CHALMERS |  GÖTEBORGS UNIVERSITET

Lösning av optimala styrproblem med finita elementmetoden

En studie i FEniCS

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Henrik Dahlström
Gustav Kettil
Sara Nilsson
Frida Svelander

Institutionen för matematiska vetenskaper
Chalmers tekniska högskola
Göteborgs universitet
Göteborg 2012

Lösning av optimala styrproblem med finita elementmetoden

En studie i FEniCS

Handledare: Stig Larsson
Examinator: Carl-Henrik Fant

Institutionen för matematiska vetenskaper
Chalmers tekniska högskola
Göteborgs universitet
Göteborg 2012

Sammanfattning

I den här studien implementeras en finita elementmetod för att lösa optimala styrproblem i programvaran FEniCS. Den finita elementmetod som används är hämtad från Karin Krafts doktorsavhandling *Adaptive Finite Element Methods for Optimal Control Problems* [2].

Optimala styrproblem behandlar styrning av dynamiska system. Systemen beskrivs av en tillståndsekvation och målet är att styra systemet mot ett visst tillstånd till en så låg kostnad som möjligt. För detta syfte introduceras en målfunktional som mäter avvikelsen från målet och även inkluderar kostnaden för styrning. Det optimala styrproblemet löses genom att minimera målfunktionalen med systemets tillståndsekvation som bivillkor.

I Krafts avhandling utnyttjas Lagranges metod och bivillkoret skrivs på variationsform och adderas till målfunktionalen. Därmed fås en funktional utan bivillkor att minimera. Variationskalkyl används sedan för att nå en svag formulering som slutligen löses med finita elementmetoden.

Syftet med studien är att undersöka hur väl lämpat FEniCS är för att hantera en finita elementmetod lik den Kraft använder, vilken bygger på en kombination av kontinuerliga och diskontinuerliga funktionsrum. De diskontinuerliga funktionsrummen innefattar i Krafts avhandling två yttre randvärden. Dessa finns inte i de funktionsrum som FEniCS tillhandahåller, vilket leder till komplikationer.

Komplikationerna kringgås genom att justera den matrisekvation som finita elementmetoden resulterar i. Detta kräver extra arbete, men uppvägs av FEniCS fördelar. Till exempel är det enkelt att ändra gradtalet hos baspolynomen i de funktionsrum som används, samt att variera definitionsintervallet och dess partitionering. Ytterligare en fördel är att konstruktionen av matrisekvationen automatiseras. Från variationsformuleringen och de valda funktionsrummen ger FEniCS en finita elementlösning till problemet.

Abstract

In this study a finite element method for solving optimal control problems is implemented using a software called FEniCS. The finite element method is presented in Karin Kraft's thesis *Adaptive Finite Element Methods for Optimal Control Problems* [2].

Optimal control problems deal with dynamical systems. These systems are described by a state equation. The aim is to steer the system towards a certain state at minimal cost. For this purpose a goal functional including deviations from the goal is introduced. The optimal control problem is solved by minimizing the cost functional subject to a constraint in the form of a differential equation describing the path of the system.

In Kraft's thesis Lagrange's method is used to reformulate the problem. The constraints are written in variational form and added to the goal functional, which gives a goal functional to minimize without constraints. The variational formulation of the goal functional is reached by applying a theorem relating the optimum of a functional to it's partial derivatives. The finite element problem to be studied is then acquired from this variational formulation.

The purpose of this study is to analyze the suitability of FEniCS in solving an optimal control problem like the one Kraft uses, which is based on a combination of continuous and discontinuous function spaces. In Kraft's thesis, the discontinuous function spaces include two external boundary limits at the endpoints of the interval. These are not included in the function spaces that are predefined in FEniCS, which results in complications.

These complications are avoided by manually adjusting the matrix equation resulting from the finite element method, but this extra effort is compensated for by the advantages of FEniCS. As an example, the degree of the basis polynomials is easily changed, as well as the domain and partition. Another advantage is the automatic construction of the finite element matrix equation, acquired from the variational formulation of the problem and the chosen function spaces.

Förord

Detta projekt uppkom ur ett intresse hos forskare vid Chalmers tekniska högskola av att se om FEniCS är en lämplig programvara för lösning av optimala styrproblem med en viss finita elementmetod. Bidragande till projektet och denna rapport har varit Henrik Dahlström, Gustav Kettil, Sara Nilsson och Frida Svelander, civilingenjörsstudenter vid Chalmers. En projektdagbok och en tidslogg över varje deltagares individuella bidrag har förts kontinuerligt under arbetsprocessen. De fyra deltagarna kan tillskrivas olika ansvarsområden och insatser, vilka beskrivs nedan.

Alla fyra har läst på för att sätta sig in i grunderna i finita elementmetoden och hur FEniCS fungerar. Mer specifikt gick projektet ut på att i FEniCS implementera den finita elementmetod för lösning av optimala styrproblem som Karin Kraft presenterar i sin doktorsavhandling *Adaptive Finite Element Methods for Optimal Control Problems* [2]. Därför har det även varit nödvändigt att grundligt sätta sig in i Krafts metod. För det arbetet har Gustav Kettil och Frida Svelander huvudsakligen stått.

Vad gäller rapporten har allt som rör inledningen och de texter som föregår den skrivits gemensamt. Detsamma gäller diskussion och slutsats. Övriga delar av rapporten har olika huvudförfattare. Kapitel 2, som beskriver grunderna i finita elementmetoden, har i huvudsak skrivits av Sara Nilsson. Hon har lagt mycket tid på att studera Galerkins metod, främst med *Computational Differential Equations* av K. Eriksson med flera [1] som källa.

Henrik Dahlström är huvudförfattare till nästföljande kapitel, vilket är en FEniCS-manual som beskriver centrala metoder och klasser för de program som skrivits i detta projektarbete. Han har därmed fokuserat extra på att förstå hur FEniCS fungerar och kan användas i detta sammanhang.

Kapitel 3 beskriver Krafts finita elementmetod och har delats mellan två författare. Gustav Kettil är ansvarig för allt som inte innefattar Sats 3.1 med tillhörande bevis och den notation för partiella derivator som införs precis innan satsen. För dessa delar har Frida Svelander stått.

I kapitel 5 är Frida Svelander författare till introduktionen och de två första exemplen, det vill säga avsnitten 5.1 och 5.2. Gustav Kettil är ansvarig för det sista exemplet som presenteras i avsnitt 5.3, vilket är ett fullständigt optimalt styrproblem. Gustav är den som kommit med allra flest idéer och i huvudsak arbetat med programkoden för lösning av det fullständiga optimala styrproblemet, medan Frida har liknande ansvar för programmet i avsnitt 5.2. FEniCS-programmet i avsnitt 5.1 kan tillskrivas alla medverkande i projektet. Varje delavsnitt i kapitel 5 innehåller förutom programkod en teoretisk del. Denna är författaren till respektive delavsnitt, Gustav eller Frida, ansvarig för. De har alltså behövt sätta sig in i den matematiska teori som ligger bakom respektive exempel.

Nämnas bör att alla deltagare har satt sig in grundligt i FEniCS, även om de inte haft huvudansvaret för något programexempel. Vissa projektmedlemmar är alltså i första hand ansvariga för de sammansatta programmen, men alla har varit med och letat metoder och klasser för att göra programmen möjliga. Henrik Dahlström har till exempel jobbat en hel del med detta.

Appendix A är i huvudsak skrivet av Gustav Kettil och Frida Svelander.

Den grundliga planeringen av projektet har alla fyra medverkande gjort tillsammans i form av en skriftlig planeringsrapport, till vilken alla har bidragit med likvärdiga insatser. Det huvudsakliga planerandet under arbetsprocessen har Frida Svelander stått för. Dock har alla medverkande varit aktiva under projektets gång.

Till sist vill vi passa på att tacka Stig Larsson, vår handledare i detta projekt, för all hjälp och stort engagemang. Vi vill även tacka Matteo Molteni, som varit delaktig jämte Stig.

Innehåll

1 Inledning

1.1 Bakgrund

Optimala styrproblem kan matematiskt beskriva situationer inom vitt skilda områden så som logistik, medicin och ekonomi. Det kan handla om ett fordon som ska ta sig fram till en specifik plats med minimal bränsleförbrukning, eller att någon vill veta när och hur mycket som ska investeras i en resurs för att få maximal avkastning. Ett annat tänkbart scenario är att ett vaccin ska fördelas över en epidemidrabbad befolkning på ett så effektivt sätt som möjligt [5].

I doktorsavhandlingen *Adaptive Finite Element Methods for Optimal Control Problems* [2] författad av Karin Kraft redovisas en numerisk lösningsmetod för optimala styrproblem. Där utnyttjas Lagranges metod för att omvandla ett optimeringsproblem till ett system av differentialekvationer. Systemet löses sedan med finita elementmetoden, vilket är okonventionellt vid lösning av ordinära differentialekvationer. Traditionellt används andra metoder, såsom finita differensmetoder.

Metoderna för att numeriskt lösa differentialekvationer är många, liksom antalet olika mjukvaror som kan användas för att implementera dessa. FEniCS är en programvara för lösning av partiella differentialekvationer med finita elementmetoder. Lanseringen av FEniCS skedde under 2011, vilket gör programvaran relativt ny. Av denna anledning kan det vara av intresse att undersöka om Krafts metod i kombination med FEniCS passar för lösning av optimala styrproblem.

1.2 Syfte

Projektets syfte är att i FEniCS implementera den finita elementmetod Kraft använder för lösning av optimala styrproblem. Lösningarna i sig är sekundära, då det intressanta är att se hur väl lämpat FEniCS är för att hantera en finita elementmetod av denna typ.

1.3 Avgränsningar

För att kunna jämföra metoden i teorin med vad som sker i FEniCS begränsar vi oss till kvadratisk-linjära optimala styrproblem. Detta för att de matrisekvationer som finita elementmetoden i Krafts avhandling leder fram till då är relativt enkla att beräkna. Dessa kan sedan jämföras med de matriser som FEniCS ger. Dessutom skulle lösningen av icke-linjära problem kräva mer avancerade metoder. Att bara titta på kvadratisk-linjära styrproblem är dock ingen större inskränkning, då många naturliga system går att beskriva på denna form.

1.4 Tillvägagångssätt

Målet med denna studie är alltså att implementera den finita elementmetod Kraft använder i FEniCS. Därför faller det sig naturligt att utgå ifrån hennes doktorsavhandling för att få kunskap om den teori som behövs och som kommer presenteras i den här rapporten. Av praktiska skäl används i teoriavsnitten en notation som liknar den i Krafts avhandling.

Kraft använder matematisk teori och en finita elementmetod som inte är helt grundläggande. Därför är det nödvändigt att grunderna i variationskalkyl och numerisk lösning av differentialekvationer med finita elementmetoden studeras och presenteras. Krafts avhandling och boken *Computational Differential Equations* av K. Eriksson med flera [1] fungerar som huvudsaklig källa till detta.

För att implementera metoderna krävs kunskap om programvaran FEniCS och om programspråket Python, vilket är ett av språken som används i FEniCS. I huvudsak används två källor för information om hur programmen kan skrivas. Dessa är *Automated Solution of Differential Equations by the Finite Element Method, The FEniCS Book* av A. Logg med flera [15] och dokumentationen på FEniCS Projects hemsida [16].

1.5 Rapportens innehåll

Målet med rapporten är att presentera hur finita elementmetoden från Krafts avhandling kan implementeras i FEniCS och hur väl lämpad programvaran är för detta. På vägen dit kommer en del grundläggande matematisk teori bakom metoden att gås igenom och några delproblem kommer att lösas. Problemen finns med för att demonstrera hur FEniCS fungerar och hur programvaran kan användas för projektets syfte.

Först presenteras ett avsnitt om finita elementmetoden, en numerisk metod för lösning av differentialekvationer. Metoden går ut på att finna en approximativ lösning till den ursprungliga ekvationen i ett funktionsrum bestående av styckvisa polynomfunktioner. Kapitlet finns med för att beskriva den grundläggande matematiken bakom den finita elementmetod som Kraft presenterar i sin doktorsavhandling.

Därefter följer ett avsnitt om optimala styrproblem. Här introduceras de optimala styrproblem som ska implementeras i FEniCS, samt den lösningsmetod som Kraft beskriver. Avsnittet är teoretiskt och inte kopplat till programvaran.

Efter kapitlet om optimala styrproblem följer en FEniCS-manual. Detta för att öka förståelsen för den Pythonkod som presenteras i avsnitt 5. Manualen är en alfabetiskt ordnad lista över de viktigaste funktioner och objekt som används i programkoden.

I kapitel 5 presenteras tre matematiska problem som är kopplade till lösning av optimala styrproblem med Krafts finita elementmetod. Både teorin bakom problemen och hur de kan lösas numeriskt med hjälp av FEniCS redovisas. Det första exemplet är grundläggande i finita elementsammanhang, men inte av samma typ som de optimala styrproblemen. Det är dock relevant eftersom det kan härledas från ett minimeringsproblem precis som de optimala styrproblemens finita elementformulering. Det andra exemplet är ett förenklat optimalt styrproblem på formen av ett begynnelsevärdesproblem. Slutligen följer ett fullständigt optimalt styrproblem på kvadratisk-linjär form, vilket målet med detta arbete är att lösa.

Sist presenteras ett diskussionsavsnitt med efterföljande slutsats, där vi knyter an till syftet och analyserar hur väl lämpat FEniCS är för implementation av Krafts finita elementmetod. Förslag på fortsatta studier ges också.

2 Finita elementmetoden

I det här kapitlet introduceras Galerkins metod, en numerisk metod för lösning av differentialekvationer. Detta utgör grunden för den matematik som används i senare kapitel. Informationen har huvudsakligen hämtats från *Computational Differential Equations* av K. Eriksson med flera [1] samt Mohammad Asadzadehs *An introduction to the Finite Element Method (FEM) for Differential Equations* [4].

I avsnitt 2.1 introduceras några funktionsrum som vanligen används när en lösning till en differentialekvation approximeras med Galerkins metod. Metoden i sig presenteras i avsnitt 2.2.

De två efterföljande avsnitten, 2.3 och 2.4, behandlar två exempel som löses med Galerkins metod. Det första problemet är en linjär differentialekvation av grad två med homogena randvillkor som approximeras med en linjärkombination av styckvis linjära polynom. I det andra demonstreras en diskontinuerlig Galerkinmetod, där ansatsfunktion och testfunktion tillhör olika rum.

2.1 Basfunktioner

En godtycklig funktion kan approximeras med en linjärkombination av basfunktioner för ett vektorrum. I finita elementmetoden approximeras den exakta lösningen till en differentialekvation på detta sätt. Nedan presenteras en vanlig bas i finita elementsammanhang, Lagrangebasen.

Låt en kontinuerlig funktion $f(x)$ vara definierad på intervallet $(a,\, b)$. En partitionering görs enligt:

$$\mathcal{T} : a = x_0 < x_1 < \ldots < x_{N-1} < x_N = b, \tag{2.1}$$

där delintervallen betecknas $I_i = (x_{i-1}, x_i)$ och intervallängden definieras som $h_i = x_i - x_{i-1}$.

För styckvisa polynom av grad q på $(a,\, b)$, $\mathcal{P}^q(a,\, b)$, är den så kallade Lagrangebasen avändbar. På varje delintervall definieras den som $\{\lambda_i(x)\}_{i=0}^{q}$, enligt:

$$\lambda_i = \frac{(x-\xi_0)(x-\xi_1)\ldots(x-\xi_{i-1})(x-\xi_{i+1})\ldots(x-\xi_q)}{(\xi_i-\xi_0)(\xi_i-\xi_1)\ldots(\xi_i-\xi_{i-1})(\xi_i-\xi_{i+1})\ldots(\xi_i-\xi_q)} = \prod_{j\neq i}\frac{x-\xi_j}{\xi_i-\xi_j},$$

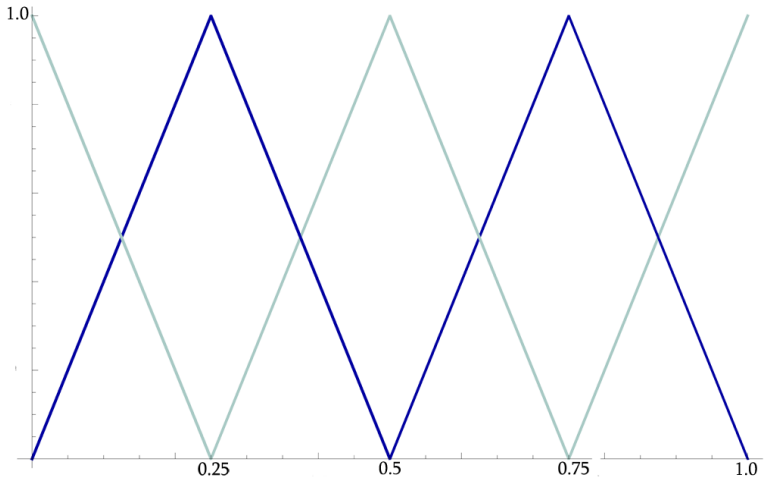
där $q+1$ punkter ξ_i partitionerar I_i. Nedan presenteras hur denna bas ser ut på partitioneringen $\mathcal{T}$ för polynom av grad noll, ett och två. För $q = 0$:

$$\varphi_i(x) = \begin{cases} 1, & x \in (x_{i-1},\, x_i), \quad i = 1, 2, ..., N, \\ 0, & \text{annars}, \end{cases}$$

och för $q = 1$, se Figur 2.1:

$$\varphi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h_i}, & x \in (x_{i-1}, x_i), \\ \dfrac{x_{i+1} - x}{h_{i+1}}, & x \in (x_i, x_{i+1}), \quad i = 1, 2, ..., N-1, \\ 0, & \text{annars}. \end{cases}$$

För $q = 2$ väljs den tredje noden på varje intervall som punkten mitt emellan intervallrandpunkterna: $x_{i-\frac{1}{2}} = \dfrac{x_i + x_{i-1}}{2}$. Detta ger följande basfunktioner som ritats ut i Figur

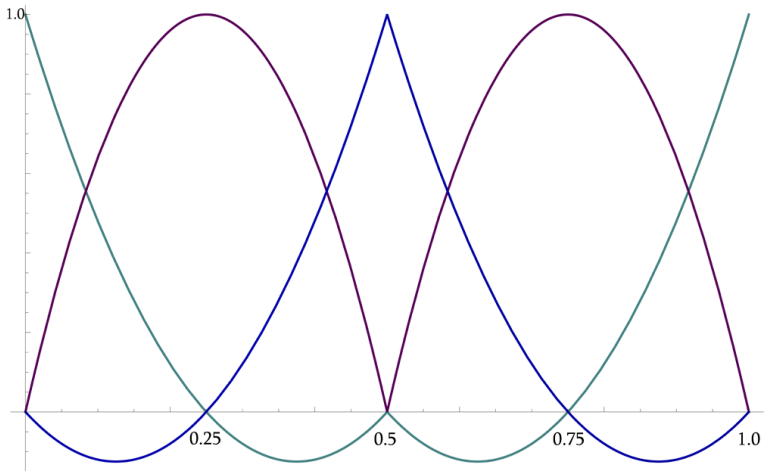


Figur 2.1: Lagrangebasfunktioner av grad 1 med ekvidistant nodavstånd 0.25.

2.2:

$$\varphi_{i-\frac{1}{2}}(x) = \begin{cases} \dfrac{4(x_i - x)(x - x_{i-1})}{h_i^2}, & x \in (x_{i-1}, x_i), \quad i = 1, 2, ..., N, \\ 0, & \text{annars}, \end{cases}$$

$$\varphi_i(x) = \begin{cases} \dfrac{2(x - x_{i+\frac{1}{2}})(x - x_{i+1})}{h_{i+1}^2}, & x \in (x_i, x_{i+1}), \quad i = 1, 2, ..., N-1, \\ \dfrac{2(x - x_{i-\frac{1}{2}})(x - x_{i-1})}{h_i^2}, & x \in (x_{i-1}, x_i), \\ 0, & \text{annars}. \end{cases}$$



Figur 2.2: Lagrangebasfunktioner av grad 2 med ekvidistant nodavstånd 0.25.

Dessa uppfyller alla villkoret $\varphi_i(\xi_j) = \delta_{ij}$ för en nodbas[1], som ger följande uttryck för approximationen av $f(x)$ i detta funktionsrum:

$$f(x) \approx \sum_i f(\xi_i)\varphi_i(x). \tag{2.2}$$

[1] För definition av Kroneckers delta se Appendix A.1.

För $q = 1$ och $q = 2$ fås ytterligare två basfunktioner, $\varphi_0(x)$ och $\varphi_N(x)$. Dessa skiljer sig från övriga baspolynom, på så sätt att de saknar den del som hamnar utanför intervallet i definitionen av de inre polynomen. De extra basfunktionerna ser alltså ut som följer för $q = 1$:

$$\varphi_0(x) = \begin{cases} \dfrac{x_1 - x}{h_1}, & x \in (x_0,\, x_1), \\ 0, & \text{annars}, \end{cases}$$

$$\varphi_N(x) = \begin{cases} \dfrac{x - x_{N-1}}{h_N}, & x \in (x_{N-1},\, x_N), \\ 0, & \text{annars}. \end{cases}$$

Enligt samma princip fås för $q = 2$.

$$\varphi_0(x) = \begin{cases} \dfrac{2(x - x_{\frac{1}{2}})(x - x_1)}{h_1^2}, & x \in (x_0,\, x_1), \\ 0, & \text{annars}, \end{cases}$$

$$\varphi_N(x) = \begin{cases} \dfrac{2(x - x_{N-\frac{1}{2}})(x - x_{N-1})}{h_N^2}, & x \in (x_{N-1},\, x_N), \\ 0, & \text{annars}. \end{cases}$$

Dessa är inkluderade i Figur 2.1 och 2.2.

2.2 Galerkins metod

En finita elementmetod är en numerisk metod konstruerad för att hitta approximativa lösningar till differentialekvationer. Galerkins metod är en av dessa finita elementmetoder, som omvandlar ett kontinuerligt problem till en diskret matrisekvation.

För att demonstrera Galerkins metod betraktas här differentialekvationen:

$$\mathscr{L}\left(u(x)\right) = f(x), \qquad a < x < b, \tag{2.3}$$

där $\mathscr{L}$ är en linjär differentialoperator och $f(x)$ är en känd funktion. Ekvationen multipliceras med en så kallad testfunktion v tillhörande ett funktionsrum V och integreras över $(a,\, b)$. Problemet är nu att hitta en funktion u i ett funktionsrum W sådan att följande ekvation gäller:

$$\int_a^b \mathscr{L}\left(u(x)\right) v(x)\, dx = \int_a^b f(x) v(x)\, dx, \quad \forall v \in V. \tag{2.4}$$

Rummen V och W är i många fall desamma, men det är inte nödvändigt. Formuleringen i (2.4) kallas *variationsformuleringen* eller den svaga formuleringen av (2.3). Formuleringen kallas svag för att den bara kräver att ekvationen är uppfylld för alla testfunktioner v i ett begränsat funktionsrum.

Flyttas sedan alla termer till vänsterledet fås villkoret:

$$\int_a^b \left(\mathscr{L}(u(x)) - f(x)\right) v(x)\, dx = 0, \quad \forall v \in V,$$

som leder fram till den så kallade Galerkinortogonaliteten, vilken säger att

$$\left(\mathscr{L}(u(x)) - f(x)\right) \perp v(x), \quad \forall v \in V.$$

Alltså är $\mathscr{L}(u(x)) - f(x)$ residualfelet för approximationen.

Om en av polynombaserna som definierades i kapitel 2.1 väljs, kan den approximativa lösningen skrivas som

$$U(x) = \sum_i \zeta_i \varphi_i(x), \tag{2.5}$$

där i löper över alla basfunktioner och ζ_i är de konstanter som söks.

För att få fram $U(x)$ behöver alltså alla ζ_i beräknas, vilket görs genom att utnyttja villkoret

$$\int_a^b \left(\mathscr{L}\left(U(x)\right) - f(x)\right)\varphi_i(x)\,dx = 0, \quad \forall i. \tag{2.6}$$

Här har utnyttjats att det räcker att testa mot alla φ_i istället för alla v, eftersom de bildar en bas för V. Ekvationen ger inte den exakta lösningen, men ger den bästa lösningen i det rum där approximationen söks.

Beroende på hur $\mathscr{L}\left(U\right)$ ser ut kommer lösningsmetoden variera, men oavsett skrivs $U(x)$ som en linjärkombination av funktionerna $\varphi_i(x)$ som ovan och koefficienterna ζ_i beräknas. Då hela summan som utgör $U(x)$ måste multipliceras med alla $\varphi_i(x)$ fås ett ekvationssystem av samma storlek som antalet basfunktioner, vilket kan skrivas om som en kvadratisk matrisekvation. Lösningsmetoden gås igenom i detalj för två olika typer av problem i avsnitt 2.3 och 2.4 nedan.

2.3 Ett grundläggande exempel

Här demonstreras Galerkins finita elementmetod för följande randvärdesproblem:

$$\begin{cases} -u''(x) = x, \quad 0 < x < 1, \\ u(0) = 0, \quad u(1) = 0. \end{cases} \tag{2.7}$$

Först skrivs problemet på variationsform. För detta syfte introduceras funktionsrummet

$$V = \left\{ v : \int_0^1 (v^2 + (v')^2)\,dx \, < \infty,\, v(0) = v(1) = 0 \right\}.$$

Ekvationen i (2.7) multipliceras med en testfunktion $v \in V$ och integreras över $(0,1)$, enligt:

$$\begin{aligned} -\int_0^1 u''(x)v(x)\,dx &= -\left[u'(x)v(x)\right]_0^1 + \int_0^1 u'(x)v'(x)\,dx \\ &= \int_0^1 u'(x)v'(x)\,dx = \int_0^1 xv(x)\,dx. \end{aligned}$$

Här har partiell integration och det faktum att $v(0) = v(1) = 0$ använts. Variationsformuleringen till (2.7) lyder nu: finn $u \in V$ som uppfyller

$$\int_0^1 u'(x)v'(x)\,dx = \int_0^1 xv(x)\,dx, \quad \forall v \in V. \tag{2.8}$$

Sedan görs en partitionering av intervallet $(0,1)$ i fyra lika långa delintervall I_i, alltså med $h_i = h = 0.25$ för alla i. För finita elementformuleringen av problemet införs ett ändligdimensionellt funktionsrum $V_h \subset V$.

$$V_h = \left\{ v \in C(0,1) : \, v|_{I_i} \in \mathcal{P}^1(I_i), v(0) = v(1) = 0 \right\}.$$

Finita elementproblemet fås genom att låta V_h ersätta V i variationsformuleringen (2.8). Problemet blir att finna en funktion $U(x) = \sum_i \zeta_i \varphi_i(x) \in V_h$ sådan att

$$\int_0^1 U'(x)v'(x)\,dx = \int_0^1 xv(x)\,dx, \quad \forall v \in V_h.$$

Som bas för V_h väljs Lagrangepolynomen $\{\varphi_i\}_{i=1}^3$ av grad 1, där de yttersta polynomen uteslutits på grund av de homogena randvillkoren. Definitionen av basfunktionerna är som i

avsnitt 2.1:

$$\varphi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h_i}, & x \in (x_{i-1}, x_i), \\ \dfrac{x_{i+1} - x}{h_{i+1}}, & x \in (x_i, x_{i+1}), \quad i = 1, 2, 3, \\ 0, & \text{annars.} \end{cases}$$

Eftersom $\{\varphi_i\}_{i=1}^3$ är en bas räcker det att testa mot dessa i v:s ställe. Genom att göra det fås ett ekvationssystem av storlek 3×3, vilket ger en kvadratisk matrisekvation på formen $A\zeta = b$, där

$$\zeta = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix}$$

är koefficienterna till den diskreta finita elementlösning som söks.

För att beräkna alla matriselement explicit behövs $\varphi_i'(x)$:

$$\varphi_i'(x) = \begin{cases} \dfrac{1}{h_i}, & x \in (x_{i-1}, x_i), \\ \dfrac{-1}{h_{i+1}}, & x \in (x_i, x_{i+1}), \quad i = 1, 2, 3, \\ 0, & \text{annars.} \end{cases}$$

Elementen på diagonalen blir som följer:

$$\begin{aligned} \int_0^1 \varphi_i' \varphi_i' \, dx &= \int_{x_{i-1}}^{x_i} \frac{1}{h_i} . \frac{1}{h_i} \, dx + \int_{x_i}^{x_{i+1}} \frac{-1}{h_{i+1}} . \frac{-1}{h_{i+1}} \, dx = \\ &= \frac{1}{h_i} + \frac{1}{h_{i+1}} = 8. \end{aligned}$$

Den övre diagonalen ($j = i + 1$) ser ut som:

$$\int_0^1 \varphi_i' \varphi_{i+1}' \, dx = \int_{x_i}^{x_{i+1}} \frac{-1}{h_{i+1}} . \frac{1}{h_{i+1}} \, dx = \frac{-1}{h_i} = -4.$$

Den lägre diagonalen ($j = i - 1$) har elementen:

$$\int_0^1 \varphi_i' \varphi_{i-1}' \, dx = \int_{x_{i-1}}^{x_i} \frac{1}{h_i} . \frac{-1}{h_i} \, dx = \frac{-1}{h_i} = -4.$$

Matrisen blir alltså:

$$A = \begin{bmatrix} 8 & -4 & 0 \\ -4 & 8 & -4 \\ 0 & -4 & 8 \end{bmatrix}.$$

Högerledet beräknas också enligt:

$$\begin{aligned} b_i(x) &= \int_{x_{i-1}}^{x_i} x \frac{x - x_{i-1}}{h_i} \, dx + \int_{x_i}^{x_{i+1}} x \frac{x_{i+1} - x}{h_{i+1}} \, dx \\ &= \left[\frac{x^3}{3h_i} - \frac{x^2 x_{i-1}}{2h_i} \right]_{x_{i-1}}^{x_i} + \left[\frac{x^2 x_{i+1}}{2h_{i+1}} - \frac{x^3}{3h_{i+1}} \right]_{x_i}^{x_{i+1}} \\ &= \frac{2x_i^3}{3h} - \frac{x_i^2 (x_{i+1} + x_{i-1})}{2h} + \frac{x_{i+1}^3 + x_{i-1}^3}{6h}, \end{aligned}$$

vilket numeriskt ger b som:

$$b = \begin{bmatrix} 0.0625 \\ 0.1250 \\ 0.1875 \end{bmatrix}.$$

Löses denna matrisekvation fås koefficienterna ζ_i:

$$\zeta = \begin{bmatrix} 0.0391 \\ 0.0625 \\ 0.0547 \end{bmatrix},$$

vilket ger finita elementlösningen till (2.7) på samma form som ekvation 2.5.

2.4 Ett exempel med diskontinuerlig ansatsfunktion

Detta exempel ger en finita elementlösning till följande initialvärdesproblem:

$$\begin{cases} u'(x) = u(x), \quad 0 < x < 1, \\ u(0) = 1. \end{cases}$$

Här kommer rummet av testfunktioner bestå av kontinuerligt styckvisa polynom av grad 1 och ansatsfunktionen vara styckvis konstant med avseende på partitioneringen

$$\mathcal{T} : 0 = x_0 < x_1 < \ldots < x_{N-1} < x_N = 1,$$

där delintervallen betecknas $I_i = (x_{i-1},\, x_i)$. Att använda basfunktioner av olika grad är inget problem så länge det ger samma antal basfunktioner i test- och ansatsrummet, så att den senare matrisekvationen blir kvadratisk.

För att kompensera för att en kontinuerlig funktion approximeras med styckvis konstanta funktioner införs $[u_i] = u_i^+ - u_i^-$, där $u_i^+ = \lim_{x\to 0}(x_i + x)$ och $u_i^- = \lim_{x\to 0}(x_i - x)$.

Nu definieras rummet av testfunktioner som

$$V_h = \left\{ v \in C(0,1) :\, v|_{I_i} \in \mathcal{P}^1(I_i), v(0) = v(1) = 0 \right\}.$$

En summa som tar hänsyn till att ansatsfunktionen är diskontinuerlig införs, och Galerkinmetoden lyder:

$$\sum_{i=1}^{N} \int_{I_i} (U'(x) - U(x))v(x)\, dx + \sum_{i=0}^{N} [U_i]v(x_i) = 0, \quad \forall v \in V. \tag{2.9}$$

Då U ska vara styckvis konstant gäller att $U' \equiv 0$. Av samma anledning kan vi låta $U_i = U_i^- = U_{i-1}^+$.

I nodpunkterna gäller att $\varphi_i = 1$, vilket också förenklar uttrycket. Samma partitionering väljs som i exempel 2.3 ovan, alltså $h_i = 0.25, \forall i$. Då fås en matrisekvation $A\vec{U} = b$, med $dim(A) = 6 \times 6$, och

$$\vec{U} = \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{bmatrix}$$

är finita elementlösningen till problemet.

Genom att testa mot varje basfunktion φ_i istället för v och ta hänsyn till begynnelsevillkoret, fås 6 ekvationer i 6 variabler:

$$\begin{aligned}
U_0 &= 1, \\
-U_1 \int_{I_1} \varphi_0 \, dx + (U_1 - U_0) &= 0, \\
-U_1 \int_{I_1} \varphi_1 \, dx - U_2 \int_{I_2} \varphi_1 \, dx + (U_2 - U_1) &= 0, \\
-U_2 \int_{I_2} \varphi_2 \, dx - U_3 \int_{I_3} \varphi_3 \, dx + (U_3 - U_2) &= 0, \\
-U_3 \int_{I_3} \varphi_3 \, dx - U_4 \int_{I_4} \varphi_3 \, dx + (U_4 - U_3) &= 0, \\
-U_4 \int_{I_4} \varphi_4 \, dx + (U_5 - U_4) &= 0.
\end{aligned}$$

Varje integral antar värdet $\frac{h}{2} = 0.125$, så matrisekvationen blir

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0.875 & 0 & 0 & 0 & 0 \\ 0 & -1.125 & 0.875 & 0 & 0 & 0 \\ 0 & 0 & -1.125 & 0.875 & 0 & 0 \\ 0 & 0 & 0 & -1.125 & 0.875 & 0 \\ 0 & 0 & 0 & 0 & -1.125 & 1 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Detta ger koefficienterna U_i som:

$$\vec{U} = \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 1.1429 \\ 1.4694 \\ 1.8892 \\ 2.4290 \\ 2.7326 \end{bmatrix}.$$

Notera att funktionen bara antar värdena U_0 och U_5 i randpunkterna, och inte på några intervall.

3 Optimala styrproblem

I det här kapitlet redovisas den matematiska teorin för optimala styrproblem. Teorin är hämtad från Krafts doktorsavhandling *Adaptive Finite Element Methods for Optimal Control Problems* [2]. Först definieras optimala styrproblem. Sedan beskrivs hur ett system av differentialekvationer härleds från det optimala styrproblemet. Detta görs med hjälp av Lagranges metod och variationskalkyl. Därefter introduceras en finita elementmetod för att lösa ekvationssystemet. Sist presenteras optimala styrproblem skrivna på kvadratisk-linjär form, vilken är den typ av optimala styrproblem vi studerar i detta projekt.

3.1 Definition av optimala styrproblem

Som utgångspunkt för definitionen av optimala styrproblem står ett dynamiskt system. Systemet beskrivs av sitt tillstånd $x(t) \in \mathbb{R}^d$, där $t \in (0, T)$ betecknar tiden. Systemets styrfunktion $u(t) \in \mathbb{R}^m$ relateras till $x(t)$ genom tillståndsekvationen

$$\begin{cases} \dot{x}(t) = f(x(t), u(t)), \\ I_0 x(0) = x_0, \quad I_T x(T) = x_T, \end{cases}$$

där $I_0, I_T \in \mathbb{R}^{d\times d}$ är binärt diagonala[2]. Målet är att styra systemet mot ett visst tillstånd till en så låg kostnad som möjligt. För detta syfte införs en målfunktional[3] $\mathcal{J}(x, u)$ som beskriver avvikelsen från målet samt kostnaden för styrningen.

Sammanfattningsvis följer nedanstående definition av ett optimalt styrproblem [2]:

Definition 3.1. *Följande problem är ett* ***optimalt styrproblem****: hitta $x(t)$ och $u(t)$ som*

$$\textit{minimerar} \quad \mathcal{J}(x(t), u(t)) = l(x(0), x(T)) + \int_0^T L(x(t), u(t))\, dt, \tag{3.1}$$

$$\textit{under bivillkoret} \quad \begin{cases} \dot{x}(t) = f(x(t), u(t)), \quad 0 < t < T, \\ I_0 x(0) = x_0, \quad I_T x(T) = x_T, \end{cases} \tag{3.2}$$

där

$$\begin{aligned} l &: \ \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \\ L &: \ \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}, \\ f &: \ \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d \end{aligned}$$

är glatta funktioner[4] *och $I_0, I_T \in \mathbb{R}^{d\times d}$ är binärt diagonala.*

3.2 Lösning med finita elementmetoden

I detta avsnitt presenteras en lösningsmetod som baseras på en finita elementmetod. Metoden hämtas från Krafts doktorsavhandling [2].

Givet är alltså ett optimalt styrproblem på samma form som i Definition 3.1. Kortfattat går lösningsproceduren ut på att föra in bivillkor (3.2) på variationsform i funktionalen $\mathcal{J}(x, u)$ med hjälp av en Langrangemultiplikator[5]. Detta ger upphov till en ny funktional utan bivillkor och då är det nödvändiga villkoret för minimum att derivatan är noll. Derivering av funktionalen leder till tre ekvationer på variationsform som löses med finita elementmetoden.

[2] För definition av binärt diagonal matris se Appendix A.3.
[3] För definition av funktional se Appendix A.2.
[4] För definition av glatt funktion se Appendix A.4.
[5] För definition av Lagrangemultiplikator se Appendix A.5.

3.2.1 Abstrakt formulering

För partioneringen $0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = T$ där $I_n = (t_{n-1}, t_n)$ definieras följande funktionsrum:

$$W = \mathbb{R}^d \times \{w : w|_{I_n} \in H^1(I_n, \mathbb{R}^d), n = 1, \ldots, N\} \times \mathbb{R}^d, \tag{3.3}$$

$$\dot{W} = \{w \in W : I_0 w_0^- = 0, I_T w_N^+ = 0\}, \tag{3.4}$$

$$\tilde{W} = \{w \in W : I_0 w_0^- = x_0, I_T w_N^+ = x_T\}, \tag{3.5}$$

$$U = H^1((0,T), \mathbb{R}^m), \tag{3.6}$$

$$V = H^1((0,T), \mathbb{R}^d), \tag{3.7}$$

där H betecknar ett Sobolevrum[6], $w_n = w(t_n)$, $w_n^- = \lim_{t \to t_n^-} w(t)$, $w_n^+ = \lim_{t \to t_n^+} w(t)$. De två $\mathbb{R}^d$-faktorerna i definitionen av W innehåller två yttre randvärden w_0^- och w_N^+.

Låt $x \in W$ och $u \in U$. Angrip $\dot{x}(t) = f(x(t), u(t))$ genom att multiplicera ekvationen med en testfunktion $\varphi \in V$ och integrera över $(0,T)$. Detta leder till variationsformuleringen

$$\sum_{n=1}^{N} \int_{I_n} (\dot{x} - f(x,u), \varphi)\, dt + \sum_{n=0}^{N} ([x]_n, \varphi_n) = 0, \quad \forall \varphi \in V, \tag{3.8}$$

där $[x]_n = x_n^+ - x_n^-$ och $(\cdot, \cdot)$ betecknar skalärprodukt[7]. Inför notationen $\mathcal{F}(x, u, \varphi)$ för vänsterledet i (3.8). Innebörden av variationsformuleringen blir att alla $x \in \tilde{W}$ som uppfyller $\mathcal{F}(x, u, \varphi) = 0$ också uppfyller (3.2).

I nästa steg sätts en Lagrangefunktional samman enligt

$$\mathcal{L}(x, u, z) = \mathcal{J}(x, u) + \mathcal{F}(x, u, z), \quad (x, u, z) \in W \times U \times V.$$

Sats 3.1 nedan anger de nödvändiga villkor som en given trippel (x, u, z) måste uppfylla för att $\mathcal{L}(x, u, z)$ ska vara ett optimum. Först införs ett antal beteckningar som används i satsen.

Låt $\mathcal{L}'_x(x, u, z)\varphi = \mathcal{L}'_x(x, u, z, \varphi)$ beteckna Fréchetderivatan[8] av $\mathcal{L}(x, u, z)$ med avseende på x och verkande på testfunktionen φ. För en reell- eller vektorvärd funktion $g(x_1, x_2)$ införs beteckningen $g'_i(x_1, x_2)$ för den partiella derivatan med avseende på x_i där $i = 1, 2$. Om g är reellvärd, $g : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$, gäller att

$$g'_1(x_1, x_2) : \mathbb{R}^d \to \mathbb{R}, \quad g'_2(x_1, x_2) : \mathbb{R}^m \to \mathbb{R}$$

är linjära operatorer som verkar på vektorer. Detta betecknas här

$$g'_1(x_1, x_2)y = (y, g'_1(x_1, x_2)), \quad y \in \mathbb{R}^d, \quad g'_2(x_1, x_2)y = (y, g'_2(x_1, x_2)), \quad y \in \mathbb{R}^m. \tag{3.9}$$

Om g är vektorvärd, $g : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$, satisfierar operatorerna

$$g'_1(x_1, x_2) : \mathbb{R}^d \to \mathbb{R}^d, \quad g'_2(x_1, x_2) : \mathbb{R}^m \to \mathbb{R}^d.$$

I detta fall är de partiella derivatorna matriser som verkar på vektorer.

Sats 3.1. *Betrakta*

$$\mathcal{L}(x, u, z) = \mathcal{J}(x, u) + \mathcal{F}(x, u, z), \quad (x, u, z) \in W \times U \times V,$$

[6] För definition av Sobolevrum se Appendix A.6.
[7] För definition av skalärprodukt se Appendix A.7.
[8] För definition av Fréchetderivata se Appendix A.8.

där $\mathcal{F}$ beror linjärt av z. Antag att $\mathcal{L}(x,u,z)$ har ett optimum i $(x,u,z) \in \tilde{W} \times U \times V$, då gäller att

$$\mathcal{J}'_x(x,u)\varphi_x + \mathcal{F}'_x(x,u,z)\varphi_x = 0, \quad \forall \varphi_x \in \dot{W}, \tag{3.10}$$
$$\mathcal{J}'_u(x,u)\varphi_u + \mathcal{F}'_u(x,u)\varphi_u = 0, \quad \forall \varphi_u \in U, \tag{3.11}$$
$$\mathcal{F}(x,u)\varphi_z = 0, \quad \forall \varphi_z \in V. \tag{3.12}$$

Om

$$\mathcal{J}(x,u) = l(x_0^-, x_N^+) + \int_0^T L(x,u)\,dt, \quad (x,u) \in W \times U, \tag{3.13}$$
$$\mathcal{F}(x,u,z) = \mathcal{F}(x,u)z = \sum_{n=1}^{N} \int_{I_n} (\dot{x} - f(x,u), z)\,dt + \sum_{n=0}^{N} ([x]_n, z_n),$$
$$(x,u,z) \in W \times U \times V \tag{3.14}$$

blir (3.10), (3.11) *och* (3.12) *i tur och ordning*

$$\begin{aligned} &\sum_{n=1}^{N} \int_{I_n} (\varphi_x, L'_1(x,u) - \dot{z} - f'_1(x,u)^T z)\,dt \\ &\quad + (\varphi^-_{x,0}, l'_1(x_0^-, x_N^+) - z_0) \\ &\quad + (\varphi^+_{x,\,N}, l'_2(x_0^-, x_N^+) + z_N) = 0, \quad \forall \varphi_x \in \dot{W}, \end{aligned} \tag{3.15}$$
$$\int_0^T (\varphi_u, L'_2(x,u) - f'_2(x,u)^T z)\,dt = 0, \quad \forall \varphi_u \in U, \tag{3.16}$$
$$\sum_{n=1}^{N} \int_{I_n} (\dot{x} - f(x,u), \varphi_z)\,dt + \sum_{n=0}^{N} ([x]_n, \varphi_{z,\,n}) = 0, \quad \forall \varphi_z \in V. \tag{3.17}$$

Bevis. För att $\mathcal{L}(x,u,z)$ ska anta optimum i en punkt $(x,u,z) \in \tilde{W} \times U \times V$ måste funktionalens partiella derivator vara noll. Derivatorna tolkas i Fréchetmening. Alltså betraktas den partiella derivatan med avseende på en variabel, låt säga x, som verkande på en testfunktion φ_x och betecknas $\mathcal{L}'_x(x,u,z)\varphi_x$. Testfunktionen φ_x tillhör samma rum som variabeln x.

Ekvationerna (3.10) och (3.11) följer då av att

$$\mathcal{L}'_x(x,u,z)\varphi_x = \mathcal{J}'_x(x,u)\varphi_x + \mathcal{F}'_x(x,u,z)\varphi_x = 0, \quad \forall \varphi_x \in \dot{W}, \tag{3.18}$$
$$\mathcal{L}'_u(x,u,z)\varphi_u = \mathcal{J}'_u(x,u)\varphi_u + \mathcal{F}'_u(x,u,z)\varphi_u = 0, \quad \forall \varphi_u \in U. \tag{3.19}$$

För (3.12) gäller

$$\begin{aligned} \mathcal{L}'_z(x,u,z)\varphi_z &= \mathcal{J}'_z(x,u)\varphi_z + \mathcal{F}'_z(x,u,z)\varphi_z = 0 + \mathcal{F}(x,u)\varphi_z \\ &= \mathcal{F}(x,u)\varphi_z = 0, \quad \forall \varphi_z \in V, \end{aligned} \tag{3.20}$$

där den andra likheten följer av att $\mathcal{J}$ är oberoende av z och $\mathcal{F}$ beror linjärt av z.

För att visa (3.15) noterar vi först att $l : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ beror av x i båda sina argument. Att derivera $\mathcal{J}(x,u) = l(x_0^-, x_N^+) + \int_0^T L(x,u)\,dt$ med avseende på x innebär därför att l måste deriveras med avseende på båda sina variabler. Matriserna I_0 och I_T som förekommer i definitionen av funktionsrummen i (3.4) och (3.5) gör att inte alla funktionsvärden nödvändigtvis är kända på randen. Därför blir derivatan av $l(x_0^-, x_N^+)$ med avseende på x nollskild för motsvarande element i vektorn x.

Den testfunktion φ_x som derivatan beräknas med avseende på tillhör $\dot{W}$. Därför är de element i φ_x som motsvarar de kända randvärdena i x_0^- och x_N^+ noll. Eftersom variablerna i l bara är definierade som de ensidiga gränsvärdena av x i randpunkterna, kommer derivatan med avseende på x bara ha mening då testfunktionen utvärderas i dessa punkter. För den partiella derivatan av $l(x_0^-, x_N^+)$ verkande på φ_x fås, med allt detta i åtanke:

$$\begin{aligned} l'_x(x_0^-, x_N^+)\varphi_x &= l'_1(x_0^-, x_N^+)\varphi^-_{x,\,0} + l'_2(x_0^-, x_N^+)\varphi^+_{x,\,N} \\ &= (\varphi^-_{x,\,0}, l'_1(x_0^-, x_N^+)) + (\varphi^+_{x,\,N}, l'_2(x_0^-, x_N^+)). \end{aligned}$$

Partiella derivatan av $\mathcal{J}$ med avseende på x blir därmed

$$\mathcal{J}'_1(x,u)\varphi_x = \int_0^T L'_1(x,u)\varphi_x\,dt + (\varphi^-_{x,\,0}, l'_1(x_0^-, x_N^+)) + (\varphi^+_{x,\,N}, l'_2(x_0^-, x_N^+)), \tag{3.21}$$

där derivatan flyttats innanför integraltecknet under antagandet att $L(x,u)$ och $L'_1(x,u)$ är kontinuerliga och begränsade i $W \times U$.

För fortsatta studier av $\mathcal{F}$ delas funktionalen upp i två delar:

$$\mathcal{F}(x,u,z) = \sum_{n=1}^{N} \int_{I_n} (\dot{x}, z)\,dt + \sum_{n=0}^{N} ([x]_n, z_n) - \sum_{n=1}^{N} \int_{I_n} (f(x,u), z)\,dt.$$

Låt $\mathcal{G}(x,z) = \sum_{n=1}^{N} \int_{I_n} (\dot{x}, z)\,dt + \sum_{n=0}^{N} ([x]_n, z_n)$ och variera $\mathcal{G}$ kring x. Detta ger

$$\begin{aligned} \mathcal{G}(x+\varphi_x, z) &= \sum_{n=1}^{N} \int_{I_n} (\dot{x} + \dot{\varphi}_x, z)\,dt + \sum_{n=0}^{N} ([x+\varphi_x]_n, z_n) \\ &= \sum_{n=1}^{N} \int_{I_n} (\dot{x}, z)\,dt + \sum_{n=0}^{N} ([x]_n, z_n) + \sum_{n=1}^{N} \int_{I_n} (\dot{\varphi}_x\, z)\,dt + \sum_{n=0}^{N} ([\varphi_x]_n, z_n) \\ &= \mathcal{G}(x,z) + \mathcal{G}(\varphi_x, z), \end{aligned}$$

för $\varphi_x \in \dot{W}$. Alltså kan deriveringsoperatorn verkande på φ_x läsas ut som

$$\mathcal{G}'_1(x,z)\varphi_x = \sum_{n=1}^{N} \int_{I_n} (\dot{\varphi}_x, z)\,dt + \sum_{n=0}^{N} ([\varphi_x]_n, z_n).$$

Den partiella Fréchetderivatan av $\mathcal{F}$ med avseende på x blir därmed

$$\mathcal{F}'_1(x,u,z)\varphi_x = \sum_{n=1}^{N} \int_{I_n} (\dot{\varphi}_x - f'_1(x,u)\varphi_x, z)\,dt + \sum_{n=0}^{N} ([\varphi_x]_n, z_n), \tag{3.22}$$

där återigen deriveringen gjorts innanför integraltecknet för den del av $\mathcal{F}$ som inte innefattar $\mathcal{G}$. Partiell integration görs på varje delterm av den första summan i (3.22). Detta ger

$$\begin{aligned} \mathcal{F}'_1(x,u,z)\varphi_x &= \sum_{n=1}^{N} \int_{I_n} (\dot{\varphi}_x - f'_1(x,u)\varphi_x, z)\,dt + \sum_{n=0}^{N} ([\varphi_x]_n, z_n) \\ &= \sum_{n=1}^{N} (\varphi_x, z)\Big|_{t=t_{n-1}}^{t_n} - \sum_{n=1}^{N} \int_{I_n} (\varphi_x, \dot{z})\,dt - \sum_{n=1}^{N} \int_{I_n} (f'_1(x,u)\varphi_x, z)\,dt \\ &\quad + \sum_{n=0}^{N} ([\varphi_x]_n, z_n) = \sum_{n=1}^{N} (\varphi^-_{x,\,n}, z_n) - \sum_{n=1}^{N} (\varphi^+_{x,\,n-1}, z_{n-1}) - \sum_{n=1}^{N} \int_{I_n} (\varphi_x, \dot{z})\,dt \\ &\quad - \sum_{n=1}^{N} \int_{I_n} (f'_1(x,u)\varphi_x, z)\,dt + \sum_{n=0}^{N} (\varphi^+_{x,\,n}, z_n) - \sum_{n=0}^{N} (\varphi^-_{x,\,n}, z_n). \end{aligned}$$

I den sista likheten har använts att $\lim_{t\to t_n^-} \varphi_x(t) = \varphi_{x,\,n}^-$ och $\lim_{t\to t_{n-1}^+} \varphi_x(t) = \varphi_{x,\,n-1}^+$.

Nu noteras att

$$\begin{aligned}\sum_{n=1}^{N}(\varphi_{x,\,n}^-, z_n) - \sum_{n=1}^{N}(\varphi_{x,\,n-1}^+, z_{n-1}) + \sum_{n=0}^{N}(\varphi_{x,\,n}^+, z_n) - \sum_{n=0}^{N}(\varphi_{x,\,n}^-, z_n)\\ = (\varphi_{x,\,N}^+, z_N) - (\varphi_{x,\,0}^-, z_0),\end{aligned}$$

vilket inses eftersom $\sum_{n=0}^{N}(\varphi_{x,\,n}^+, z_n) = \sum_{n=1}^{N+1}(\varphi_{x,\,n-1}^+, z_{n-1})$.

Till sist används en räkneregel för skalärprodukt:

$$(Ax, b) = (x, A^T b), \quad A \in \mathbb{R}^{d\times d}, x \in \mathbb{R}^d,\, b \in \mathbb{R}^d,$$

för att skriva om $\sum_{n=1}^{N}\int_{I_n}(f_1'(x,u)\varphi_x, z)\,dt = \sum_{n=1}^{N}\int_{I_n}(\varphi_x, f_1'(x,u)^T z)\,dt$. Här gäller det att $f_1'(x,u) \in \mathbb{R}^{d\times d}$. Slutligen kan $\mathcal{F}_1'(x,u,z)\varphi_x$ skrivas

$$\mathcal{F}_1'(x,u,z)\varphi_x = \sum_{n=1}^{N}\int_{I_n}(\varphi_x,\, -\dot z - f_1'(x,u)^T z)\,dt + (\varphi_{x,\,N}^+, z_N) - (\varphi_{x,\,0}^-, z_0). \tag{3.23}$$

Nu följer (3.15) genom att sätta in $\mathcal{J}_1'(x,u)\varphi_x$ från (3.21) och $\mathcal{F}_1'(x,u,z)\varphi_x$ från (3.23) i (3.18):

$$\begin{aligned}\mathcal{L}_1'(x,u,z)\varphi_x = \mathcal{J}_1'(x,u)\varphi_x + \mathcal{F}_1'(x,u,z)\varphi_x = \sum_{n=1}^{N}\int_{I_n}(\varphi_x, L_1'(x,u) - \dot z - f_1'(x,u)^T z)\,dt\\ + (\varphi_{x,\,0}^-, l_1'(x_0^-, x_N^+) - z_0) + (\varphi_{x,\,N}^+, l_2'(x_0^-, x_N^+) + z_N) = 0, \quad \forall\varphi_x \in \dot W.\end{aligned}$$

Återstår gör att visa (3.16) och (3.17). Den förstnämnda ekvationen följer av det nödvändiga villkoret på partiella derivatan i (3.19) efter att deriveringsoperatorn fått verka innanför integrationstecknet på samma sätt som tidigare.

$$\begin{aligned}\mathcal{L}_2'(x,u,z)\varphi_u &= \mathcal{J}_2'(x,u)\varphi_u + \mathcal{F}_2'(x,\,u,z)\varphi_u\\ &= 0 + \int_0^T L_2'(x,u)\varphi_u\,dt - \sum_{n=1}^{N}\int_{I_n}(f_2'(x,u)\varphi_u, z)\,dt + 0\\ &= \sum_{n=1}^{N}\int_{I_n}(\varphi_u, L_2'(x,u) - f_2'(x,u)^T z)\,dt = 0, \quad \forall\varphi_u \in U.\end{aligned}$$

Slutligen gäller, enligt definitionen av $\mathcal{F}$ och (3.20), att

$$\mathcal{L}_3'(x,u,z)\varphi_u = \mathcal{F}(x,\,u)\varphi_z = \sum_{n=1}^{N}\int_{I_n}(\dot x - f(x,u), \varphi_z)\,dt + \sum_{n=0}^{N}([x]_n, \varphi_{z,\,n}) = 0, \quad \forall\varphi_z \in V.$$

□

Alltså fås för $(x,u,z) \in \tilde{W} \times U \times V$ följande tre ekvationer

$$\begin{aligned}
&\sum_{n=1}^{N} \int_{I_n} (\varphi_x, L_1'(x,u) - \dot{z} - f_1'(x,u)^T z)\, dt \\
&\quad + (\varphi_{x,0}^-, l_1'(x_0^-, x_N^+) - z_0) \\
&\quad + (\varphi_{x,N}^+, l_2'(x_0^-, x_N^+) + z_N) = 0, \quad \forall \varphi_x \in \dot{W}, \qquad (3.24) \\
&\quad \int_0^T (\varphi_u, L_2'(x,u) - f_2'(x,u)^T z)\, dt = 0, \quad \forall \varphi_u \in U, \qquad (3.25) \\
&\sum_{n=1}^{N} \int_{I_n} (\dot{x} - f(x,u), \varphi_z)\, dt + \sum_{n=0}^{N} ([x]_n, \varphi_{z,n}) = 0, \quad \forall \varphi_z \in V. \qquad (3.26)
\end{aligned}$$

Eftersom en lösning $(x,u,z) \in \tilde{W} \times U \times V$ till (3.24), (3.25) och (3.26) medför att $\mathcal{F}(x,u,z) = 0$ och att $\mathcal{L}(x,u,z)$ är optimal innebär det att lösningstrippeln också löser det optimala styrproblemet.

3.2.2 Diskretiserad formulering

För att lösa det kopplade ekvationssystemet bestående av (3.24), (3.25) och (3.26) diskretiseras problemet och angrips med en finita elementmetod. Följande funktionsrum introduceras:

$$\begin{aligned}
W_h &= \mathbb{R}^d \times \{w : w|_{I_n} \in \mathcal{P}^q(I_n, \mathbb{R}^d), n = 1, \ldots, N\} \times \mathbb{R}^d, \\
\dot{W}_h &= \{w \in W_h : I_0 w_0^- = 0, I_T w_N^+ = 0\}, \\
U_h &= \{u \in C([0,T], R^m) : u|_{I_n} \in \mathcal{P}^{q+1}(I_n, \mathbb{R}^m)\}, \\
V_h &= \{v \in C([0,T], R^d) : v|_{I_n} \in \mathcal{P}^{q+1}(I_n, \mathbb{R}^d)\}.
\end{aligned}$$

Finita elementproblemet blir nu att hitta $(x_h, u_h, z_h) \in W_h \times U_h \times V_h$ sådana att

$$\begin{aligned}
&\sum_{n=1}^{N} \int_{I_n} (\varphi_x, L_1'(x_h,u_h) - \dot{z}_h - f_1'(x_h,u_h)^T z_h)\, dt \\
&\quad + (\varphi_{x,0}^-, l_1'(x_{h,0}^-, x_{h,N}^+) - z_{h,0}) \\
&\quad + (\varphi_{x,N}^+, l_2'(x_{h,0}^-, x_{h,N}^+) + z_{h,N}) = 0, \quad \forall \varphi_x \in \dot{W}_h, \qquad (3.27) \\
&\quad \int_0^T (\varphi_u, L_2'(x_h,u_h) - f_2'(x_h,u_h)^T z_h)\, dt = 0, \quad \forall \varphi_u \in U_h, \qquad (3.28) \\
&\sum_{n=1}^{N} \int_{I_n} (\dot{x}_h - f(x_h,u_h), \varphi_z)\, dt + \sum_{n=0}^{N} ([x_h]_n, \varphi_{z,n}) = 0, \quad \forall \varphi_z \in V_h, \qquad (3.29) \\
&\quad I_0 x_{h,0}^- = x_0, \quad I_T x_{h,T}^+ = x_T. \qquad (3.30)
\end{aligned}$$

Notera hur de två sista termerna i (3.27) hänger samman med randvillkoret (3.30). Att I_0 och I_T är binärt diagonala medför att randpunkterna inte nödvändigtvis är kända för alla x_i i $x = (x_1, \ldots, x_d)$. Att ett diagonalelement I_{kk} är noll innebär att motsvarande start- eller slutvärde för x_k är okänt. Definitionen

$$\dot{W} = \{w \in W : I_0 w_0^- = 0, I_T w_N^+ = 0\}$$

har då till följd att för de x_i som har kända start- eller slutvärden blir motsvarande $\varphi_{x,0}^- = 0$ eller $\varphi_{x,T}^+ = 0$ i (3.27).

3.3 Kvadratisk-linjära optimala styrproblem

I detta avsnitt betraktas en förenkling av det allmänna styrproblemet till att omfatta problem med kvadratisk $\mathcal{J}(x,u)$ och linjär $\mathcal{F}(x,u,z)$. Den generella formen blir: hitta $x(t)$ och $u(t)$

som

$$\text{minimerar} \quad \mathcal{J}(x(t),u(t)) = \|x(0)\|_{S_0}^2 + \|x(T)\|_{S_T}^2 + \int_0^T (\|x\|_Q^2 + \|u\|_R^2)\,dt, \tag{3.31}$$

$$\text{under bivillkoret} \quad \begin{cases} \dot{x}(t) = A(t)x + B(t)u, \quad 0 < t < T, \\ I_0 x(0) = x_0, \quad I_T x(T) = x_T, \end{cases} \tag{3.32}$$

där $A(t), I_0, I_T, Q(t), S_0, S_T \in \mathbb{R}^{d\times d}$, $B(t) \in \mathbb{R}^{d\times m}$, $R(t) \in \mathbb{R}^{m\times m}$ och $\|v\|_S^2 = (v, Sv)$. Ytterligare krav är att $Q(t), S_0, S_T$ är positivt semidefinita[9] symmetriska matriser och att $R(t)$ är en positivt definit[10] symmetrisk matris.

Med de allmänna beteckningarna i Definition 3.1 fås

$$\begin{aligned} l(x(0),x(T)) &= \|x(0)\|_{S_0}^2 + \|x(T)\|_{S_T}^2, \\ L(x,u) &= \|x\|_Q^2 + \|u\|_R^2, \\ f(x,u) &= Ax + Bu. \end{aligned}$$

Om ovanstående uttryck deriveras med avseende på respektive variabel fås

$$\begin{aligned} l_1'(x(0),x(T)) &= 2S_0x(0), \\ l_2'(x(0),x(T)) &= 2S_Tx(T), \\ L_1'(x,u) &= 2Qx, \\ L_2'(x,u) &= 2Ru, \\ f_1'(x,u) &= A, \\ f_2'(x,u) &= B. \end{aligned}$$

Dessa insatt i (3.27), (3.28) och (3.29) ger följande finita elementproblem: hitta $(x_h, u_h, z_h) \in W_h \times U_h \times V_h$ som uppfyller

$$\begin{aligned} &\sum_{n=1}^N \int_{I_n} (\varphi_x, 2Qx_h - \dot{z}_h - A^T z_h)\,dt \\ &\quad + (\varphi_{x,0}^-, 2S_0x_{h,0}^- - z_{h,0}) \\ &\quad + (\varphi_{x,N}^+, 2S_Tx_{h,N}^+ + z_{h,N}) = 0, \quad \forall \varphi_x \in \dot{W}_h, \end{aligned} \tag{3.33}$$

$$\int_0^T (\varphi_u, 2Ru_h - B^T z_h)\,dt = 0, \quad \forall \varphi_u \in U_h, \tag{3.34}$$

$$\sum_{n=1}^N \int_{I_n} (\dot{x}_h - Ax_h - Bu_h, \varphi_z)\,dt + \sum_{n=0}^N ([x_h]_n, \varphi_{z,n}) = 0, \quad \forall \varphi_z \in V_h, \tag{3.35}$$

$$I_0x_{h,0}^- = x_0, \quad I_Tx_{h,T}^+ = x_T. \tag{3.36}$$

[9] För definition av positivt semidefinit matris se Appendix A.10.
[10] För definition av positivt definit matris se Appendix A.9.

4 FEniCS-manual

Programvaran vi använder för att implemera finita elementmetoden från Krafts avhandling [2] heter FEniCS. FEniCS är sammansatt av flera mjukvarukomponenter, däribland DOLFIN som är ett bibliotek för hantering av finita elementmetoder och Viper som sköter visualisering. Det går att implementera FEniCS via både C++ och Python. Det här arbetet görs uteslutande i Python.

I kapitel 5 presenteras några olika problem och deras lösningar i FEniCS. För att underlätta förståelsen för hur FEniCS används följer här en alfabetiskt ordnad lista över de viktigaste funktioner och objekt som förekommer i kapitel 5. Information till detta kapitel hämtas från två huvudsakliga källor: *Automated Solution of Differential Equations by the Finite Element Method* av A. Logg med flera [15] och dokumentationen på FEniCS Projects hemsida [16].

`assemble`(a) - a består av en eller flera termer ur finita elementproblemet som ska lösas. Kommandot ger möjlighet att plocka ut delar ur ekvationssystemet separat. Låt variationsformuleringen av problemet vara

$$\int_\Omega \psi'(t)\varphi'(t)\,dt = \int_\Omega f(t)\varphi(t)\,dt,$$

där ψ och φ är ansats- respektive testfunktion och f är känd. Beteckna vänsterledet a och högerledet L samt låt test- och ansatsfunktioner tillhöra ett rum uppspänt av linjära Lagrangebaser. Anropet `assemble`(a) ger då en matris A bestående av element

$$A_{ji} = \int_\Omega \psi_i'(t)\varphi_j'(t)\,dt, \quad i,j = 0,1,\ldots,n, \tag{4.1}$$

och `assemble`(L) ger en vektor b där

$$b_j = \int_\Omega f(t)\varphi_j(t)\,dt, \quad j = 0,1,\ldots,n, \tag{4.2}$$

där n är antal delintervall i partitioneringen. I matrisen A samt i vektorn b svarar första och sista raderna mot de basfunktioner som antar värdet 1 på randen. Dessa två rader får mening först när randvillkoren lagts på ekvationen.

`apply`(A,b) - Lägger till randvillkoren i A och b, vilka är av typ matris eller vektor erhållna via `assemble`. Kommandot `apply` kan också anropas med bara ett argument, vilket används då endast en matris eller en vektor ska anpassas till randvärdena. Med A som matrisen från (4.1) och b som vektorn från (4.2) kommer vi med randvillkoren sparade i variabeln *bc* efter kommandot

```
bc.apply(A,b)
```

fram till ekvationssystemet

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & s_{11} & \cdots & s_{1,n-1} & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & s_{n-1,1} & \cdots & s_{n-1,n-1} & \vdots \\ 0 & \cdots & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} \xi_0 \\ \hline \xi_1 \\ \vdots \\ \xi_{n-1} \\ \hline \xi_n \end{bmatrix} = \begin{bmatrix} u_0 \\ \hline b_1 \\ \vdots \\ b_{n-1} \\ \hline u_T \end{bmatrix}, \tag{4.3}$$

där u_0 och u_T är initial- respektive slutvärde. Systemet i (4.3) är FEniCS representation av finita elementformuleringen till problemet:

$$\begin{cases} -\ddot{u}(t) = f(t), & t \in (0,T), \\ u(0) = u_0, & u(T) = u_T, \end{cases} \tag{4.4}$$

till vilken alltså

$$\xi = \begin{bmatrix} \xi_0 \\ \xi_1 \\ \vdots \\ \xi_{n-1} \\ \xi_n \end{bmatrix} \tag{4.5}$$

är den diskreta lösningen.

assemble_system(a, L, bc) - Ekvivalent till att anropa `assemble` med a och L som argument och sedan `bc.apply`(a, L).

avg(v) - Nyttjas då en diskontinuerlig test- eller ansatsfunktion v används och returnerar genomsnittet av v i en punkt där funktionen är diskontinuerlig. Med genomsnittet menas här $\frac{v^- + v^+}{2}$, där v^- och v^+ är gränsvärdena i punkten från vänster respektive höger.

Constant$(value)$ - Skapar en skalär eller en vektor av skalärer beroende på hur *value* är definierad. `A = Constant(1)` och `A = Constant((1,2))` är exempel på hur klassen kan användas.

DirichletBC$(V, g, subdomain)$ - Klass för att ange Dirichletrandvärden till en partiell differentialekvation. V är rummet dit ansatsfunktionerna hör, g anger vilket värde funktionen ska anta på randen och *subdomain* talar om var i området g ska gälla. I de fall olika Dirichletrandvärden i start- respektive slutpunkt önskas anges randvärdena i form av en vektor. Det kan exempelvis se ut så här:

```
def left_boundary(x, on_boundary):
    tol = 1E-14   # tolerance for coordinate comparisons.
    return on_boundary and abs(x[0]) < tol
    #Returns true if on boundary and if t=0,
    #i.e. the lower part of the boundary.

def right_boundary(x, on_boundary):
    tol = 1E-14   # tolerance for coordinate comparisons.
    return on_boundary and abs(x[0] - T) < tol
    #T is the final value, returns true if on boundary and if t=T,
    #i.e. the upper part of the boundary.

lowerBC = DirichletBC(V, "u_0", left_boundary)  #u_0=u(0).
upperBC = DirichletBC(V, "u_T", right_boundary)  #u_T=u(T).
bc = [lowerBC,upperBC]
```

DOLFIN_EPS - Konstant värde som används för att kompensera för datorns precision vid avrundning. Istället för att exempelvis kontrollera om ett tal är lika med noll så kontrolleras om talet är mindre än `DOLFIN_EPS` för att undvika avrundningsfel.

Dx(u, i) - Returnerar derivatan av funktionen u med avseende på variabeln x_i.

Expression$(formula, p1 = v1, p2 = v2 \ldots)$ - Skapar en skalär- eller vektorvärd funktion som kan innehålla parametrar $p1, p2, \ldots$. efter önskemål, och i sådana fall tilldelas dessa värden $v1, v2, \ldots$ Det går även att tilldela parametrarna i efterhand som visas i exempelkoden nedan. Variablerna i $formula$ tilldelas enligt $x[0], x[1]$ och så vidare beroende på hur många som behövs. Nedan visas ett exempel på hur `Expression` kan användas.

```
f = Expression(("A*x[0] + 2 - 3*x[1]", "2"), A = 2)
#Vector valued expression, A sets equal to 2.
f.A = 3 #Changes the value of A to 3.
print f(2,1) #Prints out the value of f evaluated at the point (2, 1).
```

Function(V, ξ) - Skapar en funktion u_h i ett funktionsrum V enligt

$$u_h = \sum_{i=1}^{n} \xi_i \varphi_i \,, \tag{4.6}$$

där $\{\varphi_i\}_{i=1}^n$ är en bas för V och ξ är den diskreta lösningsvektorn till problemets finita elementformulering. Ofta är ξ inte känd vid anropet av `Function`. Då utelämnas det andra argumentet tills dess att ξ är framräknad (se `solve`). Om V är en funktion istället för ett funktionsrum skapas en ny funktion i samma funktionsrum som V.

FunctionSpace$(mesh, family, degree)$ - Skapar ett funktionsrum över en partitionering bestående av funktioner från strängen $family$ av gradtal $degree$. De vanligaste funktionsfamiljerna är kontinuerlig Galerkin ($family$ = `"CG"` eller $family$ = `"Lagrange"`) och diskontinuerlig Galerkin ($family$ = `"DG"`).

Om n är antalet delintervall ger kommandot

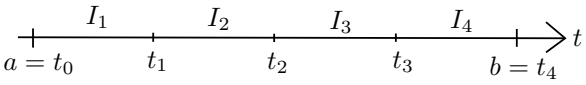
```
FunctionSpace(mesh, "Lagrange", 1)
```

ett funktionsrum av dimension $n+1$ eftersom FEniCS alltid låter de basfunktioner som antar värdet 1 på randen ingå i rummet. Det resulterande ekvationssystemet kommer således vara av dimension $(n+1)\times(n+1)$, men första och sista raderna är reserverade åt randvillkoren. Så länge randvillkoren är kända kommer alltså det totala antalet okända i ekvationen fortfarande vara $n-1$.

grad(u) - Returnerar gradienten av funktionen u.

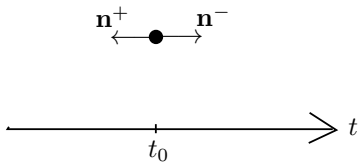
inner(u, v) - Returnerar skalärprodukten av u och v, med andra ord (u, v).

Interval(n, a, b) - Skapar en partitionering av intervallet $[a, b]$ i n delintervall. I Figur 4.1 visas ett intervall skapat med kommandot `Interval`$(4, a, b)$.



Figur 4.1: Intervall med $n = 4$.

jump(u) - Definieras som $[u] = u^+ n^+ + u^- n^-$. Metoden behövs då en diskontinuerlig test- eller ansatsfunktion används. u^+ och u^- är gränsvärdena då en variabel närmar sig en diskontinuitetspunkt från respektive håll. n^+ och n^- är ytnormaler tillhörande området där u^+ respektive u^- har stöd. Då problemet är endimensionellt blir det punktnormaler snarare än ytnormaler. I Figur 4.2 antas att funktionen har en diskontinuitet vid tiden t_0. Notera att n^+ har negativ riktning medan n^- har positiv.



Figur 4.2: Normalerna vid en diskontinuitetspunkt.

`MeshFunction`$(val, mesh, dim)$ - Ett objekt av typen `MeshFunction` är en diskret funktion definierad på en delmängd av en partitionering. Strängen *val* anger vilken typ av funktion som tillåts av följande alternativ:

`"int"` - heltal,
`"uint"` - ickenegativt heltal,
`"double"` - flyttal,
`"bool"` - boolean.

Objektet *mesh* är partitioneringen till vilken denna funktion ska höra och *dim* är dimensionen av fasetterna[11].

Ett exempel på när klassen `MeshFunction` kan användas är då olika randvillkor behöver appliceras på olika delar av randen. Detta görs genom att först skapa subklasser till klassen `SubDomain`, vilket kan se ut så här

```
class InitialBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and abs(x[0]) < DOLFIN_EPS

class FinalBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and abs(x[0] - T) < DOLFIN_EPS
```

Nästa steg är att definiera funktioner som utgör de olika delarna av randen och markera dem så att de kan särskiljas. Detta görs genom

```
boundary_parts = MeshFunction("uint", mesh, mesh.topology().dim() - 1)
#Dimension 1 lower than the mesh.

start = InitialBoundary()
start.mark(boundary_parts,0) #Marks the boundary part as 0.
final = FinalBoundary()
final.mark(boundary_parts,1) #Marks the boundary part as 1.
```

Nu är de olika delarna markerade och de nås genom `ds(0)` respektive `ds(1)`.

`MixedFunctionSpace`$(spaces)$ - där *spaces* är en lista med funktionsrum. Låt säga att funktionsrummen W, Uoch V redan är definierade. Då gör kommandot

```
M = MixedFunctionSpace([W, U, V])
```

att $M = W \times U \times V$. För att komma åt underrum till M används kommandot `sub`. För tillgång till W skrivs `M.sub(0)` och så vidare.

`plot`(U) - ger en grafisk representation av U. Objektet U ska vara av klassen `Mesh`, `MeshFunction` eller `Function`.

[11] För definition av fasett se Appendix A.12.

solve$(a == L, u, bc)$ - löser ekvationen

$$a = L,$$

som är den abstrakta formuleringen av ett finita elementproblem. Metoden `solve` anropar `assemble` med a och L som argument samt lägger på randvillkoren bc. Argumentet u är ett objekt av klassen `Function` och anger på vilken form lösningen ska returneras.

TestFunction(V) - Skapar en testfunktion i funktionsrummet V.

TestFunctions(M) - Skapar flera testfunktioner i funktionsrummet M, vilket är ett objekt av klassen `MixedFunctionsSpace`. Antalet testfunktioner är detsamma som antalet komponenter i M.

TrialFunction(V) - Skapar en ansatsfunktion i funktionsrummet V. En sådan funktion söks som lösning till finita elementproblemet.

TrialFunctions(M) - Skapar flera ansatsfunktioner i funktionsrummet M, vilket är ett objekt av klassen `MixedFunctionsSpace`. Antalet ansatsfunktioner är detsamma som antalet komponenter i M.

UnitInterval(n) - Skapar en partitionering av intervallet [0, 1] i n delintervall.

VectorFunctionSpace$(mesh, family, degree, dim)$ - Klassen fungerar på samma sätt som `FunctionSpace` fast för vektorvärda funktioner med dim komponenter. Detta kan användas exempelvis vid omskrivning av en andra ordningens differentialekvation till ett system av första ordningens differentialekvationer. Ekvationen $u'' + u = 0$ kan skrivas som systemet

$$\begin{cases} u' = v, \\ v' = -u, \end{cases}$$

vilket på matrisform blir

$$\begin{bmatrix} u' \\ v' \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4.7}$$

I FEniCS utförs detta genom att först definiera test- och ansatsfunktioner i det vektorvärda funktionsrummet av dimension 2, via

```
V = VectorFunctionSpace(mesh, "Lagrange", 1, 2)
u = TrialFunction(V)
v = TestFunction(V)
```

Ekvation (4.7) skrivs i FEniCS enligt

```
A = Constant((("0.0","1.0"), ("-1.0","0.0")))
a = inner(nabla_grad(u) - A*u, v)*dx
```

där det nu bara saknas definition av en nollvektor till högerled samt problemspecifika randvärden innan `solve` anropas för att erhålla lösningen.

5 Tre problem i teorin och i FEniCS

I detta kapitel studeras ett urval matematiska problem kopplade till optimala styrproblem och hur de kan lösas i FEniCS. Målet med studien är att lösa optimala styrproblem med den metod Kraft använder i sin doktorsavhandling *Adaptive Finite Element Methods for Optimal Control Problems* [2]. På vägen dit betraktas några problem med anknytning till detta slutgiltiga mål.

I avsnitt 5.1 presenteras ett problem som behandlar minimering av en funktional. Variationskalkyl används där för att härleda det nödvändiga villkoret för minimum. Problemet är inte av samma typ som det ett optimalt styrproblem resulterar i. Det är dock relevant eftersom det är grundläggande i finita elementsammanhang och knyter an till slutmålet, som också är ett minimeringsproblem.

I nästföljande avsnitt, 5.2, betraktas ett begynnelsevärdesproblem i en dimension. Problemet lyder:

$$\begin{cases} \dot{x}(t) = a(t)x(t), \quad 0 < t < T, \\ x(0) = x_0, \end{cases} \tag{5.1}$$

och är en förenkling av det bivillkor som ett optimalt styrproblem minimeras under. Avsnitt 5.3 handlar slutligen om lösning av ett kvadratisk-linjärt optimalt styrproblem, analytiskt och i FEniCS.

5.1 Ett minimeringsproblem

Att minimera en funktional leder i många fall naturligt till en svag formulering av en differentialekvation. Denna formulering kan ofta fås fram genom det nödvändiga villkoret att förstaderivatan av funktionalen, i den mening den är definierad, måste vara noll i extrempunkter. Vid numerisk lösning av problemet kan detta villkor utnyttjas och en finita elementmetod användas för att approximera den optimala lösningen [1]. Följande problem är ett exempel på hur variationsformuleringen av differentialekvationen fås från minimeringsproblemet och hur problemet kan lösas med en finita elementmetod.

5.1.1 Problemet i teorin

Låt, för enkelhetens skull, $\Omega = [0, 1]$ och definiera vektorrummet

$$H_0^1(\Omega) = \left\{ v : \int_\Omega (v^2 + (v')^2)\, dx \, < \infty, \, v = 0 \text{ på } \partial\Omega \right\},$$

med skalärprodukt

$$(u, v)_{H_0^1(\Omega)} = \int_\Omega (uv + u'v')\ dx$$

och norm

$$||v||_{H_0^1(\Omega)} = \left(\int_\Omega (v^2 + (v'))^2\, dx \right)^{1/2}.$$

Betrakta sedan funktionalen

$$\begin{aligned} \mathcal{F} :\ & H_0^1(\Omega) \to \mathbb{R} \\ & v \mapsto \mathcal{F}(v) = \frac{1}{2} \left|\left|v'\right|\right|^2_{L^2(\Omega)} - (f, v)_{L^2(\Omega)}, \end{aligned}$$

där $v = v(x)$ och $f = f(x)$ är en känd, begränsad funktion med Ω som definitionsmängd.

Problemet är nu att minimera $\mathcal{F}$ över alla funktioner $v \in H_0^1(\Omega)$. Alltså söks:

$$\min_{v \in H_0^1(\Omega)} \quad \mathcal{F}(v). \tag{5.2}$$

Under antagandet att funktionalen är deriverbar är ett nödvändigt villkor för minimum att funktionalens förstaderivata är noll. Då $H_0^1(\Omega)$ och $\mathbb{R}$ är Banachrum[12] ska derivatan tolkas i Fréchetmening. Genom att linearisera $\mathcal{F}$ kring en godtycklig punkt $v \in H_0^1(\Omega)$ kan funktionalens Fréchetderivata[13] läsas ut [12]. För testfunktionen $w \in H_0^1(\Omega)$ fås:

$$\begin{aligned}\mathcal{F}(v+w) &= \frac{1}{2}\left|\left|v'+w'\right|\right|^2_{L^2(\Omega)} - (f, v+w)_{L^2(\Omega)} \\ &= \frac{1}{2}\left|\left|v'\right|\right|^2_{L^2(\Omega)} + (v', w')_{L^2(\Omega)} + \frac{1}{2}\left|\left|w'\right|\right|^2_{L^2(\Omega)} - (f, v)_{L^2(\Omega)} - (f, w)_{L^2(\Omega)} \\ &= \mathcal{F}(v) + (v', w')_{L^2(\Omega)} - (f, w)_{L^2(\Omega)} + \frac{1}{2}\left|\left|w'\right|\right|^2_{L^2(\Omega)}. \end{aligned} \tag{5.3}$$

Enligt definitionen av Fréchetderivata vill vi nu visa att termen $\frac{1}{2}\left|\left|w'\right|\right|^2_{L^2(\Omega)}$ i (5.3) satisfierar

$$\frac{1}{2}\left|\left|w'\right|\right|^2_{L^2(\Omega)} = o(\left|\left|w\right|\right|_{H_0^1(\Omega)}) \text{ då } w \to 0.^{14} \tag{5.4}$$

För att underlätta detta införs

$$\left|\left|v\right|\right|_a = \left|\left|v'\right|\right|_{L^2(\Omega)} = \left(\int_\Omega (v')^2\, dx\right)^{1/2}$$

som ekvivalent norm till $\left|\left|v\right|\right|_{H_0^1(\Omega)}$ i $H_0^1(\Omega)$. Att normerna är ekvivalenta innebär att det finns positiva konstanter c_0 och c_1 sådana att $c_0\left|\left|v\right|\right|^2_{H_0^1(\Omega)} \leq \left|\left|v\right|\right|^2_a \leq c_1\left|\left|v\right|\right|^2_{H_0^1(\Omega)}$. Därmed kan $\left|\left|v\right|\right|_a$ användas istället för $\left|\left|v\right|\right|_{H_0^1(\Omega)}$ som norm i $H_0^1(\Omega)$ utan essentiellt förändrat resultat [1].

Eftersom $\frac{\left|\left|w'\right|\right|^2_{L^2(\Omega)}}{\left|\left|w'\right|\right|_{L^2(\Omega)}} \to 0$ då $w' \to 0$ kan vi efter införandet av den nya normen se att villkoret i (5.4) är uppfyllt.

Derivatan, som är en linjär operator verkande på w,

$$\mathcal{F}'(v)w = \mathcal{F}'(v;\, w) = (v', w')_{L^2(\Omega)} - (f, w)_{L^2(\Omega)}, \tag{5.5}$$

kan därmed identifieras.

Det nödvändiga villkoret för att funktionalen ska anta minimum i v lyder nu:

$$\mathcal{F}'(v;\, w) = (v', w')_{L^2(\Omega)} - (f, w)_{L^2(\Omega)} = 0, \quad \forall w \in H_0^1(\Omega), \tag{5.6}$$

vilket känns igen som variationsformuleringen av Poissons ekvation i en dimension (se till exempel *Computational Differential Equations* av K. Eriksson med flera [1]). Här är f källterm och randvillkoren homogena. Poissons ekvation lyder då:

$$\begin{cases} -u''(x) = f(x), & x \in [0,1], \\ u(0) = u(1) = 0. \end{cases} \tag{5.7}$$

Det är möjligt att visa att implikationen även gäller åt andra hållet, så att minimeringsproblemet (5.2) är ekvivalent till variationsformuleringen (5.6). Även för bevis av detta hänvisar vi till K. Eriksson med fleras bok *Computational Differential Equations* [1].

Målet är nu att approximera lösningen genom en finita elementmetod. I detta fall väljs en Galerkinmetod, cG(1), där alltså rummet av ansats- och testfunktioner består av kontinuerligt

[12] För definition av Banachrum se Appendix A.14.
[13] För definition av Fréchetderivata se Appendix A.8.
[14] För definition av ordobegreppet se Appendix A.15.

styckvisa polynom av grad ett [1]. Alltså görs först en partitionering av intervallet $[0,1]$ i N delintervall $I_i = (x_i - x_{i-1})$ enligt $\mathcal{T} : 0 = x_0 < x_1 < \ldots < x_{N-1} < x_N = 1$. Längden av varje delintervall antas här vara konstant, $h_i = h = x_i - x_{i-1}$, $i = 1, \ldots, N$.

Sedan införs vektorrummet

$$V_h^0 = \left\{ v \in C[0,1] : v|_{I_i} \in \mathcal{P}^1(I_i),\, v(0) = v(1) = 0 \right\}, \tag{5.8}$$

vilket är rummet där en finita elementlösning till problemet söks. Som bas för V_h^0 väljs $\{\varphi_i(x)\}_{i=1}^{N-1}$, där

$$\varphi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{h}, & x \in (x_{i-1}, x_i), \\ \dfrac{x_{i+1} - x}{h}, & x \in (x_i, x_{i+1}). \end{cases}$$

Från variationsformuleringen fås finita elementproblemet: finn $v_h = \sum_{i=1}^{N-1} \xi_i \varphi_i(x) \in V_h^0$ sådan att

$$\int_0^1 v_h'(x) w'(x)\, dx = \int_0^1 f(x) w(x)\, dx, \quad \forall w \in V_h^0. \tag{5.9}$$

Då $\{\varphi_i(x)\}_{i=1}^{N-1}$ är en bas för V_h^0 räcker det att testa mot dessa i w:s ställe. Detta ger ett linjärt ekvationssystem av storlek $(N-1) \times (N-1)$ att lösa för de okända konstanterna $\{\xi_i\}_{i=1}^{N-1}$.

5.1.2 Problemet i FEniCS

Målet är nu att numeriskt lösa Poissons ekvation (5.7) och därmed även det ursprungliga minimeringsproblemet (5.2) i FEniCS med en metod som motsvarar Galerkinmetoden beskriven i avsnittet ovan. I programmet som följer generaliseras problemet till:

$$\begin{cases} -u''(x) = f(x), & x \in [a,b], \\ u(a) = u_a, \quad u(b) = u_b. \end{cases} \tag{5.10}$$

De okända parametrarna får värden som överensstämmer med (5.7) ovan, men kan ändras om så önskas. Den fullständiga koden återfinns i avsnitt 5.1.3 och här styckas koden upp rad för rad för att följa vad som sker i programmet.

Först sker en import av alla Dolfins klasser och metoder genom kommandot:

```
from dolfin import *
```

Därefter definieras problemet, (5.10), genom att specificera intervallet $[a,b]$, randvärdena u_a, u_b och källtermen $f(x)$:

```
a = 0
b = 1
u_a = Constant(0)
u_b = Constant(0)
f = Expression("1")
```

Finita elementformuleringen av problemet börjar i och med raderna

```
N = 10
mesh = Interval(N, a, b)
```

där en partitionering av $[a, b]$ i N delintervall görs.

Sedan genereras ett funktionsrum på intervallet enligt

```
V = FunctionSpace(mesh, "CG", 1)
```

vilket i stort sett motsvarar V_h i den matematiska formuleringen i avsnitt 5.1.1 ovan,

$$V_h^0 = \left\{ v \in C[0,1] : v|_{I_i} \in \mathcal{P}^1(I_i),\, v(0) = v(1) = 0 \right\}.$$

Skillnaden är att inga randvillkor införs i rummet som i teorin. Dessa appliceras i FEniCS separat och tas om hand då en numerisk lösare senare anropas för problemet.

För att ta hand om randvillkoren definieras först funktioner för att markera olika delar av intervallet. Detta sker här enligt

```
def ua_boundary(x):
    return abs(x - a) < DOLFIN_EPS

def ub_boundary(x):
    return abs(x - b) < DOLFIN_EPS
```

Ovanstående är funktioner som returnerar det booleska uttrycktet "sant" om en viss punkt ligger inom ett givet litet avstånd från början respektive slutet av intervallet. Tillsammans med klassen `DirichletBC` i Dolfin används dessa funktioner sedan för att sätta Dirichletvillkor på intervallets rand genom kodavsnittet

```
bc_a = DirichletBC(V, u_a, ua_boundary)
bc_b = DirichletBC(V, u_b, ub_boundary)
bcs = [bc_a, bc_b]
```

Därpå sätts den svaga formuleringen av problemet upp och definieras på abstrakt form enligt FEniCS standard:

```
u = TrialFunction(V)
v = TestFunction(V)
a = Dx(u,0)*Dx(v,0)*dx
L = f*v*dx
```

där vi alltså söker en lösning till problemet: finn $u \in V$ som uppfyller

$$a(u, v) = L(v), \quad \forall v \in V.$$

Den eftersökta finita elementlösningen fås genom

```
u = Function(V)
solve(a == L, u, bcs)
```

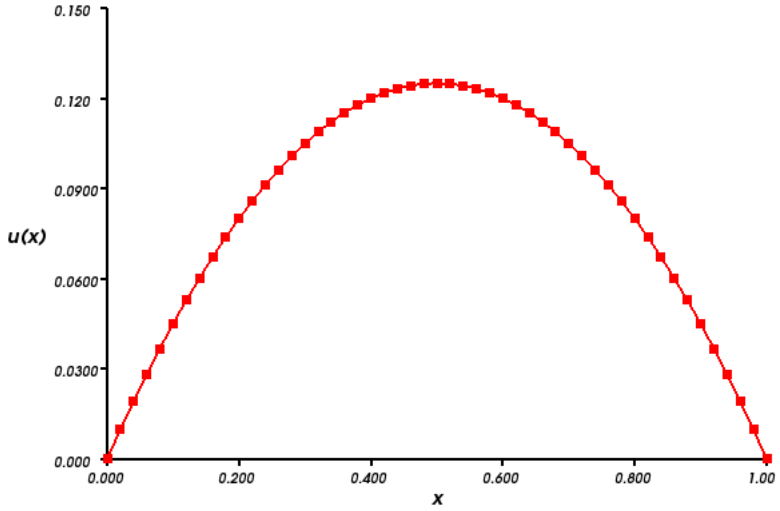
Slutligen ges en grafisk representation av lösningen och felet i approximationen av

```
u_analytic = Expression("-0.5*(x[0] - 1)*x[0]")
error = u_analytic - u

plot(u, title = "u_h")
plot(error, title = "error=u-u_h")

interactive()
```

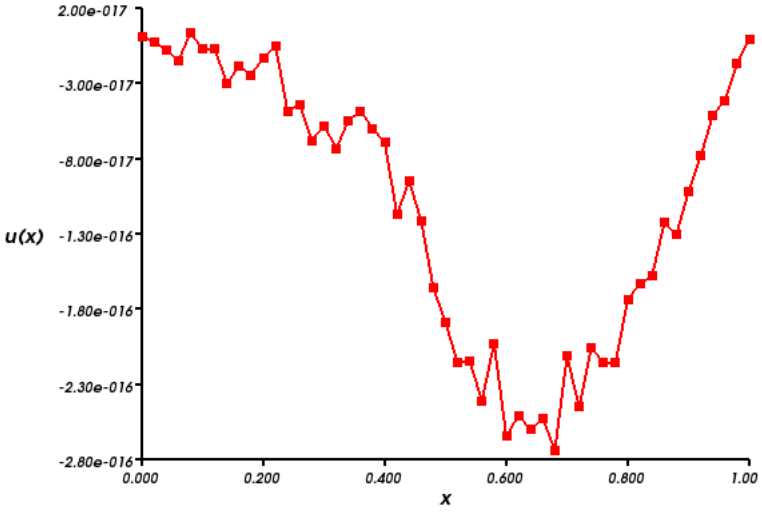
Felet kan här beräknas exakt i och med att problemet, (5.7), har den analytiska lösningen:

$$u(x) = -\frac{1}{2}(x^2 - x).$$

I Figur 5.1 visas finita elementlösningen som erhålls med ovanstående kod. Intervallet $[0, 1]$ är här partitionerat i 50 delintervall. Felet vid motsvarande approximation presenteras i Figur 5.2.



Figur 5.1: Finita elementlösning av Poissons ekvation (5.7) med $N = 50$.



Figur 5.2: Fel vid finita elementlösning av Poissons ekvation (5.7) med $N = 50$.

5.1.3 Fullständig kod

Listing 1: Poissons ekvation

```
"""
This program solves Poisson's equation in 1D

        -u''(x)=f(x), a<x<b
        x(a)=x_a, x(b)=x_b

using a continuous Galerkin method.
"""

from dolfin import *

N = 10 #Number of sub intervals.
a = 0 #Beginning of interval.
b = 1 #End of interval.
u_a = Constant(0) #Boundary condition at x=a.
u_b = Constant(0)  #Boundary condition at x=b.
f = Expression("1") #Source term in Poisson's equation.

#Creates a partition of [0,1] in N sub intervals.
mesh = Interval(N, a, b)
#Generates a finite dimensional function space of continuous piecewise
#linears with Lagrange basis of order 1 on each sub interval.
V = FunctionSpace(mesh, "CG", 1)

#Function for marking where the boundary conditions should be applied.
def ua_boundary(x):
    return abs(x - a) < DOLFIN_EPS

def ub_boundary(x):
    return abs(x - b) < DOLFIN_EPS

#Connects the boundary condition with the boundary.
bc_a = DirichletBC(V, u_a, ua_boundary)
bc_b = DirichletBC(V, u_b, ub_boundary)
bcs = [bc_a, bc_b]

#Defines variational problem in abstract form.
u = TrialFunction(V)
v = TestFunction(V)
a = Dx(u,0)*Dx(v,0)*dx
L = f*v*dx

#Redefines u as the solution function and computes solution.
u = Function(V)
solve(a == L, u, bcs)

#Analytic solution and error.
u_analytic = Expression("-0.5*(x[0] - 1)*x[0]")
error = u_analytic-u_h

plot(u, title = "u_h")
plot(error, title = "error=u-u_h")

interactive()
```

5.2 Ett begynnelsevärdesproblem

Vid lösning av optimala styrproblem med problemställning och metod definierade som i avsnitt 3, fås ett system av differentialekvationer att lösa. Dessa ekvationer måste lösas samtidigt för att ett optimum till styrproblemet ska hittas [2]. Här kommer systemet att förminskas, och endast en av ekvationerna betraktas. Efter vissa ytterligare modifikationer leder detta fram till ett begynnelsevärdesproblem i en dimension, vilket är problemet som studeras i detta avsnitt.

5.2.1 Problemet i teorin

Problemet som betraktas här lyder

$$\begin{cases} \dot{x}(t) = a(t)x(t), \quad 0 < t < T, \\ x(0) = x_0. \end{cases} \tag{5.11}$$

Innan vi går vidare och löser problemet tittar vi på hur det hänger samman med det ursprungliga problemet i avsnitt 3, vilket alltså är att finna tillstånd $x(t)$ och styrning $u(t)$ som uppfyller:

$$\min \quad \mathcal{J}(x(t), u(t)) = l(x(0), x(T)) + \int_0^T L(x(t), u(t))\, dt, \tag{5.12}$$

$$\text{under bivillkoret} \quad \begin{cases} \dot{x}(t) = f(x(t), u(t)), \quad 0 < t < T, \\ I_0 x(0) = x_0, \quad I_T x(T) = x_T, \end{cases} \tag{5.13}$$

där det gäller att

$$\begin{aligned} l &: \ \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \\ L &: \ \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}, \\ f &: \ \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d \end{aligned}$$

är glatta funktioner och $I_0, I_T \in \mathbb{R}^{d \times d}$.

Bivillkoret (5.13) multiplicerades i avsnitt 3 med en testfunktion tillhörande ett lämpligt rum och integrerades över $(0, T)$. Det gav variationsformuleringen av bivillkoret på partitioneringen

$$\mathcal{T} : \ 0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = T, \tag{5.14}$$

som alltså lyder: finn $x(t) \in \tilde{W}$ och $u(t) \in U$ som satisfierar

$$\sum_{n=1}^{N} \int_{I_n} (\dot{x} - f(x, u), \varphi)\, dt + \sum_{n=0}^{N} ([x]_n, \varphi_n) = 0, \quad \forall \varphi \in V, \tag{5.15}$$

där

$$\begin{aligned} W &= \mathbb{R} \times \{w : w|_{I_n} \in H^1(I_n, \mathbb{R}^d), n = 1, \ldots, N\} \times \mathbb{R}, \\ \tilde{W} &= \{w \in W : I_0 w_0^- = x_0, I_T w_N^+ = x_T\}, \\ V &= H^1((0, T), \mathbb{R}^d), \\ U &= H^1((0, T), \mathbb{R}^m). \end{aligned}$$

Övriga definitioner stämmer även de överens med avsnitt 3.

Om nu $f(x, u) = a(t)x(t)$ i (5.13) och $x(t) \in \mathbb{R}$ fås, med randvillkor endast vid $t = 0$, begynnelsevärdesproblemet

$$\begin{cases} \dot{x}(t) = a(t)x(t), \quad 0 < t < T, \\ x(0) = x_0, \end{cases}$$

vilket är (5.11). Ekvationen i (5.15) är därmed en svag formulering av denna differentialekvation.

Den diskretiserade versionen av (5.15) betraktas enligt avsnitt 3.2.2. Alltså införs funktionsrummen

$$W_h = \mathbb{R} \times \{w : w|_{I_n} \in \mathcal{P}^q(I_n, \mathbb{R}),\, n = 1, \ldots, N\} \times \mathbb{R}, \tag{5.16}$$

$$\tilde{W}_h = \left\{w \in W_h : w_0^- = x_0\right\}, \tag{5.17}$$

$$V_h = \left\{v \in C([0,T], \mathbb{R}) : v|_{I_n} \in \mathcal{P}^{q+1}(I_n,\, \mathbb{R})\right\}. \tag{5.18}$$

Låt oss nu studera det enklaste fallet, $q = 0$, det vill säga en funktion som är styckvis konstant på $\mathcal{T}$ söks. Då funktionerna i W_h är styckvis konstanta sätts $w_n^- = w_{n-1}^+ = w_n$ och $w|_{I_n} = w_n$. Testfunktionerna är styckvis linjära. Finita elementproblemet blir då att finna $x_h \in \tilde{W}_h$ med $x_h|_{I_n} = x_{h,n}$ sådan att:

$$\begin{aligned}\sum_{n=1}^{N} \Big(-x_{h,\,n} \int_{I_n} a(t) v(t)\, dt + (x_{h,\,n} - x_{h,\,n-1})\, v(t_{n-1}) \Big) \\ + (x_{h,\,N+1} - x_{h,\,N})\, v(t_N) = 0, \quad \forall v \in V_h.\end{aligned} \tag{5.19}$$

Tidsderivatan försvinner då ansatsfunktionen är styckvis konstant. Som bas för V_h väljs $\{\varphi_n(t)\}_{n=0}^N$ där

$$\varphi_n = \begin{cases} \dfrac{t - t_{n-1}}{t_n - t_{n-1}}, & t \in I_n, \\ \dfrac{t_{n+1} - t}{t_{n+1} - t_n}, & t \in I_{n+1}, \\ 0, \quad \text{annars.} \end{cases} \tag{5.20}$$

Basfunktionerna (5.20) gäller för $n = 1, \ldots, N-1$, medan φ_0 och φ_N ser ut som

$$\begin{aligned} \varphi_0 &= \begin{cases} \dfrac{t_1 - t}{t_1}, & t \in I_1, \\ 0, \quad \text{annars,} \end{cases} \\ \varphi_N &= \begin{cases} \dfrac{t - t_{N-1}}{t_N - t_{N-1}}, & t \in I_N, \\ 0, \quad \text{annars.} \end{cases} \end{aligned} \tag{5.21}$$

Genom att testa mot varje basfunktion och inkludera randvillkoret fås ett ekvationssystem av storlek $(N+2) \times (N+2)$ att lösa för x_h. Ekvationerna blir:

$$\begin{aligned} x_{h,\,0} &= x_0, \\ -x_{h,\,1} \int_{I_1} a\varphi_0\, dt + (x_{h,\,1} - x_{h,\,0}) &= 0, \\ -x_{h,\,1} \int_{I_1} a\varphi_1\, dt - x_{h,\,2} \int_{I_2} a\varphi_1\, dt + (x_{h,\,2} - x_{h,\,1}) &= 0, \\ &\vdots \\ -x_{h,\,n} \int_{I_n} a\varphi_n\, dt - x_{h,\,n+1} \int_{I_{n+1}} a\varphi_n\, dt + (x_{h,\,n+1} - x_{h,\,n}) &= 0, \\ &\vdots \\ -x_{h,\,N} \int_{I_N} a\varphi_N\, dt + (x_{h,\,N+1} - x_{h,\,N}) &= 0. \end{aligned}$$

De extra värdena $x_{h,0}$ och $x_{h,N+1}$ motsvarar de yttre randvärdena av intervallet $(0,T)$.

Om $a(t) = a$ är konstant fås det ekvivalenta systemet $Ax = b$, där matrisen A är

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & (1-a\frac{h}{2}) & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & -(1+a\frac{h}{2}) & (1-a\frac{h}{2}) & 0 & \cdots & \cdots & 0 \\ 0 & 0 & -(1+a\frac{h}{2}) & (1-a\frac{h}{2}) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & -(1+a\frac{h}{2}) & (1-a\frac{h}{2}) & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & -(1+a\frac{h}{2}) & 1 \end{bmatrix}. \tag{5.22}$$

Här har vi antagit konstant intervallängd, h, för alla delintervall I_n. De båda vektorerna x och b är

$$x = \begin{bmatrix} x_{h,\,0} \\ x_{h,\,1} \\ \vdots \\ x_{h,\,N} \\ x_{h,\,N+1} \end{bmatrix}, \quad b = \begin{bmatrix} x_0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{5.23}$$

5.2.2 Numerisk metod

För $n = 1, \ldots, N-1$ kan, enligt ekvationssystemet ovan, $x_{h,\,n+1}$ lösas ut i termer av $x_{h,\,n}$ enligt

$$x_{h,\,n+1} = \frac{1+a\frac{h}{2}}{1-a\frac{h}{2}} x_{h,\,n}. \tag{5.24}$$

Här noterar vi att (5.24) motsvarar Crank-Nicolsons finita differensmetod [3]. Motsvarande ekvationer för de första och sista värdena lyder:

$$x_{h,\,1} = \frac{1}{1-a\frac{h}{2}} x_{h,\,0}, \tag{5.25}$$

$$x_{h,\,N+1} = (1+a\tfrac{h}{2}) x_{h,\,N}. \tag{5.26}$$

Dessa känns igen som varianter av Eulers framåt- respektive bakåtmetod [3].

5.2.3 Problemet i FEniCS

En finita elementlösning till begynnelsevärdesproblemet (5.11)

$$\begin{cases} \dot{x}(t) = a(t)x(t), & 0 < t < T, \\ x(0) = x_0 \end{cases}$$

enligt metoden beskriven i avsnitt 5.2.1 söks nu. Efter de studier vi gjort i FEniCS har det visat sig att programvaran inte automatiskt är lämpat för detta. Problemet ligger till största del i definitionen av funktionsrummet W_h som i (5.16)

$$W_h = \mathbb{R} \times \{w : w|_{I_n} \in \mathcal{P}^q(I_n, \mathbb{R}),\, n = 1, \ldots, N\} \times \mathbb{R}.$$

I den matematiska teorin definieras två extra termer som ligger utanför det egentliga intervallet $(0,T)$. För $w \in W_h$ kallas dessa w_0^- och w_N^+ och definieras som de ensidiga gränsvärdena

$$w_0^- = \lim_{t \to 0^-} w(t), \quad w_N^+ = \lim_{t \to T^+} w(t).$$

Dessa båda termer är inte inkluderade i funktionsrummen som skapas med fördefinierade klasser i Dolfin. Vissa metoder för att komma runt detta har dock studerats, mer om dessa nedan.

5.2.4 Ett försök att implementera metoden i avsnitt 5.2.1

Ett rättframt sätt att försöka lösa problemet är att definiera funktionsrummen W_h och V_h som i teorin och sätta upp variationsformuleringen av begynnelsevärdesproblemet på FEniCS sätt. Här följer, rad för rad, en Pythonkod som åstadkommer detta.

Först införs objekt som specificerar problemet, det vill säga ger värden till $a(t)$, T och x_0. Detta görs enligt

```
at = Expression("1")
T = 1
x_0 = Constant(1)
```

Dessa värden är inte unika, men något måste väljas. Inom rimliga gränser kan godtyckliga värden ges.

Därefter görs partitioneringen av intervallet genom raderna

```
N = 10
mesh = Interval(N, 0, T)
```

Härnäst definieras funktionsrummen W_h och V_h i FEniCS som

```
q = 0
W_h = FunctionSpace(mesh, "DG", q)
V_h = FunctionSpace(mesh, "CG", q + 1)
```

Igen noteras att W_h i FEniCS alltså inte stämmer exakt överens med motsvarande funktionsrum i teorin. Genom att ändra värdet på q fås en finita elementmetod med styckvisa polynom av högre ordning.

FEniCS skiljer på yttre och inre fasetter[15], vilket motsvarar noder i detta endimensionella fall. För att handskas med det som händer i ändpunkterna av intervallet $(0, T)$ införs därför klasser som bestämmer huruvida en punkt ligger på randen eller inte. Det görs som följer:

```
class InitialBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and abs(x[0]) < DOLFIN_EPS

class EndBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and abs(x[0] - T) < DOLFIN_EPS
```

Dessa behöver kopplas till tidsintervallet och objektet `mesh` ovan. I FEniCS kan detta göras genom att införa

```
facet_dimension = mesh.topology().dim() - 1
boundary_parts = MeshFunction("uint", mesh, facet_dimension)
```

som ser till att punkterna som utgör intervallets rand kan behandlas separat.

Nu skapas instanser av klasserna ovan. De två randpunkterna markeras med ett nummer vardera enligt

[15] För definition av fasett se Appendix A.12.

```
start_time = InitialBoundary()
start_time.mark(boundary_parts, 0)

final_time = EndBoundary()
final_time.mark(boundary_parts, 1)
```

Vidare definieras ansats- och testfunktionerna. Den svaga formuleringen sätts upp på abstrakt form på FEniCS sätt. Detta motsvarar den matematiska formuleringen (5.19) och görs genom

```
x = TrialFunction(W_h)
v = TestFunction(V_h)

a = -at*x*v*dx - jump(x)*avg(v)*dS - x*v*ds(1) + x*v*ds(0)

f = Constant(0)
L = f*v*dx
```

Här noteras att ett minustecken föregår termen som innehåller `jump`. Termen tar hand om diskontinuiteter i de inre noderna. I det teoretiska finita elementproblemet, vilket lyder

$$\sum_{n=1}^{N} \left(-x_{h,\,n} \int_{I_n} a(t) v(t)\, dt + (x_{h,\,n} - x_{h,\,n-1})\, v(t_{n-1}) \right) + (x_{h,\,N+1} - x_{h,\,N})\, v(t_N) = 0, \quad \forall v \in V_h,$$

förekommer hoppen alltså som

$$\sum_{n=1}^{N+1} (x_{h\,,n} - x_{h,\,n-1})\, v(t_{n-1}) \tag{5.27}$$

med ett plustecken framför summationstecknet. Hoppet över en diskontinuitet är där definierat som $(x_{h,\,n} - x_{h,\,n-1})$. Minustecknet i FEniCS-formuleringen följer efter definitionen av metoden `jump`, vilken återfinns i kapitel 4.

Termen `avg(v)` ersätter v i finita elementformuleringen vid summering över de inre noderna `(*dS)`. Detta beror på att FEniCS förväntar sig en diskontinuerlig testfunktion när ansatsfunktionen är diskontinuerlig. Om t_n är en nod i partitioneringen $\mathcal{T}$, se (5.14), och w är en styckvis kontinuerlig funktion på $\mathcal{T}$ existerar inte $w(t_n)$, utan endast de ensidiga gränsvärdena

$$\lim_{t \to t_n^-} w(t) \text{ och } \lim_{t \to t_n^+} w(t).$$

Därmed är $w(t_n)$ inte entydigt bestämt och vi måste specificera huruvida ett av gränsvärdena eller medelvärdet av dem ska definiera $w(t_n)$. I detta fall, då testfunktionen v är kontinuerlig, sammanfaller de båda gränsvärdena med deras medelvärde. Alltså kan medelvärdet av de båda gränsvärdena användas som ekvivalent till funktionsvärdet i punkten.

De två termerna `x*v*ds(1)` och `x*v*ds(0)` i den abstrakta formuleringen motsvarar integration över de yttre noderna. I en dimension likställs det med multiplikation. Testfunktionen v uppfyller $v(0) = 1$ endast då v ersätts med φ_0 och $v(T) = 1$ endast då v ersätts med φ_N. Övriga basfunktioner saknar stöd i randpunkterna. Det gör att de två sista termerna motsvarar $+x_{h,1}$ och $-x_{h,N}$ i (5.27) ovan, och detta endast i de ekvationer som svarar mot φ_0 och φ_N.

Ekvationssystemet som följer genom att testa mot respektive basfunktion i V_h fås från

```
A = assemble(a, exterior_facet_domains = boundary_parts)
b = assemble(L, exterior_facet_domains = boundary_parts)
```

Det extra argumentet till metoden, `exterior_facet_domains = boundaryparts`, behövs för att integrationen över randpunkterna inte ska ignoreras.

Funktionsrummet W_h har i FEniCS dimension N, där N är antalet delintervall och $q = 0$ har antagits. Rummet av testfunktioner, V_h, har under samma antaganden dimension $N+1$. Det gör att antalet ekvationer blir ett fler än antalet obekanta i det system som sätts samman i FEniCS. Ekvationssystemet kan således inte lösas med FEniCS metod `solve`, som kräver ett kvadratiskt system. Initialvillkoret har inte heller tagits om hand i ovanstående kod. I avsnitt 5.2.6 studerar vi hur dessa problem kan lösas.

5.2.5 Ekvationssystemet i FEniCS

Rummet av testfunktioner,

$$V_h = \left\{ v \in C([0,T], \mathbb{R}) : v|_{I_n} \in \mathcal{P}^{q+1}(I_n, \mathbb{R}) \right\},$$

stämmer överens i FEniCS och i teorin. Ekvationssystemet som sätts samman i FEniCS fås på samma sätt som i avsnitt 5.2.1 genom att testa mot varje basfunktion för V_h. För $q = 0$ är basfunktionerna i FEniCS desamma som i teorin, det vill säga

$$\varphi_0 = \begin{cases} \dfrac{t_1 - t}{t_1}, & t \in I_1, \\ 0, & \text{annars}, \end{cases}$$

$$\varphi_n = \begin{cases} \dfrac{t - t_{n-1}}{t_n - t_{n-1}}, & t \in I_n, \\ \dfrac{t_{n+1} - t}{t_{n+1} - t_n}, & t \in I_{n+1}, \\ 0, & \text{annars}, \end{cases} \quad n = 1, \ldots, N-1,$$

$$\varphi_N = \begin{cases} \dfrac{t - t_{N-1}}{t_N - t_{N-1}}, & t \in I_N, \\ 0, & \text{annars}. \end{cases}$$

Basfunktionerna för W_h stämmer överens förutom att de två extra värdena i teorin inte finns med i FEniCS. På samma sätt som i avsnitt 5.2.1 ger detta ett ekvationssystem $Ax = b$, denna gången av storlek $(N+1) \times N$, där A nu beskrivs av

$$\begin{bmatrix} (1 - a\frac{h}{2}) & 0 & \cdots & \cdots & 0 \\ -(1 + a\frac{h}{2}) & (1 - a\frac{h}{2}) & 0 & \cdots & 0 \\ 0 & -(1 + a\frac{h}{2}) & (1 - a\frac{h}{2}) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -(1 + a\frac{h}{2}) & (1 - a\frac{h}{2}) \\ 0 & \cdots & \cdots & 0 & -(1 + a\frac{h}{2}) \end{bmatrix} \tag{5.28}$$

och de båda vektorerna x och b ges av

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{5.29}$$

Här har vi noterat att ekvationssystemet som fås i FEniCS, (5.28)-(5.29), är en del av systemet som fås i teorin, alltså

$$A_{\text{teori}} = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & (1-a\frac{h}{2}) & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & -(1+a\frac{h}{2}) & (1-a\frac{h}{2}) & 0 & \cdots & \cdots & 0 \\ 0 & 0 & -(1+a\frac{h}{2}) & (1-a\frac{h}{2}) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & -(1+a\frac{h}{2}) & (1-a\frac{h}{2}) & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & -(1+a\frac{h}{2}) & 1 \end{bmatrix},$$

$$x_{\text{teori}} = \begin{bmatrix} x_{h,\,0} \\ x_{h,\,1} \\ \vdots \\ x_{h,\,N} \\ x_{h,\,N+1} \end{bmatrix}, \quad b_{\text{teori}} = \begin{bmatrix} x_0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

De båda matriserna stämmer överens bortsett från den första raden samt de första och sista kolumnerna i matrisen, A_{teori}, som fås med den finita elementmetod Kraft använder [2]. För vektorerna motsvarar detta att eliminera det första och sista elementet i x_{teori} samt det första elementet i b_{teori}. Ett problem med att göra detta är att begynnelsevärdet $x(0) = x_0$ ignoreras, vilket är vad som händer i den FEniCS-kod som presenteras i avsnitt 5.2.4. Därför måste det läggas till på annat sätt.

En lösning till problemet med det överbestämda ekvationssystemet i FEniCS är att bortse från det sista värdet i finita elementlösningen som söks, $x_{h,\,N+1}$. Det motsvarar att inte testa mot den sista basfunktionen, φ_N, i V_h. För ekvationssystemet som fås i FEniCS innebär det att den sista raden i matris och högerled tas bort.

Återstår gör att ta hänsyn till begynnelsevärdet vid $t = 0$. För detta syfte betraktas de två första ekvationerna i det teoretiska ekvationssystemet:

$$x_{h,\,0} = x_0, \tag{5.30}$$

$$-x_{h,\,0} + (1 - a\frac{h}{2})x_{h,\,1} = 0. \tag{5.31}$$

Genom att sätta in (5.30) i (5.31) kan den första ekvationen elimineras. Istället förflyttas värdet av $x_{h,\,0}$, som är känt men inte explicit finns med i FEniCS-formuleringen, till högerledet av (5.31). Detta gör att de två ekvationerna ovan övergår i

$$(1 - a\frac{h}{2})x_{h,\,1} = x_0.$$

Denna ekvation kan tas om hand i FEniCS genom att ändra högerledets första elementet i (5.29) från 0 till x_0, vilket ger ett system som i huvudsak stämmer överens med det i teorin. Skillnaden är att ekvationerna som innefattar de extra värdena utanför intervallet inte är med.

5.2.6 Lösning av problemet i FEniCS

För att kunna lösa problemet i FEniCS är alltså tanken nu att vi ska göra ändringar i det ekvationssystem som fås från koden i avsnitt 5.2.4. Det har visat sig att de matriser och vektorer FEniCS ger med metoden `assemble` inte direkt går att manipulera. För att kunna göra det måste de underliggande NumPy-objekten [14] kommas åt. Detta görs enligt

```
A_modified = A.array()
b_modified = b.array()
```

Sedan modifieras systemet som beskrivet i avsnitt 5.2.5 ovan:

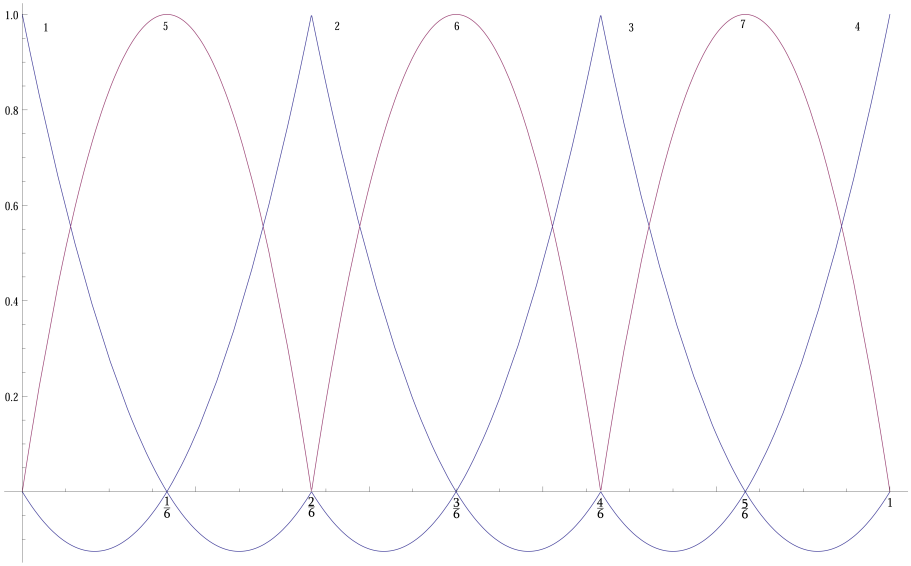
```
A_modified = delete(A_modified, N, 0)
b_modified = delete(b_modified, N, 0)
b_modified[0] = x_0
```

det vill säga den sista raden i matris och vektor elimineras, samtidigt som begynnelsevärdet läggs in i ekvationssystemets högerled.

Raden som måste tas bort är den som motsvarar basfunktionen för V_h som i $t = T$ antar värdet 1. Med styckvis linjära funktioner har detta visat sig vara den sista, $(N + 1)$:sta, raden i den matris och vektor vi får i FEniCS. Att det står N i kodavsnittet ovan beror på att vektorerna indexeras från 0. Även för högre ordningens polynom är den $(N + 1)$:sta raden resultatet av testning mot basfunktionen som antar värdet 1 i T. Koden är därmed korrekt oberoende av polynomens gradtal. Att just den $(N + 1)$:sta raden motsvarar denna basfunktion beror på hur basfunktionerna ordnas i FEniCS.

Antag att polynomen i V_h är av grad $q+1$ och att intervallet består av N delintervall. Då kommer $N+1$ noder dela in intervallet, medan varje delintervall har q inre noder. FEniCS testar då först mot den Lagrangebasfunktion som antar värdet 1 i den första noden, motsvarande $t = 0$, och fortsätter sedan successivt mot slutet av intervallet och den Lagrangefunktion som är 1 i den $(N + 1)$:sta och sista nod som utgör en del av randen till ett delintervall. Detta görs oberoende av hur många inre noder varje delintervall har, det vill säga oberoende av polynomens gradtal. Därefter tas delintervallen om hand.

Lagrangebasen på varje delintervall består av $q + 2$ polynom, varav q stycken antar värdet 1 i en inre nod. Intervallen gås igenom ett efter ett innan samma sak upprepas på nästföljande delintervall till dess att alla basfunktioner har tagits med. I Figur 5.3 illustreras hur detta fungerar för Lagrangepolynom av grad två. Varje basfunktion har tilldelats ett nummer och basfunktionerna testas i nummerordning.



Figur 5.3: Lagrangebas av grad 2 på en partitionering av $[0, 1]$ i 3 lika långa intervall.

Därefter löses ekvationssystemet med hjälp av NumPy:

```
solution = linalgSolve(A_modified, b_modified)
```

Vektorn `solution` innehåller nu den sökta finita elementlösningen. För fortsatt analys av lösningen med FEniCS metoder är det lämpligt att den sparas i ett FEniCS-objekt. Detta åstadkoms med raderna

```
x_h = uBLASVector(n*(q + 1))
x_h.zero()

for i in range(n*(q + 1)):
    insertIndex = array([i], dtype = "I")
    toInsert = array(solution[i])
    x_h.add(toInsert, insertIndex)

print x_h.array()
```

där först en tom vektor som ska hålla lösningen skapas. Lösningen kopieras sedan till denna vektor och skrivs ut på skärmen.

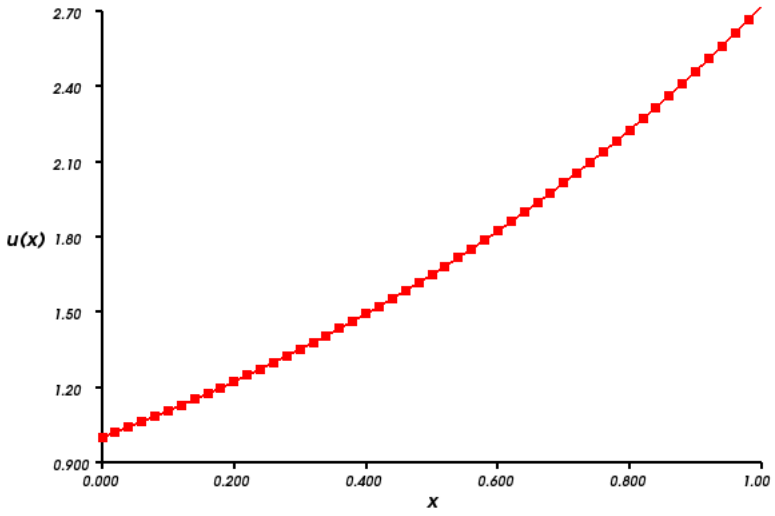
Sist görs lösningen om till ett FEniCS-objekt som kan presenteras i en graf med FEniCS metoder. Även felet i approximationen beräknas och presenteras grafiskt enligt

```
x_h = Function(W_h, x_h)
plot(x_h, title = "x_h")

x_analytic = Expression("exp(x[0])")
error = x_analytic - x_h
plot(error, title = "Error")

interactive()
```

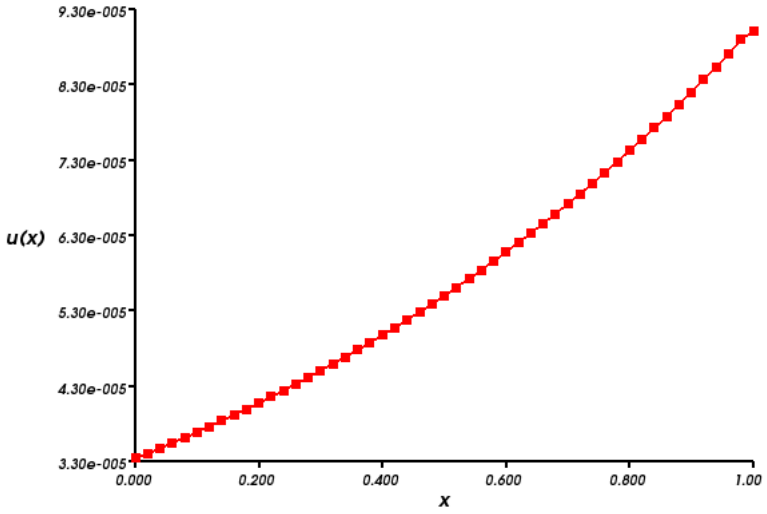
Felet kan här beräknas exakt i och med att problemet har den analytiska lösningen

$$x(t) = e^t.$$

I Figur 5.4 presenteras den finita elementlösning som fås med ovanstående kod. Här har en partitionering bestående av 50 delintervall använts. Felet vid finita elementapproximationen visas i Figur 5.5.



Figur 5.4: Finita elementlösning av begynnelsevärdesproblemet (5.11) med $N = 50$.



Figur 5.5: Fel vid finita elementlösning av begynnelsevärdesproblemet (5.11) med $N = 50$.

5.2.7 Fullständig kod

Listing 2: Ett begynnelsevärdesproblem

```
"""
This program solves the initial value problem

    dx(t)/dt=a(t)x(t)
    x(0)=x_0

using a FEM with piecewise constant, discontinuous trial function and
continuous piecewise linears as test function.
"""

from dolfin import *
from numpy import *
from numpy.linalg import solve as linalgSolve

at = Expression("1") #a(t) in the problem.
T = 1 #End time.
x_0 = Constant(1) #Initial value.

#Defines time mesh.
N = 100 #Number of sub intervals
mesh = Interval(N, 0, T)

#Defines function spaces.
q = 0 #Degree of trial space polynomials.
W_h = FunctionSpace(mesh, "DG", q)
V_h = FunctionSpace(mesh, "CG", q + 1)

#Classes for marking different parts of the boundary.
class InitialBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and abs(x[0]) < DOLFIN_EPS

class EndBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary and abs(x[0] - T) < DOLFIN_EPS
```

```
#Creates a mesh function to represent subdomains of the boundary.
facet_dimension = mesh.topology().dim() - 1 #The facets are nodes in this
     case
boundary_parts = MeshFunction("uint", mesh, facet_dimension)

#Creates instance of the boundary classes.
lower_boundary = InitialBoundary()
lower_boundary.mark(boundary_parts, 0)
upper_boundary = EndBoundary()
upper_boundary.mark(boundary_parts, 1)

#Defines test and trial functions.
x = TrialFunction(W_h)
v = TestFunction(V_h)

#Defines the abstract formulation.
a = (Dx(x,0) - at*x)*v*dx - jump(x)*avg(v)*dS - x*v*ds(1) + x*v*ds(0)
#The last terms make sure we get the same equations as with Kraft's
    method.

f = Constant(0) #rhs in abstract formulation is zero.
L = f*v*dx

#Assembles system and connects the ds(i) symbols to A and L.
A = assemble(a, exterior_facet_domains = boundary_parts)
b = assemble(L, exterior_facet_domains = boundary_parts)

#Defines NumPy matrix and vector to modify according to the method in
    Kraft's thesis.
A_modified = A.array()
b_modified = b.array()
A_modified = delete(A_modified, n, 0)
b_modified = delete(b_modified, n, 0)
b_modified[0] = x_0 #The leftmost value in the method in Kraft's thesis
    is inserted into the rhs.

#Solves the problem using NumPy's solve.
solution = linalgSolve(A_modified, b_modified)

#Creates a vector to get a FEniCS representation of the solution.
x_h = uBLASVector(n*(q + 1))
x_h.zero()

#Inserts the solution into the vector.
for i in range(n*(q + 1)):
    insertIndex = array([i], dtype = "I")
    toInsert = array(solution[i])
    x_h.add(toInsert, insertIndex)

print x_h.array()

#Creates a FEniCS Function on W_h of the solution.
x_h = Function(W_h, x_h)
plot(x_h, title = "x_h")

#analytic solution and error
x_analytic = Expression("exp(x[0])")
error = x_analytic - x_h
plot(error, title = "Error")

interactive()
```

5.3 Kvadratisk-linjära optimala styrproblem

I detta avsnitt studeras optimala styrproblem på kvadratisk-linjär form. Teorin som lades fram i avsnitt 3.3 uttnyttjas för att lösa problemen, dels analytiskt och dels med en finitia elementmetod implementerad i FEniCS.

Teorin i detta exempel bygger alltså på kapitel 3 och därför utelämnas vissa definitioner och beteckningar för att hålla texten koncis. Om inte annat anges överenstämmer därmed definitioner och andra detaljer med dem i kapitel 3.

5.3.1 Problemet i teorin

Studien grundar sig i kvadratisk-linjära optimala styrproblem på följande form: hitta $x(t)$ och $u(t)$ som

$$\begin{aligned} \text{minimerar} \quad & \mathcal{J}(x(t),u(t)) = \|x(0)\|_{S_0}^2 + \|x(T)\|_{S_T}^2 \\ & \qquad + \int_0^T (\|x\|_Q^2 + \|u\|_R^2)\,dt, \end{aligned} \tag{5.32}$$

$$\text{under bivillkoret} \quad \begin{cases} \dot{x}(t) = A(t)x(t) + B(t)u(t), \quad 0 < t < T, \\ I_0 x(0) = x_0, \quad I_T x(T) = x_T, \end{cases} \tag{5.33}$$

där $A(t), I_0, I_T, Q(t), S_0, S_T \in \mathbb{R}^{d\times d}$, $B(t) \in \mathbb{R}^{d\times m}$ och $R(t) \in \mathbb{R}^{m\times m}$. I_0 och I_T är binärt diagonala.

Som i kapitel 3 adderas en Lagrangemultiplikator till $\mathcal{J}(x,u)$. Den resulterande summan deriveras och följande problem fås: hitta $(x,u,z) \in \tilde{W} \times U \times V$ som uppfyller

$$\begin{aligned} \sum_{n=1}^{N} & \int_{I_n} (\varphi_x, 2Qx - \dot{z} - A^T z)\,dt \\ & + (\varphi_{x,0}^-, 2S_0 x_0^- - z_0) \\ & + (\varphi_{x,N}^+, 2S_T x_N^+ + z_N) = 0, \quad \forall \varphi_x \in \dot{W}, \end{aligned} \tag{5.34}$$

$$\int_0^T (\varphi_u, 2Ru - B^T z)\,dt = 0, \quad \forall \varphi_u \in U, \tag{5.35}$$

$$\sum_{n=1}^{N} \int_{I_n} (\dot{x} - Ax - Bu, \varphi_z)\,dt + \sum_{n=0}^{N} ([x]_n, \varphi_{z,n}) = 0, \quad \forall \varphi_z \in V. \tag{5.36}$$

Eftersom (5.34), (5.35) och (5.36) ska gälla för alla testfunktioner i respektive funktionsrum går det att visa att skalärprodukterna måste vara identiskt noll [4]. Denna iaktagelse innebär att följande ekvationssystem ska vara uppfyllt för kontinuerliga lösningar till styrproblemet:

$$\begin{aligned} 2Qx - \dot{z} - A^T z &= 0, \\ (I - I_0)(2S_0 x(0) - z(0)) &= 0, \\ (I - I_T)(2S_T x(T) + z(T)) &= 0, \\ 2Ru - B^T z &= 0, \\ \dot{x} - Ax - Bu &= 0, \\ I_0 x(0) &= x_0, \\ I_T x(T) &= x_T. \end{aligned} \tag{5.37}$$

5.3.2 Analytisk lösning av två optimala styrproblem

Nedan löses två kvadratisk-linjära optimala styrproblem analytiskt då $d = m = 1$. Ekvationsystemet (5.37) används för att beräkna de analytiska lösningarna.

Exempel 1

Hitta $x(t)$ och $u(t)$ som

$$\begin{aligned} &\text{minimerar} \quad \mathcal{J}(x(t),u(t)) = \frac{1}{2}x(T)^2 + \int_0^T \frac{1}{2}u^2\,dt, \\ &\text{under bivillkoret} \quad \begin{cases} \dot{x}(t) = u, \quad 0 < t < T, \\ x(0) = x_0. \end{cases} \end{aligned}$$

De koefficienter som specifierar styrproblemet är alltså

$$\begin{aligned} A = I_T = Q = S_0 = 0, \\ R = S_T = \frac{1}{2}, \\ B = I_0 = 1. \end{aligned}$$

Från (5.37) fås därmed följande ekvationssystem:

$$\begin{cases} \dot{z} = 0, & (5.38) \\ x(T) + z(T) = 0, & (5.39) \\ u - z = 0, & (5.40) \\ \dot{x} - u = 0, & (5.41) \\ x(0) = x_0. & (5.42) \end{cases}$$

Ekvation (5.38) ger att $z(t) = C$, där C är konstant, vilket i kombination med (5.39) medför att $C = -x(T)$. Används vidare ekvation (5.40) fås att $u = -x(T)$. Detta tillsammans med differentialekvationen (5.41) ger att

$$x(t) = -x(T)t + D,$$

som med initialvillkoret (5.42) blir

$$x(t) = -x(T)t + x_0.$$

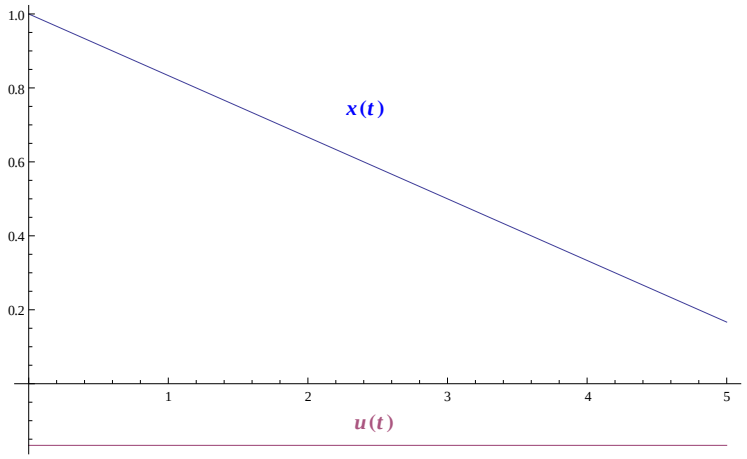
Om nu t sätts till T i ekvationen för $x(t)$, fås

$$x(T) = -x(T)T + x_0 \Longrightarrow x(T) = \frac{x_0}{1+T}.$$

Lösningen till styrproblemet är alltså

$$\begin{cases} x(t) = -\dfrac{x_0}{1+T}t + x_0, \\ u(t) = -\dfrac{x_0}{1+T}. \end{cases}$$

Se Figur 5.6 för en grafisk representation av lösningen då $x_0 = 1$ och $T = 5$.



Figur 5.6: Lösningen $x(t)$ och $u(t)$ till Exempel 1 då $x_0 = 1$ och $T = 5$.

Exempel 2

Hitta $x(t)$ och $u(t)$ som

$$\text{minimerar} \quad \mathcal{J}(x(t), u(t)) = \frac{1}{2}x(0)^2 + \int_0^1 \left(\frac{1}{2}x^2 + \frac{1}{2}u^2\right) dt,$$

$$\text{under bivillkoret} \quad \begin{cases} \dot{x}(t) = x + u, & 0 < t < 1, \\ x(1) = 1. \end{cases}$$

Koefficienterna som specificerar problemet är

$$\begin{aligned} I_0 = S_T &= 0, \\ Q = R = S_0 &= \frac{1}{2}, \\ A = B = I_T &= 1. \end{aligned}$$

Från (5.37) ges därmed följande ekvationssystem:

$$\begin{cases} x - \dot{z} - z = 0, & (5.43) \\ x(0) - z(0) = 0, & (5.44) \\ u - z = 0, & (5.45) \\ \dot{x} - x - u = 0, & (5.46) \\ x(1) = 1. & (5.47) \end{cases}$$

Ekvation (5.45) och (5.46) medför att

$$z = \dot{x} - x \Longrightarrow \dot{z} = \ddot{x} - \dot{x},$$

som insatt i (5.43) ger

$$x - (\ddot{x} - \dot{x}) - (\dot{x} - x) = 0 \Longrightarrow \ddot{x} = 2x.$$

Detta är en andra ordningens linjär differentialekvation som har lösningen

$$x(t) = c_1 e^{\sqrt{2}t} + c_2 e^{-\sqrt{2}t}.$$

Ekvation (5.45) och (5.46) ger att

$$z(t) = u(t) = c_1\sqrt{2}e^{\sqrt{2}t} - c_2\sqrt{2}e^{-\sqrt{2}t} - c_1 e^{\sqrt{2}t} - c_2 e^{-\sqrt{2}t}.$$

Randvillkoren (5.44) och (5.47) leder till följande ekvationssystem:

$$\begin{cases} c_1(\sqrt{2}-2) = c_2(\sqrt{2}+2), \\ c_1 e^{\sqrt{2}} + c_2 e^{-\sqrt{2}} = 1, \end{cases}$$

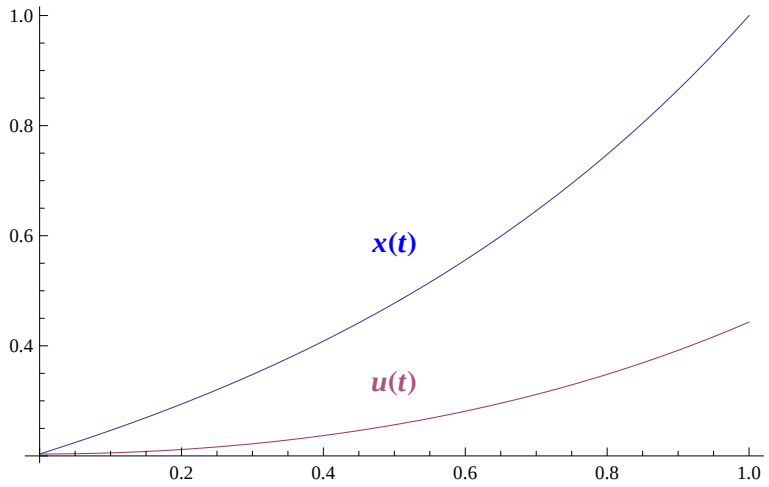
som har lösningen:

$$\begin{cases} c_1 = \dfrac{(\sqrt{2}+2)e^{\sqrt{2}}}{\sqrt{2}-2+(\sqrt{2}+2)e^{2\sqrt{2}}}, \\ c_2 = \dfrac{(\sqrt{2}-2)e^{\sqrt{2}}}{\sqrt{2}-2+(\sqrt{2}+2)e^{2\sqrt{2}}}. \end{cases}$$

Efter insättning och omskrivning fås följande lösning till styrproblemet:

$$\begin{cases} x(t) = \dfrac{\sqrt{2}\cosh(\sqrt{2}t) + 2\sinh(\sqrt{2}t)}{\sqrt{2}\cosh(\sqrt{2}) + 2\sinh(\sqrt{2})}, \\ u(t) = \dfrac{\sqrt{2}\cosh(\sqrt{2}t)}{\sqrt{2}\cosh(\sqrt{2}) + 2\sinh(\sqrt{2})}. \end{cases}$$

Se Figur 5.7 för en grafisk representation av $x(t)$ och $u(t)$.



Figur 5.7: Lösningen $x(t)$ och $u(t)$ till Exempel 2.

I avsnitt 5.3.4 presenteras ett FEniCS-program som löser ovanstående exempel med den finita elementmetod Kraft använder.

5.3.3 Beräkning av finita elementmatrisen

De kvadratisk-linjära styrproblemen med $d = m = 1$ ger motsvarande finita elementformulering: hitta $(x_h, u_h, z_h) \in W_h \times U_h \times V_h$ så att

$$\begin{aligned} &\sum_{n=1}^{N} \int_{I_n} (2Qx_h - \dot{z}_h - A^T z_h)\varphi_x \, dt \\ &\quad + \varphi_{x,0}^{-}(2S_0 x_{h,0}^{-} - z_{h,0}) \\ &\quad + \varphi_{x,N}^{+}(2S_T x_{h,N}^{+} + z_{h,N}) = 0, \quad \forall \varphi_x \in \dot{W}_h, \end{aligned} \tag{5.48}$$

$$\int_0^T (2Ru_h - B^T z_h)\varphi_u \, dt = 0, \quad \forall \varphi_u \in U_h, \tag{5.49}$$

$$\sum_{n=1}^{N} \int_{I_n} (\dot{x}_h - Ax_h - Bu_h)\varphi_z \, dt + \sum_{n=0}^{N} [x_h]_n \varphi_{z,n} = 0, \quad \forall \varphi_z \in V_h, \tag{5.50}$$

$$I_0 x_{h,0}^{-} = x_0, \quad I_T x_{h,N}^{+} = x_T. \tag{5.51}$$

För att beräkna matrisekvationen som ovanstående finita elementproblem resulterar i görs först följande partionering:

$$0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = T,$$
$$I_j = (t_{j-1}, t_j), \quad t_j - t_{j-1} = h, \quad \forall j.$$

Sedan definieras basfunktionerna för de tre funktionsrummen. För W_h används

$$\phi_0 = \begin{cases} 1, & t = 0, \\ 0, & \text{annars}, \end{cases}$$
$$\phi_j = \begin{cases} 1, & t \in I_j, \\ 0, & \text{annars}, \end{cases} \quad j = 1, \ldots, N,$$
$$\phi_{N+1} = \begin{cases} 1, & t = T, \\ 0, & \text{annars}. \end{cases}$$

För U_h och V_h används basfunktionerna

$$\varphi_0 = \begin{cases} \dfrac{t_1 - t}{h}, & t \in I_1, \\ 0, & \text{annars}, \end{cases}$$
$$\varphi_j = \begin{cases} \dfrac{t - t_{j-1}}{h}, & t \in I_j, \\ \dfrac{t_{j+1} - t}{h}, & t \in I_{j+1}, \\ 0, & \text{annars}, \end{cases} \quad j = 1, \ldots, N-1,$$
$$\varphi_N = \begin{cases} \dfrac{t - t_{N-1}}{h}, & t \in I_N, \\ 0, & \text{annars}. \end{cases}$$

Vidare ansätts de tre finita elementlösningarna på följande sätt:

$$x_h = \sum_{i=0}^{N+1} \xi_i \phi_i, \quad u_h = \sum_{i=0}^{N} \zeta_i \varphi_i, \quad z_h = \sum_{i=0}^{N} \eta_i \varphi_i.$$

Studera nu följande omskrivning av (5.48):

$$\begin{aligned} \int_0^T \Big(2Q \sum_{i=0}^{N+1} \xi_i \phi_i - \sum_{i=0}^{N} \eta_i \dot{\varphi}_i - A \sum_{i=0}^{N} \eta_i \varphi_i \Big) \phi_j \, dt \\ + \phi_j^-(0)(I - I_0)(2S_0\xi_0 - \eta_0) \\ + \phi_j^+(T)(I - I_T)(2S_T\xi_{N+1} + \eta_N) = 0, \qquad j = 0, \ldots, N+1, \end{aligned} \tag{5.52}$$

där ansatserna för x_h och z_h satts in. Villkoret som säger att ekvationen ska gälla för alla testfunktioner i $\dot{W}_h$ har ersatts med att testa mot basfunktionerna för W_h tillsammans med faktorerna $I - I_0$ och $I - I_T$.

På liknande sätt insätts ansatserna av u_h och z_h i ekvation (5.49):

$$\int_0^T \Big(2R \sum_{i=0}^{N} \zeta_i \varphi_i - B \sum_{i=0}^{N} \eta_i \varphi_i \Big) \varphi_j \, dt = 0, \quad j = 0, \ldots, N. \tag{5.53}$$

Den tredje ekvationen, (5.50), blir

$$\begin{aligned} -\int_0^T \Big(A \sum_{i=0}^{N+1} \xi_i \phi_i + B \sum_{i=0}^{N} \zeta_i \varphi_i \Big) \varphi_j \, dt + (\xi_1 - \xi_0)\varphi_j(0) \\ + (\xi_2 - \xi_1)\varphi_j(t_1) + \cdots + (\xi_{N+1} - \xi_N)\varphi_j(t_N), \quad j = 0, \ldots, N, \end{aligned} \tag{5.54}$$

eftersom $\dot{\phi}_j = 0$ på $(0,T)$. Randvillkoren (5.51) blir

$$I_0\xi_0 = x_0, \quad I_T\xi_{N+1} = x_T. \tag{5.55}$$

På flera ställen i ekvationerna ovan förekommer integraler där integranden består av två basfunktioner multiplicerade med varandra. Därför beräknas först två sådana integraler:

$$\begin{aligned}\int_{I_j} \varphi_j^2\,dt &= \int_{I_j} \Big(\frac{t-t_{j-1}}{h}\Big)^2\,dt = \frac{1}{3h^2}\left[(t-t_{j-1})^3\right]_{t_{j-1}}^{t_j} \\ &= \frac{1}{3h^2}(t_j - t_{j-1})^3 = \frac{h^3}{3h^2} = \frac{h}{3}\end{aligned} \tag{5.56}$$

och

$$\begin{aligned}\int_{I_j} \varphi_{j-1}\varphi_j\,dt &= \int_{I_j} \Big(\frac{t_j - t}{h}\Big)\Big(\frac{t-t_{j-1}}{h}\Big)\,dt \\ &= \frac{1}{h^2}\left[\frac{t_j t^2}{2} - t_{j-1}t_j t - \frac{t^3}{3} + \frac{t_{j-1}t^2}{2}\right]_{t_{j-1}}^{t_j} \\ &= \frac{1}{6h^2}\left(t_j^3 - t_{j-1}^3 - 3t_{j-1}t_j^2 + 3t_{j-1}^2 t_j\right) = \frac{1}{6h^2}(t_j - t_{j-1})^3 = \frac{h}{6}.\end{aligned} \tag{5.57}$$

Antag nu att A, B, Q och R i (5.48), (5.49) och (5.50) är oberoende av t. Ekvation (5.52) resulterar i följande ekvationer för $j = 1, \ldots, N$:

$$\begin{aligned}&\int_0^T \Big(2Q\sum_{i=0}^{N+1}\xi_i\phi_i - \sum_{i=0}^{N}\eta_i\dot{\varphi}_i - A\sum_{i=0}^{N}\eta_i\varphi_i\Big)\phi_j\,dt \\ &= \int_{I_j}\Big(2Q\xi_j\phi_j - \eta_{i-1}\dot{\varphi}_{i-1} + \eta_i\dot{\varphi}_i) - A(\eta_{i-1}\varphi_{i-1} + \eta_i\varphi_i)\Big)\phi_j\,dt \\ &= \int_{I_j}\Big(2Q\xi_j\phi_j^2 - (-\frac{1}{h}\eta_{i-1} + \frac{1}{h}\eta_i)\phi_j - A(\eta_{i-1}\varphi_{i-1} + \eta_i\varphi_i)\phi_j\Big)\,dt \\ &= 2Qh\xi_j + (1 - \frac{Ah}{2})\eta_{j-1} + (-1 - \frac{Ah}{2})\eta_j = 0.\end{aligned} \tag{5.58}$$

I beräkningen har integralerna (5.56) och (5.57) använts, samt det faktum att $\phi_j = 1$ på I_j. De två sista termerna i (5.52) är nollskilda endast då $j = 0$ eller $j = N+1$ och blir då

$$(I - I_0)(2S_0\xi_0 - \eta_0) = 0, \quad (I - I_T)(2S_T\xi_{N+1} + \eta_N) = 0. \tag{5.59}$$

Notera här att om $I_0 = 1$, är ξ_0 given och om $I_T = 1$, är ξ_T given.

Ekvation (5.53) resulterar på liknande sätt i följande ekvationer för $j = 1, \ldots, N-1$:

$$\begin{aligned}&\int_0^T \Big(2R\sum_{i=0}^{N}\zeta_i\varphi_i - B\sum_{i=0}^{N}\eta_i\varphi_i\Big)\varphi_j\,dt \\ &= \int_{I_j}\Big(2R(\zeta_{j-1}\varphi_{j-1} + \zeta_j\varphi_j)\varphi_j - B(\eta_{j-1}\varphi_{j-1} + \eta_j\varphi_j)\varphi_j\Big)\,dt \\ &\quad + \int_{I_{j+1}}\Big(2R(\zeta_j\varphi_j + \zeta_{j+1}\varphi_{j+1})\varphi_j - B(\eta_j\varphi_j + \eta_{j+1}\varphi_{j+1})\varphi_j\Big)\,dt \\ &= \frac{Rh}{3}\zeta_{j-1} + \frac{4Rh}{3}\zeta_j + \frac{Rh}{3}\zeta_{j+1} - \frac{Bh}{6}\eta_{j-1} - \frac{2Bh}{3}\eta_j - \frac{Bh}{6}\eta_{j+1}.\end{aligned} \tag{5.60}$$

För $j = 0$ blir (5.53):

$$\begin{aligned}&\int_{I_1}\Big(2R(\zeta_0\varphi_0 + \zeta_1\varphi_1)\varphi_0 - B(\eta_0\varphi_0 + \eta_1\varphi_1)\varphi_0\Big)\,dt \\ &\quad = \frac{2Rh}{3}\zeta_0 + \frac{Rh}{3}\zeta_1 - \frac{Bh}{3}\eta_0 - \frac{Bh}{6}\eta_1.\end{aligned} \tag{5.61}$$

För $j = N$ blir (5.53):

$$\int_{I_N} \Big(2R(\zeta_{N-1}\varphi_{N-1} + \zeta_N\varphi_N)\varphi_N - B(\eta_{N-1}\varphi_{N-1} + \eta_N\varphi_N)\varphi_N\Big)\,dt$$
$$= \frac{2Rh}{3}\zeta_{N-1} + \frac{Rh}{3}\zeta_N - \frac{Bh}{3}\eta_{N-1} - \frac{Bh}{6}\eta_N. \tag{5.62}$$

Slutligen resulterar (5.54) i följande ekvationer för $j = 1, \ldots, N$:

$$-\int_0^T \Big(A\sum_{i=0}^{N+1}\xi_i\phi_i + B\sum_{i=0}^{N}\zeta_i\varphi_i\Big)\varphi_j\,dt + (\xi_1 - \xi_0)\varphi_j(0)$$
$$+ (\xi_2 - \xi_1)\varphi_j(t_1) + \cdots + (\xi_{N+1} - \xi_N)\varphi_j(t_N)$$
$$= -\int_{I_j}\Big(A\xi_j\phi_j\varphi_j + B(\zeta_{j-1}\varphi_{j-1} + \zeta_j\varphi_j)\varphi_j\Big)\,dt$$
$$-\int_{I_{j+1}}\Big(A\xi_{j+1}\phi_{j+1}\varphi_j + B(\zeta_j\varphi_j + \zeta_{j+1}\varphi_{j+1})\varphi_j\Big)\,dt + \xi_{j+1} - \xi_j$$
$$= (-1 - \frac{Ah}{2})\xi_j + (1 - \frac{Ah}{2})\xi_{j+1} - \frac{Bh}{6}\zeta_{j-1} - \frac{2Bh}{3}\zeta_j - \frac{Bh}{6}\zeta_{j+1}. \tag{5.63}$$

För $j = 0$ blir (5.54):

$$-\int_{I_1}\Big(A\xi_1\phi_1\varphi_0 + B(\zeta_0\varphi_0 + \zeta_1\varphi_1)\varphi_0\Big)\,dt + \xi_1 - \xi_0$$
$$= -\xi_0 + (1 - \frac{Ah}{2})\xi_1 - \frac{Bh}{3}\zeta_0 - \frac{Bh}{6}\zeta_1. \tag{5.64}$$

För $j = N$ blir (5.54):

$$-\int_{I_N}\Big(A\xi_N\phi_N\varphi_N + B(\zeta_{N-1}\varphi_{N-1} + \zeta_N\varphi_N)\varphi_N\Big)\,dt + \xi_{N+1} - \xi_N$$
$$= (-1 - \frac{Ah}{2})\xi_N + \xi_{N+1} - \frac{Bh}{6}\zeta_{N-1} - \frac{Bh}{3}\zeta_N. \tag{5.65}$$

Sammantaget fås följande matrisekvation:

$$\begin{bmatrix} \mathcal{Q} & 0 & \mathcal{A}^T \\ 0 & \mathcal{R} & \mathcal{B}^T \\ \mathcal{A} & \mathcal{B} & 0 \end{bmatrix} x = b,$$

där $\mathcal{Q}$ är en $(N+2)\times(N+2)$-matris med följande utseende:

$$\mathcal{Q} = \begin{bmatrix} ((I - I_0)2S_0 + I_0) & 0 & 0 & \ldots & 0 & 0 \\ 0 & 2Qh & 0 & \ldots & 0 & 0 \\ 0 & 0 & 2Qh & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ldots & 0 & 2Qh & 0 \\ 0 & 0 & \ldots & 0 & 0 & ((I - I_T)2S_T - I_T) \end{bmatrix}.$$

$\mathcal{R}$ och $\mathcal{B}$ är $(N+1)\times(N+1)$-matriser med följande utseende:

$$\mathcal{R} = \begin{bmatrix} \frac{2Bh}{3} & \frac{Rh}{3} & 0 & 0 & \dots & 0 \\ \frac{Rh}{3} & \frac{4Rh}{3} & \frac{Rh}{3} & 0 & \dots & 0 \\ 0 & \frac{Rh}{3} & \frac{4Rh}{3} & \frac{Rh}{3} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{Rh}{3} & \frac{4Rh}{3} & \frac{Rh}{3} \\ 0 & \dots & 0 & 0 & \frac{Rh}{3} & \frac{2Rh}{3} \end{bmatrix}$$

och

$$\mathcal{B} = \begin{bmatrix} \frac{-Bh}{3} & \frac{-Bh}{6} & 0 & 0 & \dots & 0 \\ \frac{-Bh}{6} & \frac{-2Bh}{3} & \frac{-Bh}{6} & 0 & \dots & 0 \\ 0 & \frac{-Bh}{6} & \frac{-2Bh}{3} & \frac{-Bh}{6} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{-Bh}{6} & \frac{-2Bh}{3} & \frac{-Bh}{6} \\ 0 & \dots & 0 & 0 & \frac{-Bh}{6} & \frac{-Bh}{3} \end{bmatrix}.$$

$\mathcal{A}$ känns igen från avsnitt 5.2.1 och är en ickekvadratisk matris av storlek $(N+1)\times(N+2)$ och har följande uppbyggnad:

$$\mathcal{A} = \begin{bmatrix} -(I-I_0) & (1-\frac{Ah}{2}) & 0 & 0 & \dots & 0 \\ 0 & -(1+\frac{Ah}{2}) & (1-\frac{Ah}{2}) & 0 & \dots & 0 \\ 0 & 0 & -(1+\frac{Ah}{2}) & (1-\frac{Ah}{2}) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -(1+\frac{Ah}{2}) & (1-\frac{Ah}{2}) & 0 \\ 0 & \dots & 0 & 0 & -(1+\frac{Ah}{2}) & (I-I_0) \end{bmatrix}.$$

Variabelvektorn x består av:

$$x = \begin{bmatrix} \xi_0 \\ \vdots \\ \xi_{N+1} \\ \zeta_0 \\ \vdots \\ \zeta_N \\ \eta_0 \\ \vdots \\ \eta_N \end{bmatrix}.$$

Vektorn b är av samma storlek som x och har endast följande nollskilda element:

$$b_1 = b_{2N+4} = I_0 x_0, \qquad b_{N+2} = b_{3N+4} = -I_T x_T.$$

5.3.4 Problemet i FEniCS

Den finita elementmetod som redovisats i tidigare avsnitt implementeras här i FEniCS. Programmet som följer i detta avsnitt löser kvadratisk-linjära styrproblem (se (5.32) och (5.33)) med $d = m = 1$. Den uträknade matrisen ovan och den diskussion som följer gäller för $q = 0$, men programmet fungerar även för högre q.

Först i koden väljs antalet intervall N och sluttiden T och partitioneringen skapas utifrån det. Heltalet q beskriver gradtalet hos polynomen i funktionsrummen.

```
N = 50
T = 1
mesh = Interval(N, 0, T)
q = 0
```

Därefter anges de koefficienter som specifierar problemet. Här är de valda på samma sätt som i Exempel 2 i avsnitt 5.3.2. Notera att det inte spelar någon roll vad x_0 sätts till då $I_0 = 0$. Detsamma gäller för x_T då $I_T = 0$.

```
I_0 = 0
I_T = 1
At = Expression("1.0")
Bt = Expression("1.0")
Rt = Expression("0.5")
Qt = Expression("0.5")
S_0 = Constant(0.5)
S_T = Constant(0.0)
x_0 = Constant(0.0)
x_T = Constant(1.0)
```

Sedan definieras funktionsrummen. Märk väl att det W_h som skapas i FEniCS inte innehåller de yttre randvärdena w_0^- och w_N^+. Detta kommer medföra komplikationer senare i programmet.

```
W_h = FunctionSpace(mesh, "DG", q)
U_h = FunctionSpace(mesh, "CG", q + 1)
V_h = FunctionSpace(mesh, "CG", q + 1)
```

I nästa steg sätts de tre funktionsrummen samman till ett trippelrum.

```
M_h = MixedFunctionSpace([W_h, U_h, V_h])
```

I ekvation (5.50) summeras över hoppen $[x_h]_n$. I FEniCS åtskiljs inre och yttre fasetter. För att kunna beräkna hoppen i randpunkterna 0 och T skapas därför klasser som lokaliserar randen (se avsnitt 5.2.4 för en djupare förklaring). Detta görs enligt:

```
class InitialBoundary(SubDomain):
        def inside(self, x, on_boundary):
            return on_boundary and abs(x[0]) < DOLFIN_EPS

class EndBoundary(SubDomain):
        def inside(self, x, on_boundary):
            return on_boundary and abs(x[0] - T) < DOLFIN_EPS
```

Nedanstående två rader kopplar samman partitioneringen med ovanstående klasser.

```
facet_dimension = mesh.topology().dim() - 1
boundary_parts = MeshFunction("uint", mesh, facet_dimension)
```

Av klasserna ovan skapas sedan följande instanser:

```
start_time = InitialBoundary()
start_time.mark(boundary_parts, 0)
final_time = EndBoundary()
final_time.mark(boundary_parts, 1)
```

Notera att hoppen i randpunkterna i teorin är $\xi_1 - \xi_0$ och $\xi_{N+1} - \xi_N$. Eftersom ξ_0 och ξ_{N+1} inte finns med i FEniCS:s `dG`-rum blir de endast ξ_1 och $-\xi_N$ i FEniCS.

De olika trial- och testfunktionerna skapas enligt

```
(x, u, z) = TrialFunctions(M_h)
(p_x, p_u, p_z) = TestFunctions(M_h)
```

Variationsformuleringen definieras i FEniCS på följande sätt

```
C = Expression("0.0")

a = inner(p_x, 2*Qt*x - grad(z) - At*z)*dx + inner(p_u, 2*Rt*u - Bt*z)*dx
     + inner(grad(x) - At*x - Bt*u, p_z)*dx - inner(jump(x), avg(p_z))*dS
    - inner(x, p_z)*ds(1) + inner(x, p_z)*ds(0)
L = C*p_x*dx
```

`C`-uttrycket krävs då FEniCS inte accepterar en variationsformulering på formen `L = 0` eller `L = 0*p_x*dx`, utan den måste var enligt ovan för att `L` senare ska bli en vektor.

Med hjälp av `assemble` skapas sedan finita elementekvationens matris och vektor:

```
M = assemble(a, exterior_facet_domains = boundary_parts)
b = assemble(L, exterior_facet_domains = boundary_parts)
```

Detta resulterar i ekvationssystemet $Mx^* = b^*$, där $b^* = 0$ har samma dimension som x^* och

$$M = \begin{bmatrix} \mathcal{Q}^* & 0 & (\mathcal{A}^*)^T \\ 0 & \mathcal{R} & \mathcal{B}^T \\ \mathcal{A}^* & \mathcal{B} & 0 \end{bmatrix},$$

där $\mathcal{Q}^*$ är av storlek $N \times N$ och har följande struktur:

$$\mathcal{Q}^* = \begin{bmatrix} 2Qh & 0 & \dots & 0 \\ 0 & 2Qh & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 2Qh \end{bmatrix}.$$

$\mathcal{A}^*$ är av storlek $(N+1) \times N$ och har följande struktur:

$$\mathcal{A}^* = \begin{bmatrix} (1 - a\frac{h}{2}) & 0 & \cdots & \cdots & 0 \\ -(1 + a\frac{h}{2}) & (1 - a\frac{h}{2}) & 0 & \cdots & 0 \\ 0 & -(1 + a\frac{h}{2}) & (1 - a\frac{h}{2}) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -(1 + a\frac{h}{2}) & (1 - a\frac{h}{2}) \\ 0 & \cdots & \cdots & 0 & -(1 + a\frac{h}{2}) \end{bmatrix}.$$

Variabelvektorn x^* är på följande form:

$$x^* = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_N \\ \zeta_0 \\ \vdots \\ \zeta_N \\ \eta_0 \\ \vdots \\ \eta_N \end{bmatrix}.$$

Att matrisen som genereras av FEniCS inte helt överensstämmer med teorin beror på att de yttre randvärden, x_0^- och x_N^+, utelämnas i FEniCS:s funktionsrum. Här krävs alltså att matrisen och vektorn justeras för att lösningen ska bli korrekt. För att kunna ändra i matrisen och vektorn objektomvandlas de enligt:

```
M = M.array()
b = b.array()
```

För att justeringen av matrisen ska bli smidigare undviker vi att ändra dess dimension. Den information som saknas är

$$(I - I_0)(2S_0\xi_0 - \eta_0) = 0, \tag{5.66}$$
$$(I - I_T)(2S_T\xi_{N+1} + \eta_N) = 0, \tag{5.67}$$
$$I_0\xi_0 = x_0, \tag{5.68}$$
$$I_T\xi_{N+1} = x_T. \tag{5.69}$$

Om $I_0 = 1$ elimineras ekvation (5.66) och $\xi_0 = x_0$ blir given. Om däremot $I_0 = 0$ elimineras ekvation (5.68), och (5.66) blir $2S_0\xi_0 - \eta_0 = 0$ där ξ_0 och η_0 är okända. Alltså måste matrisen och vektorn justeras beroende på vilket fall som gäller. Nedan beskrivs vilka justeringar som ska göras i respektive fall.

Då $I_0 = 1$ sätts $\xi_0 = x_0$ in i ekvationen som motsvarar första raden i $\mathcal{A}$ från avsnitt 5.3.3. x_0 flyttas sedan över till högerledet. Då överenstämmer vänsterledet med första raden i $\mathcal{A}^*$. Följande förändring i $b = (b_1, \ldots, b_{3N+2})$ görs:

$$b_{2N+2} = x_0.$$

Om istället $I_0 = 0$ sätts $\xi_0 = \frac{\eta_0}{2S_0}$ in i ekvationen vilken motsvarar första raden i $\mathcal{A}$. Detta innebär att koefficienten för η_0 i M måste ändras. Förändringen blir:

$$M(2N+2, 2N+2) = -\frac{1}{2S_0}.$$

Med liknande resonemang för I_T blir motsvarande förändringar

$$b_{3N+2} = -x_T$$

och

$$M(3N+2, 3N+2) = -\frac{1}{2S_T}.$$

Ovanstående blir med FEniCS-kod:

```
if I_0 == 1:
    b[2*(q + 1)*N + 1] = x_0(0)
else:
    M[(2*(q + 1)*N + 1), (2*(q + 1)*N + 1)] = -1.0/(2.0*S_0(0))

if I_T == 1:
    b[2*(q + 1)*N + N + 1] = -x_T(0)
else:
    M[(2*(q + 1)*N + N + 1), (2*(q + 1)*N + N + 1)] = -1.0/(2.0*S_T(0))
```

I nästa steg löses ekvationssystemet

```
solution = linalgSolve(M, b)
```

Lösningsvektorn delas upp i de tre sökta lösningarna

```
x_ = solution[0 : (q + 1)*N]
u_ = solution[(q + 1)*N : (2*(q + 1)*N + 1)]
z_ = solution[(2*(q + 1)*N + 1) : (3*(q + 1)*N + 2)]
```

där x_= $(\xi_1, \ldots, \xi_N)$, u_= $(\zeta_0, \ldots, \zeta_N)$ och z_= $(\eta_0, \ldots, \eta_N)$.

För att kunna generera FEniCS-funktioner av lösningarna skapas nya vektorer av annorlunda format och värdena kopieras till dessa enligt:

```
x = uBLASVector((q + 1)*N)
x.zero()
for i in range((q + 1)*N):
   insertIndex = array([i], dtype = "I")
   toInsert = array(x_[i])
   x.add(toInsert, insertIndex)

u = uBLASVector((q + 1)*N + 1)
u.zero()
for i in range((q + 1)*N + 1):
   insertIndex = array([i], dtype = "I")
   toInsert = array(u_[i])
   u.add(toInsert, insertIndex)

z = uBLASVector((q + 1)*N + 1)
z.zero()
for i in range((q + 1)*N + 1):
   insertIndex = array([i], dtype = "I")
   toInsert = array(z_[i])
   z.add(toInsert, insertIndex)
```

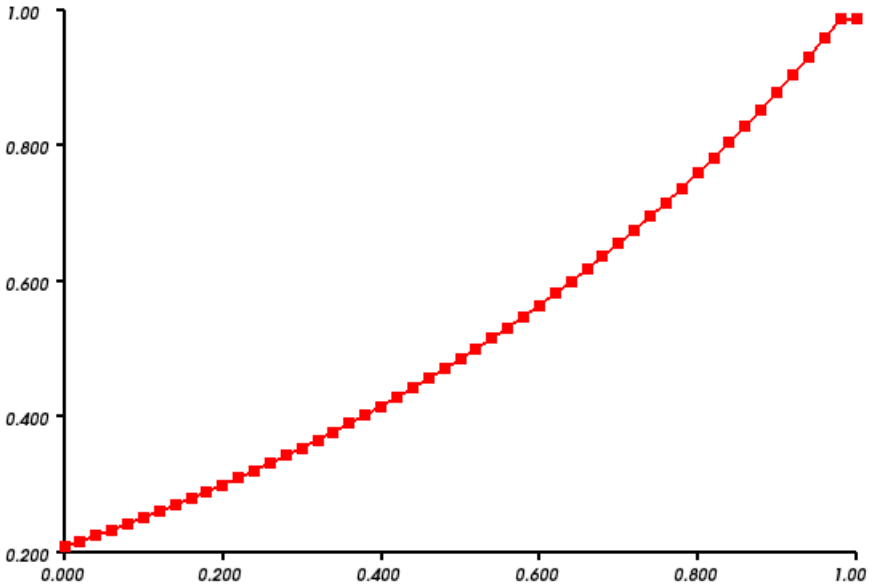
Därefter interpoleras vektorvärden till funktioner med hjälp av klassen `Function`

```
x = Function(W_h, x)
u = Function(U_h, u)
z = Function(V_h, z)
```

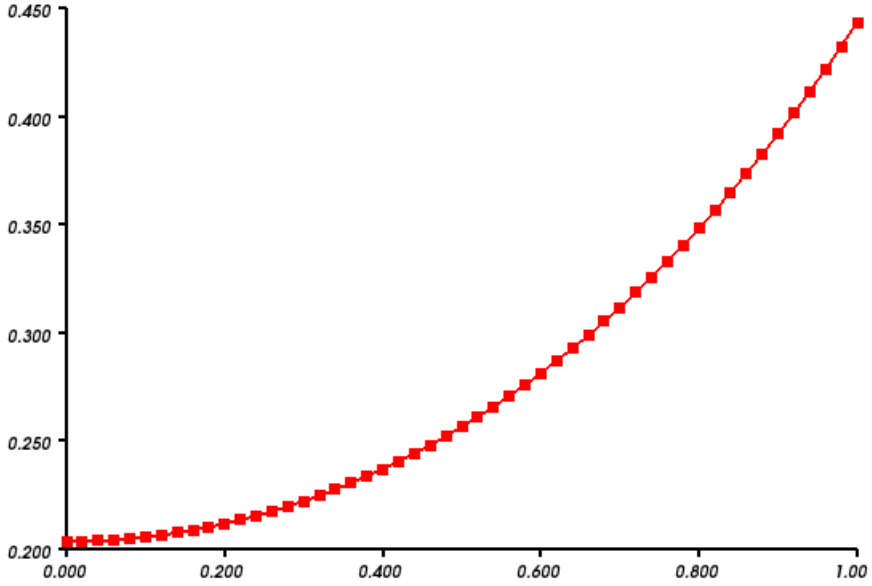
Slutligen plottas funktionerna

```
plot(x, title = "x")
plot(u, title = "u")
plot(z, title = "z")
interactive()
```

Se Figur 5.8 respektive 5.9 för de plottar FEniCS-programmet ovan genererar för $x(t)$ respektive $u(t)$.



Figur 5.8: Tillståndsfunktionen $x(t)$ för ett styrproblem med koefficienter som Exempel 2 i avsnitt 5.3.2.



Figur 5.9: Styrfunktionen $u(t)$ för ett styrproblem med koefficienter som Exempel 2 i avsnitt 5.3.2.

5.3.5 Fullständig kod

Listing 3: Kvadratisk-linjärt styrproblem

```
"""
This program solves a quadratic-linear optimal control problem with a
    finite element method.
"""

from dolfin import *
from numpy import *
```

```
from numpy.linalg import solve as linalgSolve

N = 50 #Number of sub intervals.
T = 1 #Final time.
mesh = Interval(N, 0, T) #Mesh of N subintervals on the interval (0, T).
q = 0 #Degree of basis polynomials.

#Coefficients to specify the problem.
I_0 = 0
I_T = 1
At = Expression("1.0")
Bt = Expression("1.0")
Rt = Expression("0.5")
Qt = Expression("0.5")
S_0 = Constant(0.5)
S_T = Constant(0.0)
x_0 = Constant(0.0)
x_T = Constant(1.0)

#Definitions of function spaces.
W_h = FunctionSpace(mesh, "DG", q)
U_h = FunctionSpace(mesh, "CG", q + 1)
V_h = FunctionSpace(mesh, "CG", q + 1)

M_h = MixedFunctionSpace([W_h, U_h, V_h])

#Classes to identify the boundaries.
class InitialBoundary(SubDomain):
   def inside(self, x, on_boundary):
      return on_boundary and abs(x[0]) < DOLFIN_EPS

class EndBoundary(SubDomain):
   def inside(self, x, on_boundary):
      return on_boundary and abs(x[0] - T) < DOLFIN_EPS

#Creates a mesh function to represent subdomains of the boundary.
facet_dimension = mesh.topology().dim() - 1
boundary_parts = MeshFunction("uint", mesh, facet_dimension)

#Creates instance of the boundary classes.
start_time = InitialBoundary()
start_time.mark(boundary_parts, 0)
final_time = EndBoundary()
final_time.mark(boundary_parts, 1)

#Defines test and trial functions.
(x, u, z) = TrialFunctions(M_h)
(p_x, p_u, p_z) = TestFunctions(M_h)

#Rhs in the abstract formulation is zero.
C = Expression("0.0")

#Defines the abstract formulation of the problem.
a = inner(p_x, 2*Qt*x - grad(z) - At*z)*dx + inner(p_u, 2*Rt*u - Bt*z)*dx
     + inner(grad(x) - At*x - Bt*u, p_z)*dx - inner(jump(x), avg(p_z))*dS
    - inner(x, p_z)*ds(1) + inner(x, p_z)*ds(0)
L = C*p_x*dx

#Assembles system and connects the ds(i) symbols to A and L.
M = assemble(a, exterior_facet_domains = boundary_parts)
b = assemble(L, exterior_facet_domains = boundary_parts)
```

```
#Defines NumPy matrix and vector to modify according to the method in
    Kraft's thesis.
M = M.array()
b = b.array()

#Adjusts the proper elements in the matrix.
if I_0 == 1:
    b[2*(q + 1)*N + 1] = x_0(0)
else:
    M[(2*(q + 1)*N + 1), (2*(q + 1)*N + 1)] = -1.0/(2.0*S_0(0))

if I_T == 1:
    b[2*(q + 1)*N + N + 1] = -x_T(0)
else:
    M[(2*(q + 1)*N + N + 1), (2*(q + 1)*N + N + 1)] = -1.0/(2.0*S_T(0))

solution = linalgSolve(M, b) #Solves the problem using NumPy's solve.

#Separates the solutions.
x_ = solution[0 : (q + 1)*N]
u_ = solution[(q + 1)*N : (2*(q + 1)*N + 1)]
z_ = solution[(2*(q + 1)*N + 1) : (3*(q + 1)*N + 2)]

#Create object of FEniCS objects representing the solutions.
x = uBLASVector((q + 1)*N)
x.zero()
for i in range((q + 1)*N):
   insertIndex = array([i], dtype = "I")
   toInsert = array(x_[i])
   x.add(toInsert, insertIndex)

u = uBLASVector((q + 1)*N + 1)
u.zero()
for i in range((q + 1)*N + 1):
   insertIndex = array([i], dtype = "I")
   toInsert = array(u_[i])
   u.add(toInsert, insertIndex)

z = uBLASVector((q + 1)*N + 1)
z.zero()
for i in range((q + 1)*N + 1):
   insertIndex = array([i], dtype = "I")
   toInsert = array(z_[i])
   z.add(toInsert, insertIndex)

#Creates FEniCS Functions of the solutions.
x = Function(W_h, x)
u = Function(U_h, u)
z = Function(V_h, z)

plot(x, title = "x")
plot(u, title = "u")
plot(z, title = "z")
interactive()
```

6 Diskussion

I det här projektet har vi implementerat en finita elementmetod för lösning av optimala styrproblem i FEniCS och undersökt hur väl lämpad programvaran är för ändamålet. Metoden är hämtad från Krafts doktorsavhandling *Adaptive Finite Element Methods for Optimal Control Problems* [2]. Under arbetets gång har vi stött på två huvudsakliga problem med implementationen av metoden. Dels att vissa av de funktionsrum som finns i FEniCS inte överensstämmer med metodens, dels att randvillkoren ignoreras då en diskontinuerlig Galerkinmetod används. Problemen beror till stor del på att FEniCS i första hand är till för partiella differentialekvationer, medan den metod vi använder för att lösa optimala styrproblem resulterar i ett system av ordinära differentialekvationer.

De diskontinuerliga funktionsrummen i finita elementmetoden Kraft beskriver innehåller två randvärden som ligger utanför det egentliga intervallet. FEniCS tillhandahåller en annan typ av funktionsrum som inte innefattar dessa två yttre randvärden. Därför kommer den matris som FEniCS räknar fram inte stämma överens med den från teorin Kraft använder. I avsnitt 5.2 ger FEniCS en icke-kvadratisk matris, medan den enligt teorin ska vara kvadratisk. I avsnitt 5.3 är problemet ett annat. Systemet blir kvadratiskt, men storleken avviker från den teoretiskt förväntade.

Problemet med randvillkoren tycks bero på att FEniCS inte stödjer automatisk applicering av randvillkor på diskontinuerliga funktionsrum. Att randvillkoren inte appliceras gör att högerledet i matrisekvationen blir en nollvektor. Detta resulterar alltid i en lösning som är noll överallt, även om lösningen till det ursprungsliga problemet är en annan.

Ovanstående två problem löste vi genom att modifiera matriserna från FEniCS så att de överensstämde med teorin. För att göra detta krävdes att matris och vektor typomvandlades, men då kunde inte FEniCS inbyggda lösningsmetod längre användas. Istället använde vi Pythonbiblioteket NumPy:s ekvationslösare och konverterade sedan lösningen till ett objekt av FEniCS-typ.

Från lösningsproceduren ovan har vi skapat oss en uppfattning om FEniCS styrkor och svagheter. Bortsett från den beskrivna problematiken har vi funnit att FEniCS tillhandahåller funktionaliteter som underlättar och automatiserar vissa steg i implementationen av metoden från Krafts avhandling. Bland annat finns i FEniCS inbyggda klasser för att definiera funktionsrum som är centrala i finita elementmetoder. Dessa klasser gör det enkelt att variera partitionering av intervall och gradtal hos basfunktioner. Det sistnämnda är ett av de förslag på fortsatta studier som Kraft ger i sin doktorsavhandling.

En annan fördel med FEniCS är den automatisering av lösningsprocessen som programvaran stödjer. Genom att sätta upp problemet på variationsform beräknar FEniCS automatiskt den matrisekvation som finita elementmetoden resulterar i.

Vi har slutligen funnit att en önskvärd funktionalitet i FEniCS skulle vara möjligheten att konstruera egna funktionsrum, eller åtminstone modifiera dem som redan finns. I vårt fall skulle vi haft användning av att kunna inkludera basfunktioner med stöd utanför det intervall som utgjorde definitionsmängd i problemet.

I början av projektet var tanken att vi även skulle jämföra FEniCS med annan programvara samt skriva program för att lösa optimala styrproblem med en adaptiv finita elementmetod. Det visade sig dock att implementation av metoden icke-adaptivt i FEniCS var en tillräckligt omfattande studie. Däremot kan detta vara intressanta frågeställningar för framtida projekt inom området.

Några ytterligare ämnen för fortsatta studier kan vara att implementera de metoder Kraft använder för ickelinjära styrproblem och för styrproblem med olikhetsbivillkor. Dessutom kan det vara av intresse att titta på möjligheten att modifiera den diskontinuerliga finita elementmetod som FEniCS stödjer. Även om det inte är möjligt just nu finns chansen att det kommer fungera i framtiden, då FEniCS är en relativt ny programvara under utveckling.

7 Slutsats

Vi har funnit att FEniCS är ett lämpligt verktyg för lösning av optimala styrproblem med finita elementmetoden från Krafts doktorsavhandling, även om en del modifikationer i mjukvaran hade förenklat lösningsproceduren. Då programvaran är relativt ny och under utveckling hoppas vi att detta kommer finnas med i senare versioner av FEniCS.

För framtida studier av optimala styrproblem i FEniCS föreslår vi arbete med:

- Modifiering av funktionsrum.
- Applicering av randvillkor för diskontinuerliga metoder.
- Adaptiv förfining av partitionering.
- Ickelinjära styrproblem.
- Styrproblem med olikhetsbivillkor.

A Matematiska definitioner

Definition A.1. *För i, j positiva heltal definieras* ***Kroneckers delta*** *som*

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & annars. \end{cases}$$

Definition A.2. *En* ***funktional*** *är en avbildning*

$$\mathcal{F} : X \longrightarrow \mathbb{R},$$

där X är ett vektorrum och $\mathbb{R}$ är dess skalärrum [11].

Definition A.3. *En* ***binärt diagonal matris*** *$A \in \mathbb{R}^{m\times m}$ är en kvadratisk matris på följande form:*

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{mm} \end{bmatrix},$$

där $a_{ii} \in \{0, 1\}$, $i = 1, \dots, m$.

Definition A.4. *En funktion f sägs vara en* ***glatt funktion*** *om*

$$f \in C^p,$$

där $p \in \mathbb{N}$ är tillräckligt stor i sammanhanget.

Definition A.5. *En* ***Langrangemultiplikator*** *används för att skriva om ett optimeringsproblem på formen*

$$\begin{aligned} \text{minimera} \quad & f(x), \\ \text{under bivillkoret} \quad & g(x) = 0, \end{aligned}$$

till det ekvivalenta problemet

$$\text{minimera} \quad f(x) + \lambda g(x),$$

där λ är Lagrangemultiplikatorn [17].

Definition A.6. *Ett* ***Sobolevrum*** *$H^k(\Omega, \mathbb{R}^n)$ är ett vektorrum av funktioner $f(x)$ där $f \in \mathbb{R}^n$ och $x \in \Omega \subset \mathbb{R}^d$ som definieras*

$$H^k(\Omega, \mathbb{R}^n) = \{f \in L^2(\Omega) : \partial_x^\alpha f \in L^2(\Omega), \quad \forall |\alpha| \leq k\},$$

där $\alpha = (\alpha_1, \dots, \alpha_d)$, $|\alpha| = \alpha_1 + \dots + \alpha_d$, $\partial_x^\alpha = \partial_{x_1}^{\alpha_1} \dots \partial_{x_d}^{\alpha_d}$ och $L^2(\Omega) = \{u : (\int_\Omega |u|^2 dx)^{1/2} < \infty\}$ [7].

Definition A.7. ***Skalärprodukten*** *(u, v) mellan två vektorer $u = [u_1, \ldots, u_n]$ och $v = [v_1, \ldots, v_n]$ defineras enligt* [6]

$$(u, v) = \sum_{i=1}^{n} u_i v_i.$$

Definition A.8. ***Fréchetderivatan*** *G av en funktional $F(x)$ verkande på testfunktionen h är en linjär operator som uppfyller*

$$F(x + h) = F(x) + Gh + \epsilon(h),$$

där

$$\lim_{\|h\| \to 0} \frac{\|\epsilon(h)\|}{\|h\|} = 0.$$

Detta G betecknas $F'(x)h$ [12].

Definition A.9. *En reell matris $A \in R^{m \times m}$ kallas* ***positivt definit*** *om*

$$x^T A x > 0, \quad \forall x \in R^m.$$

Definition A.10. *En reell matris $A \in R^{m \times m}$ kallas* ***positivt semidefinit*** *om*

$$x^T A x \geq 0, \quad \forall x \in R^m.$$

Definition A.11. *En* ***polytop*** *är ett geometriskt objekt som utgör en delmängd av $\mathbb{R}^n$ och vars rand utgörs av ett ändligt antal hyperplan. Det är således en generalisering av objekt som polygon och polyedrar till godtycklig dimension* [9].

Definition A.12. *Antag en n-dimensionell polytop*[16]*. En* ***fasett*** *är då en $(n-1)$-dimensionell polytop som utgör en del av den n-dimensionella polytopens rand* [10].

Definition A.13. *En följd $\{x_i\}_{i=1}^{\infty}$ är en* ***Cauchyföljd*** *om det för varje $\epsilon > 0$ finns ett positivt heltal N som uppfyller* [1]

$$|x_n - x_m| < \epsilon, \quad \forall\, m, n > N.$$

Definition A.14. *Ett* ***Banachrum*** *är ett normerat vektorrum som är komplett med avseende på denna norm. Det innebär att varje Cauchyföljd*[17] *i rummet konvergerar mot ett gränsvärde i rummet* [8].

Definition A.15. *En funktion $f(x)$ sägs vara $\mathbf{o}(x^p)$ då $x \to 0$ om* [13]

$$\frac{f(x)}{x^p} \to 0, \quad x \to 0.$$

[16] För definition av polytop se Definition A.11 i detta avsnitt.
[17] För definition av Cauchyföljd se Definition A.13 i detta avsnitt.

Referenser

[1] Eriksson, K. et al. *Computational Differential Equations*. Lund: Studentlitteratur. 1996.

[2] Kraft, K. *Adaptive Finite Element Methods for Optimal Control Problems*. Göteborg: Chalmers tekniska högskola. 2011.

[3] Heath, M. *Scientific Computing: An Introductory Survey*. New York: McGraw-Hill Companies. 2005.

[4] Asadzadeh, M. *An Introduction to the Finite Element Method (FEM) for Differential Equations*. Chalmers: Lecture notes. 2012.

[5] Jönsson, U. *Varför läsa optimal styrteori?*. `http://www.math.kth.se/optsyst/grundutbildning/5B1872/5B1872reklam.html` (tillträdd 29 april 2012).

[6] Wolfram MathWorld. *Dot Product*. `http://mathworld.wolfram.com/DotProduct.html` (tillträdd 5 april 2012).

[7] Wolfram MathWorld. *Sobolev Space*. `http://mathworld.wolfram.com/SobolevSpace.html` (tillträdd 3 april 2012).

[8] Wolfram MathWorld. *Banach Space*. `http://mathworld.wolfram.com/BanachSpace.html` (tillträdd 6 april 2012).

[9] Wolfram MathWorld. *Polytope*. `http://mathworld.wolfram.com/Polytope.html` (tillträdd 6 april 2012).

[10] Wolfram MathWorld. *Facet*. `http://mathworld.wolfram.com/Facet.html` (tillträdd 6 april 2012).

[11] Encyclopedia of Mathematics. *Functional*. `http://www.encyclopediaofmath.org/index.php/Functional` (tillträdd 27 mars 2012).

[12] Encyclopedia of Mathematics. *Fréchet derivative*. `http://www.encyclopediaofmath.org/index.php/Frechet\_derivative` (tillträdd 6 april 2012).

[13] Råde, L, Westergren, B. *Mathematics Handbook for Science and Engineering*. Femte upplagan. Lund: Studentlitteratur. 2004.

[14] Numpy developers. *NumPy*. `http://numpy.scipy.org/` (tillträdd 6 april 2012).

[15] Logg, A. Mardal, K-A. Wells, G-N. *Automated Solution of Differential Equations by the Finite Element Method, The FEniCS Book*, Chicago: Springer. 2011.

[16] FEniCS Project. *Documentation*. `http://fenicsproject.org/documentation/dolfin/1.0.0/python/genindex.html` (tillträdd 22 april 2012).

[17] Encyclopedia of Mathematics. *Lagrange multipliers*. `http://www.encyclopediaofmath.org/index.php/Lagrange_multipliers` (tillträdd 1 maj 2012).