes
minH = 0.02;         % Minimum height of tube nodes
maxH = zLen - 0.02;  % Maximum height of tube nodes
% Other settings:
optimizeRadii = false;  % Set optimizeRadii = true if you want to include
                        % node radii as variables in the optimization
                        % problem, false otherwise.
plotIterations = true;  % Set plotIterations = true if you want a simple
                        % plot of all variable values after each gradient
                        % projection iteration, false otherwise.
% ------ END OF INPUT PARAMETERS ------

tLen = 2*pi;  % domain length in theta direction
% Step lengths
hr = rLen / (Nr - 1);
ht = tLen / (Nt - 1);
hz = zLen / (Nz - 1);
k = @(r) 0.53 + (r > r0)*0.07;  % thermal conductivity of neck = 0.53,
                                % thermal conductivity of water = 0.6
solverCalls = 0;  % counter to track how many times the solver is called

heatSource = setupHeatSource(Nr, Nt, Nz);
disp(Finished setting up heat source)

% The water tubes are modelled as piecewise linear curves inbetween a
% number of nodes; these nodes have variable heights and radii, and these
% are the variables in the optimization problem* (*see option optimizeRadii
% above). The initial tube curves, c1_0, and c2_0, are defined below.
curveRes = 3;  % the number of nodes on each curve (including endpoints)
c1h = 0.4*zLen*ones(1, curveRes);  % the heights of all nodes on c1_0
c2h = 0.6*zLen*ones(1, curveRes);  % the heights of all nodes on c2_0
c1r = minR*ones(1, curveRes);  % the radii of all nodes on c1_0
c2r = minR*ones(1, curveRes);  % the radii of all nodes on c2_0
% All optimization variables (always including radii regardless of
% optimizeRadii) are collected into a single vector as defined below.
c1_0 = [c1r, c1h];
c2_0 = [c2r, c2h];
c0 = [c1_0, c2_0];

[uMaxOpt, cOpt] = gradProj(@objective, c0);
uOpt = solvePDECyl(cOpt);
c1Opt = cOpt(1:length(cOpt)/2);
c2Opt = cOpt(length(cOpt)/2+1:end);

%% Plot
figure(1)
clf
hold on
[R, T, Z] = ndgrid(1:size(uOpt, 1), 1:size(uOpt, 2), 1:size(uOpt, 3));
S = uOpt(:);
S = 1*(S - min(S) + 100);
plot3DField(R(:), T(:), Z(:), S)
view(-30, 25)
colorbar
%% Plot tubes
c1Coords = interpolateCurve(c1Opt);
c2Coords = interpolateCurve(c2Opt);
scatter3(c1Coords(1, :), c1Coords(2, :), c1Coords(3, :), ...
```

```
        200, [1, 1, 1], 'filled')
scatter3(c2Coords(1, :), c2Coords(2, :), c2Coords(3, :), ...
        200, [0, 0, 0], 'filled')
axis equal
colormap winter
colorbar
```

A.2 Gradient projection-algoritm

```
function [objOpt, cOpt] = gradProj(objective, c0)

% This function performs a gradient projection optimization with the given
% input objective function and initial point c0. The constraints of the
% problem are hard–coded below.

% ------ PARAMETERS ------
TOL_g = 0.0001; % The tolerance which determines whether the gradient of
                % the objective function is small enough to stop.
TOL_p = 0.0001; % The tolerance which determines whether the difference
                % between the previous point and the active point after
                % projection is small enough to stop.
% ------ END OF PARAMETERS ------

optimizeRadii = evalin('base', 'optimizeRadii');
minTubeDistance = evalin('base', 'minTubeDistance');
minR = evalin('base', 'minR');
maxR = evalin('base', 'maxR');
minH = evalin('base', 'minH');
maxH = evalin('base', 'maxH');
doPlot = evalin('base', 'plotIterations');

global hasUserInterrupted
hasUserInterrupted = false;

% If doPlot, the values of each variable are plotted in figure 2 after each
% iteration.
if(doPlot)
    if (optimizeRadii)
        oldPlotVals = c0;
    else
        oldPlotVals = [c0(length(c0)/4+1 : length(c0)/2), ...
                       c0(3*length(c0)/4 + 1 : end)];
    end
    figure(2)
    clf
    hold on
    grid on
    set(gcf, 'KeyPressFcn', @forceQuitFcn)
    drawnow
end

iteration = 0;  % Counter counting the number of iterations
cCurrent = c0;  % Variable containing the current active point
flag = false;   % Flag dermining whether an optimal point is reached
while(~flag)
    iteration = iteration + 1;
    stepLen = 0.005 / iteration;  % The step length used to find the next point
    disp(Starting iteration  + iteration)

    [objVal, grad] = objective(cCurrent, iteration);
    % Stop if gradient is small
    if(max(abs(grad)) < TOL_g)
        flag = true;
        disp('Gradient below tolerance, stopping')
        continue
    end

    cLen = length(c0);
    cNew = cCurrent - stepLen .* grad;  % Step along negative gradient
    c1h = cNew(cLen/4 + 1 : cLen/2);
    c2h = cNew(3*cLen/4 + 1 : end);
    c1r = cNew(1 : cLen/4);
    c2r = cNew(cLen/2 + 1 : 3*cLen/4);

    % Clamp radii and heights to their allowed intervals; project into the
    % feasible set defined by the constraints
    % minR <= r_i <= maxR for all i,
    % minH <= h_i <= maxH for all i:
    c1h = max(minH, min(maxH, c1h));
    c2h = max(minH, min(maxH, c2h));
    if (optimizeRadii)
        c1r = max(minR, min(maxR, c1r));
```

```
        c2r = max(minR, min(maxR, c2r));
    end

    % Ensure that tubes do not cross; project into the feasible set defined
    % by the constraints
    % h1_i <= h2_i - minTubeDistance for all i
    % (hj being the heights of tube number j):
    for i=1:length(c1h)
        x1 = c1h(i);
        x2 = c2h(i);
        x1Projected = x1;
        x2Projected = x2;
        if (x1 > x2 - minTubeDistance)
            % Projects(x1, x2) onto the line x1 = x2 - minTubeDistance,
            % i.e. project back into the feasible set defined by
            % x1 <= x2 - minTubeDistance:
            x2Projected = (minTubeDistance + x1 + x2) / 2;
            x1Projected = x2Projected - minTubeDistance;
        end
        c1h(i) = x1Projected;
        c2h(i) = x2Projected;
    end
    cProjected = [c1r, c1h, c2r, c2h];

    % Stop if the new point, after projection, is the original point
    if(max(abs(cProjected - cCurrent)) < TOL_p)
        flag = true;
        disp('Projected onto previous point, stopping')
        continue
    end

    if(doPlot)
        if (optimizeRadii)
            newPlotVals = cProjected;
        else
            newPlotVals = [c1h, c2h];
        end
        for i=1:length(newPlotVals)
            x = [iteration-1, iteration];
            y = [oldPlotVals(i), newPlotVals(i)];
            c = mod(0.4*i, 1);
            figure(2)
            plot(x, y, 'color', [c, c, c], 'marker', '*')
        end
        oldPlotVals = newPlotVals;
        drawnow
    end

    cCurrent = cProjected;

    if(hasUserInterrupted)
        disp('User ended optimization')
        flag = true;
    end
end

objOpt = objVal;
cOpt = cCurrent;

end

function forceQuitFcn(~, ~)

    global hasUserInterrupted
    hasUserInterrupted = true;
end
```

```
function [val, grad] = objective(c, iteration)

optimizeRadii = evalin('base', 'optimizeRadii');
rLen = evalin('base', 'rLen');
zLen = evalin('base', 'zLen');
stepLenR = -0.005 / iteration;
stepLenH = 0.03 / iteration;

u = solvePDECyl(c);
val = avgNeckTemp(u);

% Curves c1 and c2 are found by c=[c1, c2]
grad = zeros(1, length(c));
% c1 height:
buildGradient(stepLenH, 0, zLen, length(c)/4+1, length(c)/2);
% c2 height:
buildGradient(-stepLenH, 0, zLen, 3*length(c)/4+1, length(c));
```

```
if (optimizeRadii)
    % c1 radius:
    buildGradient(stepLenR, 0, rLen, 1, length(c)/4);
    % c2 radius:
    buildGradient(stepLenR, 0, rLen, length(c)/2+1, 3*length(c)/4);
end

function buildGradient(stepLen, minVal, maxVal, varIndMin, varIndMax)
    for k=varIndMin:varIndMax
        cWithStep = c;
        newPoint = c(k) + stepLen;
        if(minVal < newPoint && newPoint < maxVal)
            cWithStep(k) = newPoint;
            uStep = avgNeckTemp(solvePDECyl(cWithStep));
            grad(k) = (uStep - val) / stepLen;
        else
            cWithStep(k) = c(k) - stepLen;
            uStep = avgNeckTemp(solvePDECyl(cWithStep));
            grad(k) = - (uStep - val) / stepLen;
        end
    end

end

end
```

```
function val = avgNeckTemp(u)

% This function calculates the average temperature on the plane r=r0, i.e.
% the imagined skin on the patient's neck. This is its own dedicated
% function file in order to simplify the implementation of an arbitrary
% objective function.

Nr = evalin('base', 'Nr');
r0 = evalin('base', 'r0');
rLen = evalin('base', 'rLen');

midRadius = floor(r0 / rLen * Nr);
utemp = u(midRadius+1, :, :);
val = mean(utemp(:));

end
```

A.3 Finit differenslösare

```
function u = solvePDECyl(c)

% This function solves a finite difference heat equation using the
% successive over-relaxation iteration method.

% ------ PARAMETERS ------
TOL = 0.01;   % The tolerance which determines solver convergence. The
              % solver finishes, for all points in the grid, the difference
              % between the new value and previous iteration's value is
              % less than this tolerance.
omega = 1.7;  % The weight parameter used in the successive over-relaxation.
% ------ END OF PARAMETERS ------

Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
rLen = evalin('base', 'rLen');
hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
u = zeros(Nr, Nt, Nz);

% Generate the coordinates of all points that lie in water tubes:
c1 = c(1 : length(c)/2);
c2 = c(length(c)/2 + 1 : end);
tubeIndices = [interpolateCurve(c1), interpolateCurve(c2)];
tubeIndices = unique(tubeIndices', 'rows')';

% Construct all r,t,z such that no point lies on a tube:
interiorIndexTriples = [];
for r=1:(Nr-2)
    for t=1:(Nt-2)
        for z=1:(Nz-2)
            if(ismember([r, t, z], tubeIndices', 'rows'))
```

```
                continue
            end
            interiorIndexTriples = [interiorIndexTriples, [r; t; z]];
        end
    end
end

% Set the boundary condition on the plane r=rLen:
for t=0:(Nt-1)
    for z=0:(Nz-1)
        u(Nr, t+1, z+1) = 5;
    end
end
% Set the boundary condition on the planes z=0 and z=zLen:
for t=0:(Nt-1)
    for r=0:(Nr/2)
        u(r+1, t+1, 1) = 37;
        u(r+1, t+1, Nz) = 37;
    end
    for r=(Nr/2+1):(Nr-1)
        u(r+1, t+1, 1) = 20;
        u(r+1, t+1, Nz) = 20;
    end
end

% Solve the heat equation iteratively using successive over-relaxation:
flag = true;  % flag which determines convergence
k = 0;  % counter which counts the number of iterations
aii =@(r) -2/hr2 - 2/r^2/ht2 - 2/hz2;  % helper function
aiiCenter = -4/hr2 - 2/hz2;            % helper constant
constCenter = omega / aiiCenter;       % helper constant
while(flag)
    flag = false;
    k = k + 1;

    % Iterate over the interior of the domain
    for i=1:length(interiorIndexTriples)
        r = interiorIndexTriples(1, i);
        t = interiorIndexTriples(2, i);
        z = interiorIndexTriples(3, i);

        metricR = r / (Nr - 1) * rLen;
        const = omega / aii(metricR);
        uold = u(r+1, t+1, z+1);
        unew = iterateInterior(u, [r+1, t+1, z+1], uold, omega, const, metricR);
        u(r+1, t+1, z+1) = unew;
        if(abs(uold - unew) > TOL)
            flag = true;
        end
    end

    % Iterate over the two planes theta=0, theta=2pi, ensuring periodicity,
    % i.e (r,0,z) = (r,2pi,z)
    for r=1:(Nr-2)
        for z=1:(Nz-2)
            metricR = r / (Nr - 1) * rLen;
            const = omega / aii(metricR);
            uoldL = u(r+1, 1, z+1);
            uoldR = u(r+1, Nt, z+1);
            unewL = iterateBoundary(u, [r+1, 1, z+1], uoldL, omega, const, metricR);
            unewR = iterateBoundary(u, [r+1, Nt, z+1], uoldR, omega, const, metricR);
            u(r+1, 1, z+1) = unewL;
            u(r+1, Nt, z+1) = unewR;
            if(abs(uoldL - unewL) > TOL || abs(uoldR - unewR) > TOL)
                flag = true;
            end
        end
    end

    % Iterate over the center, r=0
    for z=1:(Nz-2)
        uold = u(1, 1, z+1);
        unew = iterateCenter(u, [1, 1, z+1], uold, omega, constCenter);
        u(1, 1, z+1) = unew;
        if(abs(uold - unew) > TOL)
            flag = true;
        end
    end

    % Copy (0,0,z) to all (0,t,z) since all of these are the center
    for t=1:(Nt-1)
        for z=1:(Nz-2)
            u(1, t+1, z+1) = u(1, 1, z+1);
        end
```

```
    end

    % Iterate over all points on tubes
    for i=1:length(tubeIndices)
        r = tubeIndices(1, i);
        t = tubeIndices(2, i);
        z = tubeIndices(3, i);

        metricR = r / (Nr - 1) * rLen;
        const = omega / aii(metricR);
        uold = u(r+1, t+1, z+1);
        unew = iterateTube(u, [r+1, t+1, z+1], uold, omega, const, metricR, tubeIndices);
        u(r+1, t+1, z+1) = unew;
        if(abs(uold - unew) > TOL)
            flag = true;
        end
    end
end

nCalls = evalin('base', 'solverCalls') + 1;
assignin('base', 'solverCalls', nCalls);
disp(PDE solver finished after  + k +  iterations,  ...
    + nCalls + th call)

end
```

```
function unew = iterateInterior(u, x, uold, omega, const, r)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
k = evalin('base', 'k');
heatSource = evalin('base', 'heatSource');

unew = (1-omega) * uold + const * ( ...
        -heatSource(x(1), x(2), x(3)) / k(r) ...
        -(-1/r/2/hr + 1/hr2) * u(x(1)-1, x(2),   x(3)) ...
        -1/hz2               * u(x(1),   x(2),   x(3)-1) ...
        -1/r^2/ht2           * u(x(1),   x(2)-1, x(3)) ...
        -1/r^2/ht2           * u(x(1),   x(2)+1, x(3)) ...
        -1/hz2               * u(x(1),   x(2),   x(3)+1) ...
        -(1/r/2/hr + 1/hr2)  * u(x(1)+1, x(2),   x(3)) ...
    );

end
```

```
function unew = iterateBoundary(u, x, uold, omega, const, r)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
heatSource = evalin('base', 'heatSource');
k = evalin('base', 'k');

if (x(2) == 1)
    unew = (1-omega) * uold + const * ( ...
        -heatSource(x(1), x(2), x(3)) / k(r) ...
        -(-1/r/2/hr + 1/hr2) * u(x(1)-1, x(2), x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)-1) ...
        -1/r^2/ht2           * u(x(1), Nt, x(3)) ...
        -1/r^2/ht2           * u(x(1), x(2)+1, x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)+1) ...
        -(1/r/2/hr + 1/hr2)  * u(x(1)+1, x(2), x(3)) ...
    );
else
    unew = (1-omega) * uold + const * ( ...
        -heatSource(x(1), x(2), x(3)) / k(r) ...
        -(-1/r/2/hr + 1/hr2) * u(x(1)-1, x(2), x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)-1) ...
        -1/r^2/ht2           * u(x(1), x(2)-1, x(3)) ...
        -1/r^2/ht2           * u(x(1), 1, x(3)) ...
```

```
        -1/hz2               * u(x(1), x(2), x(3)+1) ...
        -(1/r/2/hr + 1/hr2)  * u(x(1)+1, x(2), x(3)) ...
    );
end

end
```

```
function unew = iterateCenter(u, x, uold, omega, const)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
k = evalin('base', 'k');
heatSource = evalin('base', 'heatSource');

unew = (1-omega) * uold + const * ( ...
        -heatSource(x(1), x(2), x(3)) / k(0) ...
        -1/hr2 * u(x(1)+1, x(2), x(3)) ...               % +(1, 0, 0)
        -1/hr2 * u(x(1)+1, x(2)+floor(Nt/4), x(3)) ...   % +(1, pi/2, 0)
        -1/hr2 * u(x(1)+1, x(2)+floor(Nt/2), x(3)) ...   % +(1, pi, 0)
        -1/hr2 * u(x(1)+1, x(2)+floor(3*Nt/4), x(3)) ... % +(1, 3pi/2, 0)
        -1/hz2 * u(x(1), x(2), x(3)-1) ...
        -1/hz2 * u(x(1), x(2), x(3)+1) ...
    );

end
```

```
function unew = iterateTube(u, x, uold, omega, const, r, tubeIndices)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
fluxTube = evalin('base', 'fluxTube');
k = evalin('base', 'k');

if(ismember([x(1)-2, x(2)-1, x(3)-1], tubeIndices', 'rows'))
    % If r+1 is hydrogel
    rightCoeff = 2/hr2;
    leftCoeff = 0;
    fluxR = -fluxTube/k(r) * (2/hr - 1/r);
else
    % If r-1 is hydrogel
    rightCoeff = 0;
    leftCoeff = 2/hr2;
    fluxR = -fluxTube/k(r) * (2/hr + 1/r);
end

if(ismember([x(1)-1, x(2)-1, x(3)-2], tubeIndices', 'rows'))
    % If z+1 is hydrogel
    upCoeff = 2/hz2;
    downCoeff = 0;
    fluxZ = -2*fluxTube/k(r)/hz;
else
    % If z-1 is hydrogel
    upCoeff = 0;
    downCoeff = 2/hz2;
    fluxZ = -2*fluxTube/k(r)/hz;
end

unew = (1-omega) * uold + const * ( ...
        fluxR + fluxZ ...
        -leftCoeff  * u(x(1)-1, x(2),   x(3)  ) ...
        -downCoeff  * u(x(1),   x(2),   x(3)-1) ...
        -1/r^2/ht2  * u(x(1),   x(2)-1, x(3)  ) ...
        -1/r^2/ht2  * u(x(1),   x(2)+1, x(3)  ) ...
        -upCoeff    * u(x(1),   x(2),   x(3)+1) ...
        -rightCoeff * u(x(1)+1, x(2),   x(3)  ) ...
    );
end
```

A.4 Övriga funktioner

```
function cCoords = interpolateCurve(c)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');

startT = 0.5;
endT = 2*pi - startT;

% Approximate curve within grid:
% The first half of c contains all node radii, the second half contains all
% node heights; these are cr and cz below, respectively. The node angles
% are distributed equally from startT to endT radians; this is ct below.
ct = linspace(startT, endT, length(c)/2);
cr = c(1:(length(c)/2));
cz = c((length(c)/2+1):end);
% Convert (with rounding) node coordinates from meters to PDE domain grid
% indices:
cCoords0 = round([cr./hr; ct./ht; cz./hz]);

cCoords = [];
segmentLength = (Nt / length(ct)).^2;
for i=1:(length(ct)-1)
    p0 = cCoords0(:, i);   % segment start point
    p1 = cCoords0(:, i+1); % segment end point
    % Assume segment can cover max segmentLength grid points
    segment = [round(linspace(p0(1), p1(1), segmentLength));
               round(linspace(p0(2), p1(2), segmentLength));
               round(linspace(p0(3), p1(3), segmentLength))];

    % Make the water tube 2x2 in thickness; allowing for easier
    % implmenentation of Neumann boundary onditions:
    segmentRight = segment + [1; 0; 0];
    segmentUp = segment + [0; 0; 1];
    segmentUpRight = segment + [1; 0; 1];

    cCoords = [cCoords, segment, segmentRight, segmentUp, segmentUpRight];
end

end
```

```
function q = setupHeatSource(Nr, Nt, Nz)

% use extractNeckFromData to generate heat source with same size as u
data = extractNeckFromData;
X = data(:, 1);
Y = data(:, 2);
Z = data(:, 3);
W = data(:, 4);
r0 = evalin('base', 'r0');
rLen = evalin('base', 'rLen');
tLen = evalin('base', 'tLen');
zLen = evalin('base', 'zLen');
q = zeros(Nr, Nt, Nz);

% In data from extractNeckFromData, x coord is along slice,
% y from 0 to negative, z is height.
rMaxIndex = floor(r0 / rLen * Nr);
for r=0:rMaxIndex
    for t=0:(Nt-1)
        for z=0:(Nz-1)
            % Convert from PDE domain grid units to metric units
            metricR = r / (Nr - 1) * rLen;
            metricT = t / (Nt - 1) * tLen;
            metricZ = z / (Nz - 1) * zLen;
            % Convert to Cartesian coordinates, since simulation data is
            % Cartesian
            [xM, yM, zM] = pol2cart(metricT, metricR, metricZ);
            yM = - abs(yM);  % only negative half is included in simulation
            % data. Remove 4% to avoid sampling at (x,y) not included in
            % extracted neck data
            xData = 0.96 * xM / r0 * max(X);
            yData = 0.96 * yM / r0 * abs(min(Y));
            zData = zM / zLen * (max(Z) - min(Z)) + min(Z);
            xData = getNearestIn(xData, X);
            yData = getNearestIn(yData, Y);
            zData = getNearestIn(zData, Z);
            ind = (X == xData) & (Y == yData) & (Z == zData);
```

```
            q(r+1, t+1, z+1) = W(ind);
        end
    end
end

% for i=1:length(q)
% end
% figure(4)
% plot(xtemp, ytemp)

end

function v = getNearestIn(x, xs)
xs = sort(xs);
for i=1:length(xs)
    if (x < xs(i))
        v = xs(i);
        return
    end
end
v = xs(end);

end
```

```
function cylData = extractNeckFromData

% This function extracts a cylindrical volume from the volume given in the
% imported data. Assuming that the most energy is absorbed within the
% patient and not in the hydrogel, we extract a cylinder with a radius
% such that the average absorbed effect within the cylinder is as large as
% possible.

A = csvread('C:\Users\Erik\Documents\CST_EM_data\PLDBastiaanMediumNeck.csv');
XA = A(:,1);
YA = A(:,2);
ZA = A(:,3);
val = A(:,4);

% Here we decided, after plotting the data, to extract a cylinder between
% zmin and zmax (in the units of the imported data) in height.
zmin = -0.05;
zmax =  0.05;
I = ZA >= zmin & ZA <= zmax;
XA = XA(I);
YA = YA(I);
ZA = ZA(I);
val = val(I);

% Find a radius such that the corresponding cylinder encloses a volume of
% greatest absorbed effect:
% n = 20;
% rs = linspace(0.04, 0.1, n);
% meanVals = [];
% for r=rs
%     I = XA.^2 + YA.^2 <= r^2;
%     meanVal = mean(val(I));
%     meanVals = [meanVals, meanVal];
% end
% [~, i] = max(meanVals);
% r = rs(i+1);
r = evalin('base', 'r0');
I = XA.^2 + YA.^2 <= r^2;
cylData = [XA(I), YA(I), ZA(I), val(I)];

end
```

B Matlab-kod enkel modell

B.1 Huvudskript

```
% ------ INPUT PARAMETERS ------
% Note: all physical quantities are expressed in SI units.
% Domain dimensions:
r0 = 0.1;      % inner radius
rLen = 0.1;    % radial thickness
zLen = 0.1;    % height
% Resolution of the grid for the discrete heat equation:
Nr = 10;
Nt = 10;
```

```
Nz = 10;
% Amount of heat flux (W/m^2) from the patient's neck, i.e. along the
% normal from the skin pointing in to the hydrogel:
flux = 100;
% Optimization constraints:
minR = 0.02;         % Minimum radius (from center) of tube nodes
maxR = rLen - 0.02;  % Maximum radius (from center) of tube nodes
minH = 0.02;         % Minimum radius (from center) of tube nodes
maxH = zLen - 0.02;  % Maximum radius (from center) of tube nodes
% ------ END OF INPUT PARAMETERS ------
tLen = 2*pi;  % domain length in theta direction
% Step lengths
hr = rLen / (Nr - 1);
ht = tLen / (Nt - 1);
hz = zLen / (Nz - 1);
%Material parameters:
k = 0.6;  % thermal conductivity of water = 0.6

solverCalls = 0;  % counter to track how many times the solver is called

% ------ curve -------
% The water tube is modelled as a piecewise linear curve inbetween a
% number of nodes; these nodes have variable heights and radii, and these
% are the variables in the optimization problem. The initial tube curve,
% c0, is defined below.
curveRes = 7;  % the number of nodes on each curve (including endpoints)
ch = 0.5*zLen*ones(1, curveRes);  % the heights of all nodes on c0
cr = 0.5*rLen*ones(1, curveRes);  % the radii of all nodes on c0
% All optimization variables are collected into a single vector as defined
% below.
c0 = [cr, ch];  % input to algorithm
[uMaxOpt, cOpt] = gradProj(@objective, c0);
uOpt = solvePDECyl(cOpt);

%% Plot
figure(1)
clf
hold on
[R, T, Z] = ndgrid(1:size(uOpt, 1), 1:size(uOpt, 2), 1:size(uOpt, 3));
S = uOpt(:);
S = 1*(S - min(S) + 100);
plot3DField(R(:), T(:), Z(:), S)
view(-30, 25)
colorbar

%% Plot tube
cCoords = interpolateCurve(cOpt) + [1; 1; 1];
scatter3(cCoords(1, :), cCoords(2, :), cCoords(3, :), ...
         200, [1, 1, 1], 'filled')
axis equal
colormap winter
colorbar
```

B.2 Gradient projection-algoritm

```
% u is a function returning [uValue, gradient], c0 is the initial point to
% start minimizing from.
function [objOpt, cOpt] = gradProj(objective, c0)

minR = evalin('base', 'minR');
maxR = evalin('base', 'maxR');
minH = evalin('base', 'minH');
maxH = evalin('base', 'maxH');

global hasUserInterrupted
hasUserInterrupted = false;

doPlot = true;
oldPlotVals = c0;
if(doPlot)
    figure(2)
    clf
    hold on
    grid on
    set(gcf, 'KeyPressFcn', @forceQuitFcn)
    drawnow
end

iteration = 0;

gradTol = 0.0001;
projTol = 0.0001;
```

```
28
29  cCurrent = c0;
30  flag = false;
31  while(~flag)
32      iteration = iteration + 1;
33      stepLen = 0.005 / iteration;
34      disp(Starting iteration  + iteration)
35      [objVal, grad] = objective(cCurrent, iteration);
36
37      % Stop if gradient is small
38      if(max(abs(grad)) < gradTol)
39          flag = true;
40          disp('Gradient below tolerance, stopping')
41          continue
42      end
43  %     disp(grad)
44      cNew = cCurrent - stepLen.*grad;  % Step along negative gradient
45      cr = cNew(1 : length(cNew)/2);  % radii
46      ch = cNew(length(cNew)/2 + 1 : end);  % heights
47      cr = max(minR, min(maxR, cr));  % Clamp radii
48      ch = max(minH, min(maxH, ch));  % Clamp heights
49      cProjected = [cr, ch];
50  %     disp(cCurrent)
51  %     disp(cNew)
52  %     disp(cProjected)
53      % Stop if the new point, after projection, is the original point
54      if(max(abs(cProjected - cCurrent)) < projTol)
55          flag = true;
56          disp('Projected onto previous point, stopping')
57          continue
58      end
59
60      if(doPlot)
61          newPlotVals = cProjected;
62          for i=1:length(newPlotVals)
63              x = [iteration-1, iteration];
64              y = [oldPlotVals(i), newPlotVals(i)];
65              c = mod(0.4*i, 1);
66              plot(x, y, 'color', [c, c, c], 'marker', '*')
67          end
68          oldPlotVals = newPlotVals;
69          drawnow
70      end
71
72      cCurrent = cProjected;
73
74      if(hasUserInterrupted)
75          disp('User ended minimization')
76          flag = true;
77      end
78  end
79
80  objOpt = objVal;
81  cOpt = cCurrent;
82
83  end
84
85  function forceQuitFcn(~, ~)
86
87      global hasUserInterrupted
88      hasUserInterrupted = true;
89  end
```

```
1   function [umax, grad] = objective(c, iteration)
2
3   rLen = evalin('base', 'rLen');
4   zLen = evalin('base', 'zLen');
5
6
7   gradStepR = -0.005 / iteration;
8   gradStepH = 0.03 / iteration;
9
10  u = solvePDECyl(c);
11  umax = maxBoundaryTemp(u);
12
13  curveRes = length(c) / 2;
14  gradR = zeros(1, curveRes);
15  gradH = gradR;
16  for i=1:curveRes
17      cWithStep = c;
18      if(cWithStep(i) + gradStepR < rLen)
19          cWithStep(i) = cWithStep(i) + gradStepR;
20          uStep = maxBoundaryTemp(solvePDECyl(cWithStep));
21          gradR(i) = (uStep - umax) / gradStepR;
```

```
    else
        cWithStep(i) = cWithStep(i) - gradStepR;
        uStep = maxBoundaryTemp(solvePDECyl(cWithStep));
        gradR(i) = (umax - uStep) / gradStepR;
    end
end
for i=(curveRes+1):(2*curveRes)
    cWithStep = c;
    if(cWithStep(i) + gradStepH < zLen)
        cWithStep(i) = cWithStep(i) + gradStepH;
        uStep = maxBoundaryTemp(solvePDECyl(cWithStep));
        gradH(i-curveRes) = (uStep - umax) / gradStepH;
    else
        cWithStep(i) = cWithStep(i) - gradStepH;
        uStep = maxBoundaryTemp(solvePDECyl(cWithStep));
        gradH(i-curveRes) = (umax - uStep) / gradStepH;
    end
end

grad = [gradR, gradH];
end
```

```
function umax = maxBoundaryTemp(u)

Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');

% umax = -Inf;
% uavg = 0;
% for t=0:(Nt-1)
%     for z=0:(Nz-1)
%         i = coordToIndex([0, t, z]);
%         uavg = uavg + u(i);
%     end
% end
% umax = uavg / (Nt*Nz);
utemp = u(1, :, :);
umax = mean(utemp(:));

end
```

B.3 Finit differenslösare

```
function u = solvePDECyl(c)

% Domain dimensions
r0 = evalin('base', 'r0');
rLen = evalin('base', 'rLen');
% Number of discrete points, including endpoints
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
% Step lengths
hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;

omega = 1.6;
TOL = 0.01;

% Interpolated path of indices on curve:
tubeIndices = interpolateCurve(c);
% tubeBoundaryInds = zeros(length(cIndices(1,:)), 1);
% tubeIndices = [];
% for i=1:length(cIndices(1,:))
%     j = coordToIndex(cIndices(:,i));
% %     tubeBoundaryInds = [tubeBoundaryInds, j];
%     tubeBoundaryInds(i) = j;
% end

% u = zeros(Nt*Nr*Nz, 1);
u = zeros(Nr, Nt, Nz);
% outerBoundaryInds = zeros(Nt*Nz, 1);
% BC on the plane r=rLen:
% i = 0;
% for t=0:(Nt-1)
%     for z=0:(Nz-1)
%         i = i + 1;
```

```
%         j = coordToIndex([Nr-1, t, z]);  % outer side, constant 20
%         outerBoundaryInds(i) = j;
% %         outerBoundaryInds = [outerBoundaryInds, i];
%
%         u(j) = 5;
%     end
% end
for t=1:Nt
    for z=1:Nz
        u(Nr, t, z) = 5;
    end
end
% topBoundaryInds = zeros(Nt*Nr, 1);
% botBoundaryInds = zeros(Nt*Nr, 1);
% BC on top and bottom:
for t=1:Nt
    for r=1:Nr
        u(r, t, 1)  = 20;
        u(r, t, Nz) = 20;
    end
end

interiorIndexTriples = [];
for r=1:(Nr-2)
    for t=1:(Nt-2)
        for z=1:(Nz-2)
            if(ismember([r,t,z], tubeIndices', 'rows'))
                continue
            end
            interiorIndexTriples = [interiorIndexTriples, [r; t; z]];
        end
    end
end

% sideIndsL = zeros((Nr-2)*(Nz-2), 1);
% sideIndsR = zeros((Nr-2)*(Nz-2), 1);
% i = 0;
% for r=1:(Nr-2)
%     for z=1:(Nz-2)
%         i = i + 1;
%         j = coordToIndex([r, 0, z]);
% % %         sideIndsL = [sideIndsL, j];
%         sideIndsL(i) = j;
%
%         j = coordToIndex([r, Nt-1, z]);
% % %         sideIndsR = [sideIndsR, j];
%         sideIndsR(i) = j;
%     end
% end

% innerBoundaryInds = [];
% % innerBoundaryInds = zeros((Nt-2)*(Nz-2), 1);
% % i = 0;
% for t=1:(Nt-2)
%     for z=1:(Nz-2)
% % %         i = i + 1;
%         j = coordToIndex([0, t, z]);
%         innerBoundaryInds = [innerBoundaryInds, j];
% % %         innerBoundaryInds(i) = j;
%     end
% end

% cornerIndsL = zeros(Nz-2, 1);
% cornerIndsR = zeros(Nz-2, 1);
% for z=1:(Nz-2)
%     i = coordToIndex([0, 0, z]);
% % %     cornerIndsL = [cornerIndsL, i];
%     cornerIndsL(z) = i;
%
%     i = coordToIndex([0, Nt-1, z]);
% % %     cornerIndsR = [cornerIndsR, i];
%     cornerIndsR(z) = i;
% end

% BC on curve:
% for k=1:length(cIndices)
%     i = coordToIndex(cIndices(:,k));
%     boundaryIndices = [boundaryIndices, i];
%     u(i) = 5;
% end

% For steady-state heat eqn. Laplace(u)=0, discrete version is
%   (u_{i+1}     - 2*u_i + u_{i-1}    ) / hx^2
% + (u_{i+Nx}    - 2*u_i + u_{i-Nx}   ) / hy^2
```

```
% + (u_{i+Nx*Ny} - 2*u_i + u_{i+Nx*Ny}) / hz^2
% = 0

% NOW USING SUCCESSIVE OVER-RELAXATION, NOT GS
% Gauss-Seidel:
% u_i^0 = 0. Iterate:
% u_i^{k+1} = 1/a_{ii} * (
%   b_i - sum j=1 to i-1 of a_{ij}*u_j^{k+1}
%       - sum j=i+1 to N of a_{ij}*u_j^k
% )
% where a_ij are elements in the matrix corresponding to the discrete
% equations above, b_i is the RHS of each equation (=0).
% In this case:
% a_ii = a = -2/hx - 2/hy - 2/hz
% b_i = 0
% u_i^{k+1} = 1/a * (
%   - 1/hz*u_{i-Nx*Ny}^{k+1} - 1/hy*u_{i-Nx}^{k+1} - 1/hx*u_{i-1}^{k+1}
%   - 1/hx*u_{i+1}^k - 1/hy*u_{i+Nx}^k - 1/hz*u_{i+Nx*Ny}^k

flag = true;
k = 0;

aii =@(r) -2/hr2 - 2/r^2/ht2 -2/hz2;

while(flag)
    flag = false;
    k = k + 1;
    for i=1:length(interiorIndexTriples)
        r = interiorIndexTriples(1, i);
        t = interiorIndexTriples(2, i);
        z = interiorIndexTriples(3, i);

        metricR = (r - 1) / (Nr - 1) * rLen + r0;
        const = omega / aii(metricR);

        uold = u(r+1, t+1, z+1);
        unew = iterateInterior(u, [r+1, t+1, z+1], uold, omega, const, metricR);
        u(r+1, t+1, z+1) = unew;
        if(abs(uold - unew) > TOL)
            flag = true;
        end
    end

    for t=2:(Nt-1)
        for z=2:(Nz-1)
            metricR = r0;
            const = omega / aii(metricR);

            uold = u(1, t, z);
            unew = iterateInnerBoundary(u, [1, t, z], uold, omega, const, metricR);
            u(1, t, z) = unew;
            if(abs(uold - unew) > TOL)
                flag = true;
            end
        end
    end

    % Periodicity
    for r=2:(Nr-1)
        for z=2:(Nz-1)
            metricR = (r - 1) / (Nr - 1) * rLen + r0;
            const = omega / aii(metricR);

            uoldL = u(r, 1,  z);
            uoldR = u(r, Nt, z);
            unewL = iterateBoundary(u, [r, 1,  z], uoldL, omega, const, metricR);
            unewR = iterateBoundary(u, [r, Nt, z], uoldR, omega, const, metricR);
            u(r, 1,  z) = unewL;
            u(r, Nt, z) = unewR;
            if(abs(uoldL - unewL) > TOL ||abs(uoldR - unewR) > TOL)
                flag = true;
            end
        end
    end

    for z=2:(Nz-1)
        metricR = r0;
        const = omega / aii(metricR);

        uoldL = u(1, 1,  z);
        uoldR = u(1, Nt, z);
        unewL = iterateInnerCorner(u, [1, 1,  z], uoldL, omega, const, r);
        unewR = iterateInnerCorner(u, [1, Nt, z], uoldR, omega, const, r);
        u(1, 1,  z) = unewL;
```

```
        u(1, Nt, z) = unewR;
        if(abs(uoldL - unewL) > TOL ||abs(uoldR - unewR) > TOL)
            flag = true;
        end
    end

end

nCalls = evalin('base', 'solverCalls') + 1;
assignin('base', 'solverCalls', nCalls);
disp(PDE solver finished after  + k +  iterations,  ...
    + nCalls + th call)

end
```

```
function unew = iterateInterior(u, x, uold, omega, const, r)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');

unew = (1-omega) * uold + const * ( ...
        -(-1/r/2/hr + 1/hr2) * u(x(1)-1, x(2),   x(3)) ...
        -1/hz2               * u(x(1),   x(2),   x(3)-1) ...
        -1/r^2/ht2           * u(x(1),   x(2)-1, x(3)) ...
        -1/r^2/ht2           * u(x(1),   x(2)+1, x(3)) ...
        -1/hz2               * u(x(1),   x(2),   x(3)+1) ...
        -(1/r/2/hr + 1/hr2)  * u(x(1)+1, x(2),   x(3)) ...
    );

end
```

```
function unew = iterateBoundary(u, x, uold, omega, const, r)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');

if (x(2) == 1)
    unew = (1-omega) * uold + const * ( ...
        -(-1/r/2/hr + 1/hr2) * u(x(1)-1, x(2), x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)-1) ...
        -1/r^2/ht2           * u(x(1), Nt, x(3)) ...
        -1/r^2/ht2           * u(x(1), x(2)+1, x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)+1) ...
        -(1/r/2/hr + 1/hr2)  * u(x(1)+1, x(2), x(3)) ...
    );
else
    unew = (1-omega) * uold + const * ( ...
        -(-1/r/2/hr + 1/hr2) * u(x(1)-1, x(2), x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)-1) ...
        -1/r^2/ht2           * u(x(1), x(2)-1, x(3)) ...
        -1/r^2/ht2           * u(x(1), 1, x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)+1) ...
        -(1/r/2/hr + 1/hr2)  * u(x(1)+1, x(2), x(3)) ...
    );
end
end
```

```
function unew = iterateInnerBoundary(u, x, uold, omega, const, r)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
k = evalin('base', 'k');
```

```
flux = evalin('base', 'flux');

unew = (1-omega) * uold + const * ( ...
        -flux/k * (2/hr-1/r) ...
        -1/hz2               * u(x(1),   x(2),   x(3)-1) ...
        -1/r^2/ht2           * u(x(1),   x(2)-1, x(3)) ...
        -1/r^2/ht2           * u(x(1),   x(2)+1, x(3)) ...
        -1/hz2               * u(x(1),   x(2),   x(3)+1) ...
        -2/hr2               * u(x(1)+1, x(2),   x(3)) ...
    );

end
```

```
function unew = iterateInnerCorner(u, x, uold, omega, const, r)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
k = evalin('base', 'k');
flux = evalin('base', 'flux');

if (x(2) == 1)
    unew = (1-omega) * uold + const * ( ...
        -flux/k * (2/hr-1/r) ...
        -1/hz2               * u(x(1), x(2), x(3)-1) ...
        -1/r^2/ht2           * u(x(1), Nt, x(3)) ...
        -1/r^2/ht2           * u(x(1), x(2)+1, x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)+1) ...
        -2/hr2               * u(x(1)+1, x(2), x(3)) ...
    );
else
    unew = (1-omega) * uold + const * ( ...
        -flux/k * (2/hr-1/r) ...
        -1/hz2               * u(x(1), x(2), x(3)-1) ...
        -1/r^2/ht2           * u(x(1), x(2)-1, x(3)) ...
        -1/r^2/ht2           * u(x(1), 1, x(3)) ...
        -1/hz2               * u(x(1), x(2), x(3)+1) ...
        -2/hr2               * u(x(1)+1, x(2), x(3)) ...
    );
end

end
```

B.4 Övriga funktioner

```
function cCoords = interpolateCurve(c)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');

startT = 0.5;
endT = 2*pi - startT;

% Approximate curve within grid:
% Create vector of points on curve with the same dimensions as the domain
% grid.
% We map from xyz-coordinates to ijk-indices by
% (x,y,z) |-> (x/hx, y/hy, x/hx) and round.
% xyz-coords are [cx; cr; ch]
ct = linspace(startT, endT, length(c)/2);
cr = c(1:(length(c)/2));
cz = c((length(c)/2+1):end);
% Indices of curve points (as [i; j; k]):
cCoords0 = round([cr./hr; ct./ht; cz./hz]);

cCoords = [];
segmentLength = 1.5 * (Nt / length(ct)).^2;
for i=1:(length(ct)-1)
    p0 = cCoords0(:,i);   % segment start point
    p1 = cCoords0(:,i+1); % segment end point
%     Assume segment can cover max segmentLength grid points
    segment = [round(linspace(p0(1), p1(1), segmentLength));
```

```
                round(linspace(p0(2), p1(2), segmentLength));
                round(linspace(p0(3), p1(3), segmentLength))];
    cCoords = [cCoords, segment];
end

end
```

C Matlab-kod laboration

C.1 Huvudskript

```
% This one is Cartesian. r,t,z denote z,x,y respectively. Comments may be
% old and using cylindrical coordinates.

% ------ INPUT PARAMETERS ------
% Note: all physical quantities are expressed in SI units.
% Domain dimensions:
rLen = 0.05;
tLen = 0.36;
zLen = 0.2;
% Resolution of the grid for the discrete heat equation:
Nr = 10;  % Number of points in radius direction
Nt = 20;  % Number of points in angular direction
Nz = 10;  % Number of points in height direction
% Amount of heat flux (W/m^2) on tube boundaries:
fluxTube = -200;
% Optimization constraints:
minR = 0.02;    % Minimum radius (from center) of tube nodes
maxR = rLen - 0.02;  % Maximum radius (from center) of tube nodes
minH = 0.04;         % Minimum height of tube nodes
maxH = zLen - 0.04;  % Maximum height of tube nodes
% Other settings:
optimizeRadii = false;  % Set optimizeRadii = true if you want to include
                        % node radii as variables in the optimization
                        % problem, false otherwise.
plotIterations = true;  % Set plotIterations = true if you want a simple
                        % plot of all variable values after each gradient
                        % projection iteration, false otherwise.
% ------ END OF INPUT PARAMETERS ------
% Step lengths
hr = rLen / (Nr - 1);
ht = tLen / (Nt - 1);
hz = zLen / (Nz - 1);
k = 0.6;  % thermal conductivity of water
solverCalls = 0;  % counter to track how many times the solver is called

% The water tubes are modelled as piecewise linear curves inbetween a
% number of nodes; these nodes have variable heights and radii, and these
% are the variables in the optimization problem* (*see option optimizeRadii
% above). The initial tube curves, c1_0, and c2_0, are defined below.
curveRes = 12;  % the number of nodes on each curve (including endpoints)
c1h = 0.5*zLen*ones(1, curveRes);  % the heights of all nodes on c1_0
c1r = 0.5*rLen*ones(1, curveRes);  % the radii of all nodes on c1_0
% All optimization variables (always including radii regardless of
% optimizeRadii) are collected into a single vector as defined below.
c0 = [c1r, c1h];

cOpt = c0;
% [uMaxOpt, cOpt] = gradProj(@objective, c0);
uOpt = solvePDECyl(cOpt);

%% Plot
figure(3)
clf
hold on
[R, T, Z] = ndgrid(1:size(uOpt, 1), 1:size(uOpt, 2), 1:size(uOpt, 3));
S = uOpt(:);
S = 1*(S - min(S) + 100);
plot3DField(R(:), T(:), Z(:), S)
view(-30, 25)
colorbar
%% Plot tubes
cCoords = interpolateCurve(cOpt);
scatter3(cCoords(1, :), cCoords(2, :), cCoords(3, :), ...
         200, 'black', 'filled')
axis equal
colormap winter
colorbar
```

C.2 Gradient projection-algoritm

```
function [objOpt, cOpt] = gradProj(objective, c0)

% This function performs a gradient projection optimization with the given
% input objective function and initial point c0. The constraints of the
% problem are hard–coded below.

% ------ PARAMETERS ------
TOL_g = 0.0001; % The tolerance which determines whether the gradient of
                % the objective function is small enough to stop.
TOL_p = 0.0001; % The tolerance which determines whether the difference
                % between the previous point and the active point after
                % projection is small enough to stop.
% ------ END OF PARAMETERS ------

optimizeRadii = evalin('base', 'optimizeRadii');
minR = evalin('base', 'minR');
maxR = evalin('base', 'maxR');
minH = evalin('base', 'minH');
maxH = evalin('base', 'maxH');
doPlot = evalin('base', 'plotIterations');

global hasUserInterrupted
hasUserInterrupted = false;

% If doPlot, the values of each variable are plotted in figure 2 after each
% iteration.
if(doPlot)
    if (optimizeRadii)
        oldPlotVals = c0;
    else
        oldPlotVals = c0(length(c0)/2+1 : end);
    end
    figure(2)
    clf
    hold on
    grid on
    set(gcf, 'KeyPressFcn', @forceQuitFcn)
    drawnow
end

iteration = 0;  % Counter counting the number of iterations
cCurrent = c0;  % Variable containing the current active point
flag = false;   % Flag dermining whether an optimal point is reached
while(~flag)
    iteration = iteration + 1;
    stepLen = 0.005 / iteration;  % The step length used to find the next point
    disp(Starting iteration  + iteration)

    [objVal, grad] = objective(cCurrent, iteration);
    % Stop if gradient is small
    if(max(abs(grad)) < TOL_g)
        flag = true;
        disp('Gradient below tolerance, stopping')
        continue
    end

    cLen = length(c0);
    cNew = cCurrent - stepLen .* grad;  % Step along negative gradient
    c1h = cNew(cLen/2 + 1 : end);
    c1r = cNew(1 : cLen/2);

    % Clamp radii and heights to their allowed intervals; project into the
    % feasible set defined by the constraints
    % minR <= r_i <= maxR for all i,
    % minH <= h_i <= maxH for all i:
    c1h = max(minH, min(maxH, c1h));
    if (optimizeRadii)
        c1r = max(minR, min(maxR, c1r));
    end

    cProjected = [c1r, c1h];

    % Stop if the new point, after projection, is the original point
    if(max(abs(cProjected - cCurrent)) < TOL_p)
        flag = true;
        disp('Projected onto previous point, stopping')
        continue
    end

    if(doPlot)
        if (optimizeRadii)
```

```
            newPlotVals = cProjected;
        else
            newPlotVals = c1h;
        end
        for i=1:length(newPlotVals)
            x = [iteration-1, iteration];
            y = [oldPlotVals(i), newPlotVals(i)];
            c = mod(0.4*i, 1);
            figure(2)
            plot(x, y, 'color', [c, c, c], 'marker', '*')
        end
        oldPlotVals = newPlotVals;
        drawnow
    end

    cCurrent = cProjected;

    if(hasUserInterrupted)
        disp('User ended optimization')
        flag = true;
    end
end

objOpt = objVal;
cOpt = cCurrent;

end

function forceQuitFcn(~, ~)

    global hasUserInterrupted
    hasUserInterrupted = true;
end
```

```
function [val, grad] = objective(c, iteration)

optimizeRadii = evalin('base', 'optimizeRadii');
rLen = evalin('base', 'rLen');
zLen = evalin('base', 'zLen');
stepLenR = -0.005 / iteration;
stepLenH = 0.03 / iteration;

u = solvePDECyl(c);
val = avgNeckTemp(u);

% Curves c1 and c2 are found by c=[c1, c2]
grad = zeros(1, length(c));
% c1 height:
buildGradient(stepLenH, 0, zLen, length(c)/2 + 1, length(c));
if (optimizeRadii)
    % c1 radius:
    buildGradient(stepLenR, 0, rLen, 1, length(c)/2);
end

function buildGradient(stepLen, minVal, maxVal, varIndMin, varIndMax)
    for k=varIndMin:varIndMax
        cWithStep = c;
        newPoint = c(k) + stepLen;
        if(minVal < newPoint && newPoint < maxVal)
            cWithStep(k) = newPoint;
            uStep = avgNeckTemp(solvePDECyl(cWithStep));
            grad(k) = (uStep - val) / stepLen;
        else
            cWithStep(k) = c(k) - stepLen;
            uStep = avgNeckTemp(solvePDECyl(cWithStep));
            grad(k) = - (uStep - val) / stepLen;
        end
    end

end

end
```

```
function umax = avgNeckTemp(u)

% This function calculates the average temperature on the plane r=r0, i.e.
% the imagined skin on the patient's neck.

Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
tLen = evalin('base', 'tLen');
zLen = evalin('base', 'zLen');
```

```
11
12  fieldT1 =     tLen / 2;
13  fieldT2 = 3 * tLen / 4;
14  fieldZ1 =     zLen / 4;
15  fieldZ2 = 3 * zLen / 4;
16  t1 = round(fieldT1 / tLen * (Nt - 1));
17  t2 = round(fieldT2 / tLen * (Nt - 1));
18  z1 = round(fieldZ1 / zLen * (Nz - 1));
19  z2 = round(fieldZ2 / zLen * (Nz - 1));
20
21  uavg = 0;
22  for t=t1:t2
23      for z=z1:z2
24          uavg = uavg + u(Nr, t+1, z+1);
25      end
26  end
27  umax = uavg / (t2 - t1 + 1) / (z2 - z1 + 1);
28  % umax = mean(u(Nr, t1+1:t2+1, z1+1:z2+1), 'all');  % Only in R2018b and
29  % later
30
31  end
```

C.3 Finit differenslösare

```
1   function u = solvePDECyl(c)
2
3   % This function solves a finite difference heat equation using the
4   % successive over-relaxation iteration method.
5
6   % ------ PARAMETERS ------
7   TOL = 0.01;   % The tolerance which determines solver convergence. The
8                 % solver finishes if, for all points in the grid, the difference
9                 % between the new value and the previous iteration's value is
10                % less than this tolerance.
11  omega = 1.7;  % The weight parameter used in the successive over-relaxation.
12  % ------ END OF PARAMETERS ------
13
14  Nr = evalin('base', 'Nr');
15  Nt = evalin('base', 'Nt');
16  Nz = evalin('base', 'Nz');
17  rLen = evalin('base', 'rLen');
18  hr = evalin('base', 'hr');
19  ht = evalin('base', 'ht');
20  hz = evalin('base', 'hz');
21  hr2 = hr^2;
22  ht2 = ht^2;
23  hz2 = hz^2;
24  u = zeros(Nr, Nt, Nz);
25
26  % Generate the coordinates of all points that lie in water tubes:
27  tubeIndices = interpolateCurve(c);
28  tubeIndices = unique(tubeIndices', 'rows')' - [1; 1; 1];
29  % Construct all r,t,z such that no point lies on a tube:
30  interiorIndexTriples = [];
31  for r=1:(Nr-2)
32      for t=1:(Nt-2)
33          for z=1:(Nz-2)
34              if(ismember([r, t, z], tubeIndices', 'rows'))
35                  continue
36              end
37              interiorIndexTriples = [interiorIndexTriples, [r; t; z]];
38          end
39      end
40  end
41
42  % Set the boundary condition on the planes r=0:
43  for t=0:(Nt-1)
44      for z=0:(Nz-1)
45          u(1, t+1, z+1) = 20;
46      end
47  end
48  % Set the boundary condition on the planes t=0, t=tLen:
49  for r=0:(Nr-1)
50      for z=0:(Nz-1)
51          u(r+1, 1, z+1) = 20;
52          u(r+1, Nt, z+1) = 20;
53      end
54  end
55  % Set the boundary condition on the plane z=0, z=zLen:
56  for t=0:(Nt-1)
57      for r=0:(Nr-1)
58          u(r+1, t+1, 1) = 20;
```

```
        u(r+1, t+1, Nz) = 20;
    end
end

% Solve the heat equation iteratively using successive over-relaxation:
flag = true;  % flag which determines convergence
k = 0;  % counter which counts the number of iterations
aii = -2/hr2 - 2/ht2 - 2/hz2;  % helper function
while(flag)
    flag = false;
    k = k + 1;

    % Iterate over the interior of the domain
    for i=1:length(interiorIndexTriples)
        r = interiorIndexTriples(1, i);
        t = interiorIndexTriples(2, i);
        z = interiorIndexTriples(3, i);

        const = omega / aii;
        uold = u(r+1, t+1, z+1);
        unew = iterateInterior(u, [r+1, t+1, z+1], uold, omega, const);
        u(r+1, t+1, z+1) = unew;
        if(abs(uold - unew) > TOL)
            flag = true;
        end
    end

    % Iterate over all points on tubes
    for i=1:length(tubeIndices)
        r = tubeIndices(1, i);
        t = tubeIndices(2, i);
        z = tubeIndices(3, i);

        const = omega / aii;
        uold = u(r+1, t+1, z+1);
        unew = iterateTube(u, [r+1, t+1, z+1], uold, omega, const, tubeIndices);
        u(r+1, t+1, z+1) = unew;
        if(abs(uold - unew) > TOL)
            flag = true;
        end
    end

    for z=1:(Nr-2)
        for t=1:(Nt-2)
            const = omega / aii;
            uold = u(Nr, t+1, z+1);
            unew = iterateBoundary(u, [Nr, t+1, z+1], uold, omega, const);
            u(Nr, t+1, z+1) = unew;
            if(abs(uold - unew) > TOL)
                flag = true;
            end
        end
    end
end

nCalls = evalin('base', 'solverCalls') + 1;
assignin('base', 'solverCalls', nCalls);
disp(PDE solver finished after  + k +  iterations,  ...
    + nCalls + th call)

end
```

```
function unew = iterateInterior(u, x, uold, omega, const)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
k = evalin('base', 'k');

unew = (1-omega) * uold + const * ( ...
        -1/hr2 * u(x(1)-1, x(2),   x(3)) ...
        -1/hz2 * u(x(1),   x(2),   x(3)-1) ...
        -1/ht2 * u(x(1),   x(2)-1, x(3)) ...
        -1/ht2 * u(x(1),   x(2)+1, x(3)) ...
        -1/hz2 * u(x(1),   x(2),   x(3)+1) ...
        -1/hr2 * u(x(1)+1, x(2),   x(3)) ...
    );

```

```
end
```

```
function unew = iterateBoundary(u, x, uold, omega, const)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
k = evalin('base', 'k');

unew = (1-omega) * uold + const * ( ...
        ...-2*fluxTube / k / hz ... flux is 0
        -2/hr2 * u(x(1)-1, x(2), x(3)) ...
        -1/hz2 * u(x(1), x(2), x(3)-1) ...
        -1/ht2 * u(x(1), Nt, x(3)    ) ...
        -1/ht2 * u(x(1), x(2)+1, x(3)) ...
        -1/hz2 * u(x(1), x(2), x(3)+1) ...
        ...-1/hr2 * u(x(1)+1, x(2), x(3)) ...
    );

end
```

```
function unew = iterateTube(u, x, uold, omega, const, tubeIndices)

hr = evalin('base', 'hr');
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
hr2 = hr^2;
ht2 = ht^2;
hz2 = hz^2;
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
fluxTube = evalin('base', 'fluxTube');
k = evalin('base', 'k');

if(ismember([x(1)-2, x(2)-1, x(3)-1], tubeIndices', 'rows'))
    % If r+1 is hydrogel
    rightCoeff = 2/hr2;
    leftCoeff = 0;
    fluxR = -2*fluxTube/k/hr;
else
    % If r-1 is hydrogel
    rightCoeff = 0;
    leftCoeff = 2/hr2;
    fluxR = -2*fluxTube/k/hr;
end

if(ismember([x(1)-1, x(2)-1, x(3)-2], tubeIndices', 'rows'))
    % If z+1 is hydrogel
    upCoeff = 2/hz2;
    downCoeff = 0;
    fluxZ = -2*fluxTube/k/hz;
else
    % If z-1 is hydrogel
    upCoeff = 0;
    downCoeff = 2/hz2;
    fluxZ = -2*fluxTube/k/hz;
end

unew = (1-omega) * uold + const * ( ...
        fluxR + fluxZ ...
        -leftCoeff  * u(x(1)-1, x(2),   x(3)  ) ...
        -downCoeff  * u(x(1),   x(2),   x(3)-1) ...
        -1/ht2      * u(x(1),   x(2)-1, x(3)  ) ...
        -1/ht2      * u(x(1),   x(2)+1, x(3)  ) ...
        -upCoeff    * u(x(1),   x(2),   x(3)+1) ...
        -rightCoeff * u(x(1)+1, x(2),   x(3)  ) ...
    );
end
```

C.4 Övriga funktioner

```
function cCoords = interpolateCurve(c)

hr = evalin('base', 'hr');
```

```
ht = evalin('base', 'ht');
hz = evalin('base', 'hz');
Nr = evalin('base', 'Nr');
Nt = evalin('base', 'Nt');
Nz = evalin('base', 'Nz');
tLen = evalin('base', 'tLen');

% Approximate curve within grid:
% The first half of c contains all node radii, the second half contains all
% node heights; these are cr and cz below, respectively. The node angles
% are distributed equally from startT to endT radians; this is ct below.
ct = linspace(0, tLen, length(c)/2);
cr = c(1:(length(c)/2));
cz = c((length(c)/2+1):end);

% Convert (with rounding) node coordinates from meters to PDE domain g r i d
% indices:
cCoords0 = round([cr./hr; ct./ht; cz./hz]);

cCoords = [];
segmentLength = 2.5 * (Nt / length(ct)).^2;
for i=1:(length(ct)-1)
    p0 = cCoords0(:, i);   % segment start point
    p1 = cCoords0(:, i+1); % segment end point

    % Assume segment can cover max segmentLength grid points
    segment = [round(linspace(p0(1), p1(1), segmentLength));
               round(linspace(p0(2), p1(2), segmentLength));
               round(linspace(p0(3), p1(3), segmentLength))];

    % Don't include boundaries in tube:
    indicesToRemove = segment(2,:) == 0 | segment(2,:) == Nt-1;
    segment = segment(:, ~indicesToRemove);

    % t-indices are in 0,...,N-1, r and z and in 1,...,N. Fix:
    segment = segment + [0; 1; 0];
    % Make the water tube 2x2 in thickness; allowing for easier
    % implmenentation of Neumann boundary onditions:
    segmentRight = segment + [1; 0; 0];
    segmentUp = segment + [0; 0; 1];
    segmentUpRight = segment + [1; 0; 1];

    cCoords = [cCoords, segment, segmentRight, segmentUp, segmentUpRight];
end

end
```

D Gemensamma funktioner

```
function plot3DField(X0, Y, Z, D, ~)

if(nargin < 4)
    X = X0(:, 1);
    Y = X0(:, 2);
    Z = X0(:, 3);
    D = X0(:, 4);
else
    X = X0;
end

% scale = D / min(D);
scale = D;
if(nargin == 2 || nargin == 5)
    col = log((D - min(D)) + 1);
else
    col = D;
end

scatter3(X, Y, Z, scale, col, 'filled')
axis equal

end
```